CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

EDUARDO CAMARGO DE SIQUEIRA

HEURÍSTICAS COMPUTACIONAIS PARA RESOLUÇÃO DE UM PROBLEMA *FLOW SHOP* HÍBRIDO MULTIOBJETIVO

BELO HORIZONTE
Março, 2019

Centro Federal de Educação Tecnológica de Minas Gerais
Diretoria de Pesquisa e Pós-Graduação
Programa de Pós-Graduação em Modelagem Matemática e Computacional

HEURÍSTICAS COMPUTACIONAIS PARA RESOLUÇÃO DE UM PROBLEMA *FLOW SHOP* HÍBRIDO MULTIOBJETIVO

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, como parte dos requisitos necessários para a obtenção do título de Doutor em Modelagem Matemática e Computacional.

Aluno: Eduardo Camargo de Siqueira
Orientador: Prof. Dr. Marcone Jamilson Freitas Souza (UFOP, Brasil)
Co-Orientador: Prof. Dr. Sérgio Ricardo de Souza (CEFET-MG, Brasil)

Belo Horizonte, março de 2019.

Siqueira, Eduardo Camargo.
S618h Heurísticas computacionais para resolução de um problema flow shop híbrido multiobjetivo / Eduardo Camargo Siqueira. – 2019.
xviii, 150 f.

Tese de doutorado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional.
Orientador: Marcone Jamilson Freitas Souza.
Coorientador: Sérgio Ricardo de Souza.
Tese (doutorado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Programação de produção – Teses. 2. Algorítmos heurísticos – Teses. 3. Otimização matemática – Teses. I. Souza, Marcone Jamilson Freitas. II. Souza, Sérgio Ricardo de. III. Centro Federal de Educação Tecnológica de Minas Gerais. IV. Título.

CDD 519.6

Elaboração da ficha catalográfica pela Biblioteca-Campus II / CEFET-MG

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM MATEMÁTICA E COMPUTACIONAL

HEURÍSTICAS COMPUTACIONAIS APLICADAS A UM PROBLEMA FLOWSHOP HÍBRIDO MULTIOBJETIVO.

Tese de Doutorado apresentada por **Eduardo Camargo de Siqueira**, em 21 de março de 2019, ao Programa de Pós-Graduação em Modelagem Matemática e Computacional do CEFET-MG, e aprovada pela banca examinadora constituída pelos professores:

Prof. Dr. Marcone Jamilson Freitas Souza
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Sérgio Ricardo de Souza
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. José Elias Cláudio Arroyo
Universidade Federal de Viçosa

Prof. Dr. Luciano Perdigão Cota
Instituto Tecnológico Vale

Prof. Dr. Moacir Felizardo de França Filho
Centro Federal de Educação Tecnológica de Minas Gerais

Profª. Drª. Elisângela Martins de Sá
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Flávio Vinícius Cruzeiro Martins
Centro Federal de Educação Tecnológica de Minas Gerais

Visto e permitida à impressão,

Prof. Dr. Thiago de Souza Rodrigues
Coordenador do Programa de Pós-Graduação Stricto Sensu em Modelagem Matemática e Computacional

Dedico à minha esposa Ana Carolina e à minha filha Isabela Hadassa.

Agradecimentos

A **DEUS** pela sua imensa misericórdia, amor e proteção, e por tudo que Ele é, pois sem Ele nada seria possível.

À minha esposa, **Ana Carolina**, e à minha filha, **Isabela Hadassa**, pelo amor, pelo carinho, pelo apoio incondicional e por suas palavras de encorajamento.

Aos meus pais, **Nelson Natal** e **Rosa Maria**, e junto com eles todos meus familiares, pelo apoio, amor, dedicação e confiança que depositaram em meus projetos de vida.

Aos meus orientadores, **Dr. Marcone Jamilson** e **Dr. Sérgio Ricardo**, pelas orientações e motivações em relação a este trabalho e a minha carreira acadêmica.

À família do **Rev. Manassés Villaça** e sua esposa **Izolismaria**, por todo apoio e pelos seus conselhos, também por sempre nos hospedar na querida cidade de Belo Horizonte.

À família do **Bruno Baptista**, sua esposa **Nathália**, e a pequena **Manu** pela amizade e carinho, também foi nosso porto seguro em Belo Horizonte.

Ao meu grande amigo, **José Maurício**, que se tornou mais que um irmão para mim.

Aos meus amigos, **Roitier Campos** e **Chris Manuel** que sempre me apoiaram e encorajaram nessa jornada.

Aos meus amigos **Rodney Diana**, **Maria Amélia** e **Bruno Rosa**, que mesmo distantes sempre ajudaram em todos os momentos.

Ao meu querido diretor **Ronaldo Diláscio**, aos Coordenadores **Gustavo Alexandre**, **Gustavo Neves** e **Edwar Saliba** pelas palavras de apoio e confiança e pela compreensão em momentos chave desse processo.

Aos queridos irmãos da **Igreja Presbiteriana Memorial de Belo Horizonte**, da **Terceira Igreja Presbiteriana de Paracatu** e da **Igreja Presbiteriana Ebenézer de Anápolis**, pelas orações, compreensão e pelo apoio espiritual durante toda essa jornada.

Ao **CEFET-MG** e ao **IFTM**, pela confiança que foi depositada em meu trabalho.

E a todos que de forma direta ou indireta contribuíram para realização deste trabalho.

"*Às vezes, são as pessoas de quem menos esperamos que fazem coisas que ninguém poderia esperar.*"

Allan Turing

Resumo

Esta Tese trata o problema de sequenciamento de tarefas em ambiente *Flow Shop* Híbrido Multiobjetivo. Nesse problema tem-se um conjunto de tarefas que devem ser executadas em um conjunto de estágios. Em cada estágio tem-se um conjunto de máquinas paralelas não-relacionadas. Algumas tarefas não passam por todos os estágios. Além disso, também são consideradas características como elegibilidade de máquinas, datas de entrega e custos por atrasos e antecipações. Inicialmente, os critérios de avaliação foram as minimizações de *makespan*, soma ponderada dos atrasos e soma ponderada das antecipações. Isso caracteriza o problema abordado como multiobjetivo, por envolver três objetivos conflitantes, isto é, a minimização de um dos objetivos influencia negativamente na minimização dos outros dois. Assim, não é possível encontrar uma única solução que otimize todos os objetivos ao mesmo tempo. Posteriormente, foram adicionados ao problema dois novos critérios, que são as minimizações do tempo de ociosidade e do número de tarefas atrasadas. A resolução de problemas dessa classe é dificultada principalmente pela deterioração da seleção por dominância de Pareto e pelo aumento exponencial do número de soluções necessárias para aproximar da frente de Pareto. Para resolver os problemas, foram propostos dois algoritmos, o primeiro baseado na metaheurística *Multi-Objective General Variable Neighborhood Search* (MO-GVNS) e o segundo baseado na metaheurística *Pareto Iterated Local Search* (P-ILS). Os algoritmos desenvolvidos foram testados em instâncias adaptadas da literatura e seus resultados foram comparados com os de outros algoritmos da literatura. Os resultados foram avaliados em relação às métricas *Hypervolume*, *Epsilon*, *Spacing* e *Hierarquical Cluster Counting* (HCC), e validados estatisticamente pelo teste de Levene e pelos gráficos de intervalo de confiança. Os resultados obtidos e as análises realizadas mostraram a superioridade dos algoritmos propostos para a solução dos problemas apresentados com relação às métricas *Hypervolume*, *Epsilon* e HCC.

PALAVRAS-CHAVE: Sequenciamento de tarefas. *Flow Shop* Híbrido. Otimização Multiobjetivo. Busca em Vizinhança Variável Multiobjetivo.

Abstract

This thesis addresses the multiobjective hybrid flow shop (MOHFS) scheduling problem. In this problem there are a set of jobs that must be performed in a set of stages. For each stage there is a set of unrelated parallel machines. Some jobs do not go through all stages. In addition, machine eligibility, due dates and tardiness and earliness costs are also some characteristics considered. Initially, the evaluation criteria were the minimizations of makespan, weighted sum of the tardiness and weighted sum of the earliness. This characterizes the problem addressed as multiobjective because it involves three conflicting objectives, that is, the minimization of one of the objectives influences negatively the minimization of the other two. Thus, it is not possible to find a single solution that optimizes all objectives at the same time. Subsequently, two new objectives were added to the problem, the new criteria used were the minimization of idleness and the number of delayed jobs. The resolution of problems of this class is hampered mainly by the deterioration of the selection by Pareto dominance and the exponential increase in the number of solutions needed to approximate of the Pareto front. In order to solve it, two algorithms were proposed, the first one based on the Multi-Objective General Neighborhood Variable Search (MO-GVNS) metaheuristic and the second one based on the *Pareto Iterated Local Search* (P-ILS) metaheuristic. These algorithms were tested in instances adapted from the literature and their results were compared with those of other algorithms in the literature. The results were evaluated in relation to the Hypervolume, Epsilon, Spacing and Hierarchical Cluster Counting (HCC) metrics, and validated through the Levene test and the confidence interval graphics. The obtained results and the realized analyzes showed the superiority of the proposed algorithms for solving the treated problems with respect to the Hypervolume, Epsilon and HCC metrics.

KEYWORDS: *Scheduling. Hybrid Flow Shop. Multi-objective optimization. Multi-objective Variable Neighborhood Search.*

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Este capítulo traz uma introdução ao tema proposto para esta tese. Na Seção 1.1 é feita uma breve apresentação sobre problemas de sequenciamento de tarefas. As Seções 1.2, 1.3 e 1.4 trazem, respectivamente, a metodologia adotada, as justificativas e as contribuições deste trabalho. Por fim, a Seção 1.5 mostra como o restante deste trabalho está estruturado.

1.1 Apresentação

Problemas de sequenciamento de tarefas são comuns principalmente na produção industrial e consistem em definir uma sequência de tarefas a serem executadas em um conjunto de máquinas, otimizando um ou mais objetivos. Neste trabalho é considerado um problema de sequenciamento de tarefas multiobjetivo em ambientes *Flow Shop* Híbrido, com muitas características de problemas do mundo real. Algumas dessas características abordadas no problema apresentado são a existência de máquinas paralelas não-relacionadas em cada estágio, datas de entrega (*due dates*), custos para atraso e antecipação, elegibilidade de máquinas e salto de estágios. Naderi et al. (2010) chamam a atenção para duas questões importantes sobre problemas *Flow Shop* Híbrido: a determinação da sequência em cada estágio e a distribuição de tarefas nas máquinas em cada estágio.

Nesse problema, conhecido como MOHFS (das iniciais em inglês *Multi-Objective Hybrid Flow Shop*), inicialmente foram considerados os objetivos de minimização do *makespan*, da soma ponderada dos atrasos e da soma ponderada das antecipações. Posteriormente, foram adicionados outros dois objetivos: a minimização do tempo de ociosidade e a minimização do número de tarefas atrasadas. Essas duas variações do problema com relação ao número de objetivos estão detalhados no Capítulo 4.

Os problemas de otimização multiobjetivo são caracterizados por envolver dois ou

mais objetivos conflitantes simultaneamente. Assim, não é possível encontrar uma única solução que otimize todos os objetivos ao mesmo tempo. Em problemas multiobjetivo, o conjunto de soluções é parcialmente ordenado, isto é, dado dois elementos, nem sempre é possível compará-los. As soluções de um problema multiobjetivo formam frentes de dominância, as quais possuem entre si relações de ordem. A frente de dominância mais eficiente é chamada de conjunto de soluções não-dominadas. Para esta classe de problemas, o desafio é buscar o conjunto ótimo de soluções não-dominadas, as chamadas fronteiras de Pareto (Ehrgott, 2005).

Ishibuchi et al. (2008) apresentam uma revisão de algoritmos evolucionários multiobjetivo, destacando a dificuldade de resolver problemas com muitos objetivos usando as técnicas existentes até aquele momento. Para resolver essas dificuldades, Deb e Jain (2014) propõem um algoritmo genético chamado NSGA-III (do inglês *Nondominated Sorting Genetic Algorithm III*) que representam uma melhora do algoritmo NSGA-II, apresentado em Deb et al. (2002), para o tratamento de problemas com três ou mais objetivos. O NSGA-II, por sua vez, é uma adaptação de sua primeira versão, denominada NSGA, proposta em Srinivas e Deb (1995). O processo de seleção desses algoritmos é baseado na classificação de dominância e na distância de aglomeração. No NSGA-II a distância é calculada com base em todos os pontos da população de indivíduos; já no NSGA-III o cálculo é realizado em relação a alguns pontos de referência. Yuan et al. (2014) melhoraram o NSGA-III incluindo normalização adaptativa e preservação da diversidade dos pontos de referência. Mais detalhes sobre problemas de otimização multiobjetivo podem ser encontrados no Capítulo 2.

1.2 Metodologia

Esta pesquisa foi realizada por meio de um estudo descritivo e experimental, teórico e prático, sobre algoritmos heurísticos e problemas de sequenciamento de tarefas em ambiente *Flow Shop* híbrido, baseado em consultas bibliográficas e experimentos computacionais. Foi proposta a aplicação de algoritmos heurísticos adaptados para a resolução do problema de sequenciamento de tarefas em ambiente *Flow Shop* híbrido. Foram considerados, inicialmente, três objetivos conflitantes, e, em seguida, cinco objetivos conflitantes, que se enquadram na classe de problemas de otimização combinatória multiobjetivo. O foco da pesquisa é a adaptação e melhoria de métodos heurísticos aplicados ao problema em questão.

Baseado nisso, se fez necessário, a *priori*, a realização de uma pesquisa bibliográfica sobre problemas de sequenciamento de tarefas multiobjetivo, além de uma investigação sobre possíveis métodos de resolução destes.

O prosseguimento do trabalho se deu com a implementação de dois métodos

heurísticos aplicados ao problema tratado, apresentados no Capítulo 5. Os resultados alcançados pela execução exaustiva dos algoritmos implementados são apresentados e analisados. Foram utilizadas diferentes métricas de avaliação de desempenho, que avaliam tanto o espalhamento das soluções geradas pelo algoritmo como também a convergência destes algoritmos. Além disso, os resultados passaram por uma análise utilizando metodologias estatísticas, que mostram se há diferenças estatísticas relevantes entre os resultados. Foi utilizado também um pacote da literatura para calibração de parâmetros dos algoritmos desenvolvidos.

1.3 Justificativas

Problemas de otimização envolvem a seleção de valores para um número de variáveis, concentrando a atenção em um objetivo destinado a quantificar desempenho, sujeito às restrições que podem limitar a seleção de valores das variáveis, e esse objetivo deve ser maximizado ou minimizado (Luenberger e Ye, 1984). Em um problema combinatório, o conjunto de soluções possíveis é finito. Apesar disso, um problema dessa natureza não é necessariamente fácil de ser solucionado. Segundo Goldbarg e Luna (2000), um problema de otimização combinatória envolve variáveis que não podem assumir valores contínuos, ficando restritas a assumir valores discretos. Esse requisito implica, em geral, uma maior complexidade computacional. A enumeração completa das possíveis soluções pode se tornar inviável à medida que a cardinalidade do conjunto de soluções aumenta.

Existe uma variedade de problemas reais que possuem a característica de otimização combinatória. Segundo Arroyo (2002), otimização combinatória é um campo de pesquisa que pode ser encontrado em diversas áreas do conhecimento, como planejamento da produção, corte e empacotamento, roteamento de veículos, dentre outras.

O problema tratado é da classe de problemas combinatórios multiobjetivo. De acordo com Ticona (2003), geralmente os problemas de otimização encontrados no mundo real possuem múltiplos objetivos, com os aqui tratados. Essa situação ainda é pouco encontrada na literatura para problemas *Flow Shop*. A semelhança do problema abordado com o mundo real está nas características do ambiente de máquinas. Isso mostra a relevância do tema levantado.

1.4 Contribuições

Este trabalho propõe a aplicação de métodos heurísticos para o tratamento de um problema *Flow Shop* Híbrido com muitos objetivos. Uma contribuição importante é que esse problema considera características comuns em problemas do mundo real.

Por exemplo, existe um conjunto de máquinas disponíveis em cada estágio, e essas são independentes entre si. Além disso, alguns produtos específicos podem pular etapas do processo, o que é muito comum em vários processos industriais. Da mesma forma, os produtos têm prazos de entrega desejáveis, e podem haver custos relacionados caso eles não sejam atendidos.

A principal contribuição deste trabalho é a otimização de até cinco objetivos no problema *Flow Shop* Híbrido. Inicialmente foram considerados simultaneamente três objetivos conflitantes: a minimização do *makespan* e das somas ponderadas dos atrasos e das antecipações. Uma solução ótima para o objetivo *makespan* é muito eficiente do ponto de vista da produção, mas pode ser ruim para o cliente que deseja que o serviço seja entregue sem atrasos. Uma solução ideal para o critério da soma ponderada dos atrasos pode satisfazer aos desejos dos clientes, mas pode ser altamente ineficiente por diminuir a capacidade de produção e aumentar os custos de estoque. A melhor solução para o objetivo de minimização da soma ponderada das antecipações, por sua vez, pode reduzir os custos de estoque, porém pode aumentar o *makespan* e/ou o atraso. Posteriormente, foram introduzidos mais dois outros objetivos, quais sejam: a minimização do tempo de ociosidade e do número de tarefas atrasadas, que também apresentam conflitos com os demais objetivos.

1.5 Estrutura do trabalho

O restante deste trabalho está dividido como segue. O Capítulo 2 traz uma abordagem teórica sobre problemas de otimização combinatória multiobjetivo. O Capítulo 3 apresenta a notação utilizada e os principais problemas de sequenciamento de tarefas. Já o Capítulo 4 apresenta os detalhes e características do problema abordado nesta Tese, e o Capítulo 5 mostra os algoritmos propostos para resolvê-lo. O Capítulo 6 apresenta os resultados encontrados pela execução dos algoritmos. Por fim, o Capítulo 7 traz a conclusão deste trabalho. Os Anexos I e II mostram os gráficos referentes a análise dos resultados obtidos.

Capítulo 2

Fundamentos de Otimização Combinatória Multiobjetivo

Este Capítulo apresenta fundamentos de Otimização Combinatória Multiobjetivo, os quais são caracterizados por possuírem dois ou mais objetivos. Em diversos trabalhos encontrados na literatura de língua inglesa, problemas com dois ou três objetivos são chamados de *multi-objectives*, enquanto problemas com um número maior de objetivos são referenciados como *many-objectives*. Esta tese trata de um problema com cinco objetivos, porém como não há uma diferenciação em língua portuguesa será referenciado como multiobjetivo. Na Seção 2.1 é feita uma apresentação dos fundamentos teóricos. A Seção 2.2 mostra uma revisão bibliográfica de métodos de resolução de problemas de otimização multiobjetivo. A Seção 2.3 discute os métodos de resolução em problemas de otimização multiobjetivo, com especial ênfase aos diretamente adotados nesta Tese. A Seção 2.4 apresenta as métricas de desempenho associadas à análise de problemas de otimização multiobjetivo.

2.1 Fundamentos Teóricos

Segundo Ehrgott e Gandibleux (2008), a otimização de multiobjetivo se difere da otimização mono-objetivo principalmente pelas seguintes características. Primeiramente, na otimização multiobjetivo, o significado tradicional de solução ótima não faz sentido porque uma única solução que otimiza todos os objetivos simultaneamente geralmente não existe. Em segundo lugar, é necessário durante a tomada de decisão levar em consideração quais são as preferências ao avaliar um conjunto de soluções. Por último, os diferente objetivos encontrados em problemas da vida real podem ser expressos por funções matemáticas de uma variedade de formas e estruturas.

No mesmo sentido, para Coello et al. (2010) em um problema combinatório multi-

objetivo muitas vezes é necessário expressar as preferências de um tomador de decisão a fim de obter uma única solução. Assim, antes de começar a resolver um problema de otimização combinatória multiobjetivo, deve-se decidir como as preferências podem ser consideradas. Os autores apresentam três formas de incorporá-las:

(i) abordagem *a priori*: o tomador de decisão dá indicações sobre a importância dos diferentes objetivos antes da otimização;

(ii) abordagem progressiva: o tomador de decisão participa durante o processo de otimização, expressando, assim, suas preferências à medida que elas são utilizadas;

(iii) abordagem *a posteriori*: é apresentado ao tomador de decisão um conjunto de soluções geradas por um método de otimização e, então, ele pode escolher dentro do conjunto a solução mais adequada aos seus interesses.

A fundamentação apresentada nesta Seção está baseada na abordagem *a posteriori*. Assim, um problema de Otimização Combinatória Multiobjetivo (ou vetorial) pode ser descrito da seguinte forma:

$$\begin{aligned} \min \quad & z(x) = Cx && (2.1)\\ \text{sujeito a} \quad & Ax = b && (2.2)\\ & x \in \{0,1\}^{\mu} && (2.3) \end{aligned}$$

em que

- $C \in \mathbb{Z}^{\rho \times \mu}$ é uma matriz de coeficientes de ρ funções objetivo $z_k(x) = c^k x$, $k = 1, \ldots, \rho$;
- $x = (x_1, x_2, \ldots, x_\mu) \in \{0,1\}^{\mu}$ é o vetor de variáveis de decisão de μ variáveis binárias;
- $A \in \mathbb{Z}^{\eta \times \mu}$ é uma matriz de coeficientes das restrições; e
- $b \in \mathbb{Z}^{\mu}$ é um vetor de termos independentes das restrições.

O conjunto $\mathcal{X} = \{x \in \{0,1\}^{\mu} \ : \ Ax = b\}$ é chamado de região factível no espaço de variáveis de decisão (ou espaço de decisão) e o conjunto $\mathcal{Z} = z(\mathcal{X}) = \{Cx \ : \ x \in \mathcal{X}\}$ é chamado de região factível no espaço de objetivos (ou espaço dos objetivos). A função objetivo $z(\cdot)$, portanto, mapeia o espaço de variáveis de decisão no espaço de objetivos. Por simplicidade de notação, define que os pontos $z = z(x) \in \mathcal{Z}$.

Em um problema com uma única função objetivo, avaliar a eficiência de uma solução é trivial, pois existe uma relação de ordem, isto é, dados dois elementos quaisquer z^1 e z^2, é sempre verdade que $z^1 \leq z^2$ ou $z^2 \leq z^1$, ainda se $z^1 \leq z^2$ e $z^2 \leq z^3$, então $z^1 \leq z^3$. Por exemplo, em problemas de minimização, o valor da função objetivo $z^1 = 3$ é mais eficiente que $z^2 = 7$, e $z^2 = 7$ é mais eficiente que $z^3 = 10$, e portanto, z^1 é mais eficiente que z^3.

Em problemas de otimização multiobjetivo, por outro lado, a relação de ordem não existe, isto é, dados dois vetores z^1 e z^2, nem sempre $z^1 \leq z^2$ ou $z^2 \leq z^1$. O conjunto de soluções de um problema de otimização com dois ou mais objetivos é chamado de parcialmente ordenado. Assim, a eficiência é avaliada de acordo com relações de dominância. Para a comparação de duas soluções x_1 e x_2, por exemplo, a dominância de Pareto (Ehrgott, 2005) é frequentemente usada (Ehrgott, 2005).

Definição 2.1.1 (Coello et al. (2010))
Dominância de Pareto*: Uma solução $y = (y_1, y_2, \ldots, y_\mu) \in \mathcal{X}$ domina, no sentido de Pareto, uma solução $x = (x_1, x_2, \ldots, x_\mu) \in \mathcal{X}$, em um contexto de minimização, se e somente se $\forall\ i \in [1 \ldots \rho]$, $z_i(y) \leq z_i(x)$ and $\exists\ i \in [1...\rho]$ tal que $z_i(y) < z_i(x)$. Neste caso, diz-se que y domina x e denota-se $y \prec x$.*

Definição 2.1.2 (Coello et al. (2010))
Uma solução $x^* \in \mathcal{X}$ é denominada Pareto Ótima se não existe outra solução $x \in \mathcal{X}$ que a domina.

Soluções Pareto Ótimas também são nomeadas como soluções eficientes. Assim, conforme a Definição 2.1.2, x^* é uma solução eficiente. Neste caso, $z(x^*)$ é denominado ponto não-dominado (Ehrgott, 2005).

Definição 2.1.3 (Coello et al. (2010))
Conjunto Pareto Ótimo*: conjunto formado pela união de todas as soluções Pareto Ótimas x^*, dado, portanto, por:*

$$\mathcal{X}^* = \{x \in \mathcal{X} \ : \ \nexists y \in \mathcal{X} \ : \ y \prec x\}$$

Definição 2.1.4 (Coello et al. (2010))
Conjunto dos Pontos Não-dominados*: conjunto formado pela união de todos os pontos*

não-dominados z^*, dado, portanto, por:

$$\mathcal{Z}^* = \{z^* \in \mathcal{Z} \ : \ z^* = z(x^*), \quad x^* \in \mathcal{X}^*\}$$

Ou seja, o conjunto $\mathcal{X}^*$ é formado pelas soluções $x^* \in \mathcal{X}$ tais que $\nexists x \in \mathcal{X}$ de modo que, $\forall x^* \in \mathcal{X}^*$ e $x^* \neq x$, tem-se que $z^* = z(x^*) \prec z(x)$. O Conjunto Pareto Ótimo é também chamado Conjunto das soluções eficientes.

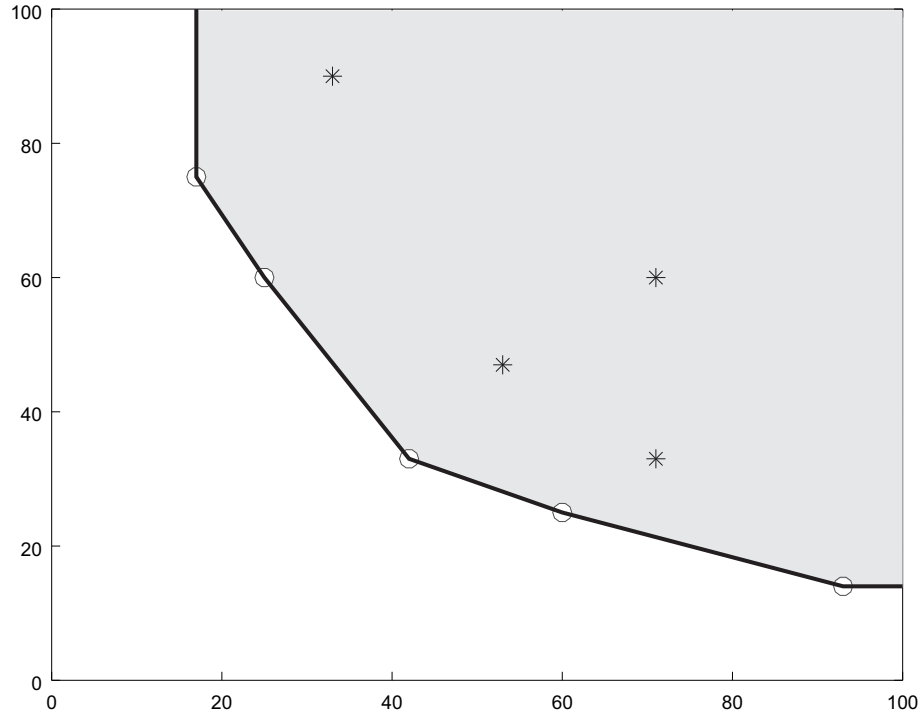


Figura 2.1: Fronteira de Pareto.

Definição 2.1.5 (Ehrgott (2005))
Fronteira Pareto Ótima*: O conjunto Z^* de vetores objetivo correspondente ao conjunto Pareto Ótimo é chamado de Frente ou Fronteira Pareto Ótima.*

Ou seja, a Fronteira Pareto Ótima corresponde à imagem dos pontos do Conjunto Pareto Ótimo, na forma

$$\mathcal{Z}^* = \left\{z \ : \ z = c^k x^*, \quad x^* \in \mathcal{X}^*\right\}$$

As soluções podem ser incomparáveis entre si. Nesse caso, essas soluções fazem parte de uma mesma Fronteira de Dominância.

Definição 2.1.6 (Ehrgott (2005))
Uma Fronteira de Dominância $\mathcal{X}'$ é formada por: dadas duas soluções quaisquer $x \in \mathcal{X}'$ e $y \in \mathcal{X}'$, então não é verdade que $z(y) \prec z(x) \vee z(x) \prec z(x)$.

As diversas fronteiras de dominância existentes possuem entre si relações de ordem, ou seja, pode-se inferir qual fronteira é mais eficiente que a outra. Assim, em problemas de otimização multiobjetivo, a busca não é pela solução ótima, mas, sim, pelo conjunto de soluções eficientes, o qual é chamado de soluções não-dominadas. A Figura 2.1 mostra um exemplo de Fronteira de Pareto. Note que o conjunto Pareto Ótimo é formado por pontos não-dominados por qualquer outro ponto. Este pontos formam um limite inferior esquerdo da região de z, que é a chamada Fronteira Pareto Ótima, conforme mostra a Figura 2.1. Assim, a eficiência de um problema multiobjetivo é dado pela otimalidade de Pareto, pois, conforme a Definição 2.1.3, um Conjunto de Pareto é considerado ótimo para um problema específico se não houver nenhum outro conjunto de soluções que o domine.

Uma vez que há dificuldades de comparação entre as soluções, é interessante definir pontos de referência para a avaliação dos resultados. Estes pontos, segundo Ehrgott (2005), representam indicações das faixas de valores que as soluções não-dominantes podem alcançar no espaço dos objetivos $\mathcal{Z}$.

Definição 2.1.7 (Ehrgott (2005))
Ponto Ideal*: o ponto*

$$z^I = \begin{bmatrix} z_1^I & z_2^I \dots & z_k^I \dots z_\rho^I \end{bmatrix}$$

em que:

$$z_k^I = \min_{x \in \mathcal{X}} z_k(x)$$

é denominado ponto ideal.

Portanto, o ponto ideal corresponde ao ponto obtido a partir da minimização de cada objetivo $z_k(\cdot)$ separadamente. Conforme apontam Ehrgott (2005) e Coello et al. (2010), o ponto ideal não é um ponto factível, pois, caso o fosse, indicaria ausência de conflito entre os objetivos e, portanto, não seria necessário usar o conceito de Otimalidade de Pareto. Por esse motivo, pode não ser uma boa opção minimizar um dos objetivos isoladamente. Mas, claramente, o ponto ideal representa um limitante inferior para o conjunto dos pontos não-dominados.

Definição 2.1.8 (Ehrgott (2005))
Ponto Utópico*: O ponto*

$$z^U = z^I - \epsilon 1$$

é denominado ponto utópico, sendo $\epsilon > 0$ *e* $1 = [1 \dots 1] \in \Re^\rho$ *o vetor unitário.*

Obviamente, $z^U < z^I$.

Uma outra definição importante é o ponto Nadir.

Definição 2.1.9 (Ehrgott (2005))
Ponto Nadir*:O ponto*

$$z^N = \left[z_1^N \quad z_2^N \dots z_k^N \dots z_\rho^N\right]$$

definido como:

$$\begin{aligned} z_k^N &= \max z_k(x) \\ & \quad x \in \mathcal{X}^* \end{aligned}$$

é denominado ponto Nadir.

O ponto Nadir é o vetor que contém os piores valores para cada objetivo ao otimizar os outros objetivos e é um limitante superior para o conjunto dos pontos não-dominados. Claramente, $z^U < z^I < z_k$ e $z_k < z^N$, $\forall z \in \mathcal{Z}^*$.

Por exemplo, seja um problema com dois objetivos de minimização ($\rho = 2$), e o conjunto de pontos $Z = \{(53, 47), (71, 33), (71, 60), (33, 90), (93, 14), (60, 25), (42, 33), (25, 60), (17, 75)\}$, que está plotado na Figura 2.2, na qual os eixos x e y representam os dois objetivos. Observe que z^1 domina z^3, isto é, $(53, 47)$ domina $(71, 60)$, pois ${z^1}_1 = 53 \leq {z^3}_1 = 71$, ${z^1}_2 = 47 \leq {z^3}_2 = 60$ e $z^1 \neq z^3$. Pode-se notar, nesse exemplo, que não há relação de dominância entre os pontos $(93, 14), (60, 25), (42, 33), (25, 60), (17, 75)$, e também, entre os pontos $(71, 33), (42, 33), (33, 90)$. Esses dois conjuntos citados formam duas frentes de dominância, sendo a primeira mais eficiente que a segunda.

Nesse exemplo, pode-se notar que a solução que minimiza o primeiro objetivo e maximiza o outro e vice-versa, são os seguintes pontos em z: $(93, 14)$ e $(17, 75)$, respectivamente. O ponto que combina a minimização de todos os objetivos, ou seja, a solução utópica do exemplo citado é $(17, 14)$. O ponto de nadir, nesse caso, é $(93, 75)$.

A Figura 2.3, adaptada de Coello et al. (2010) mostra uma representação das soluções factíveis, das soluções Pareto, da solução utópica e do ponto de nadir.

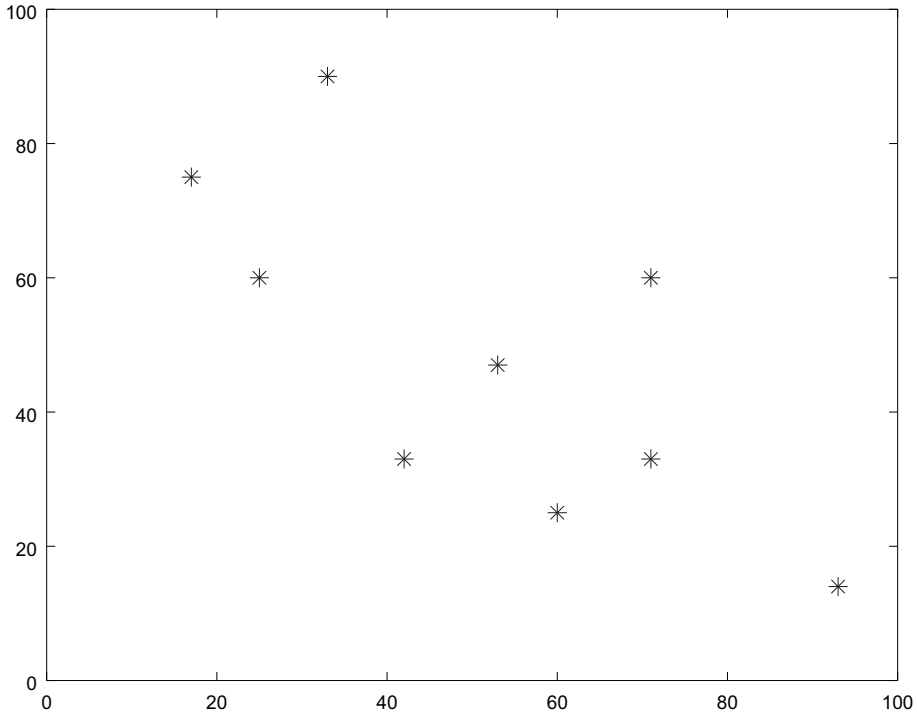


Figura 2.2: Soluções no espaço de objetivos.

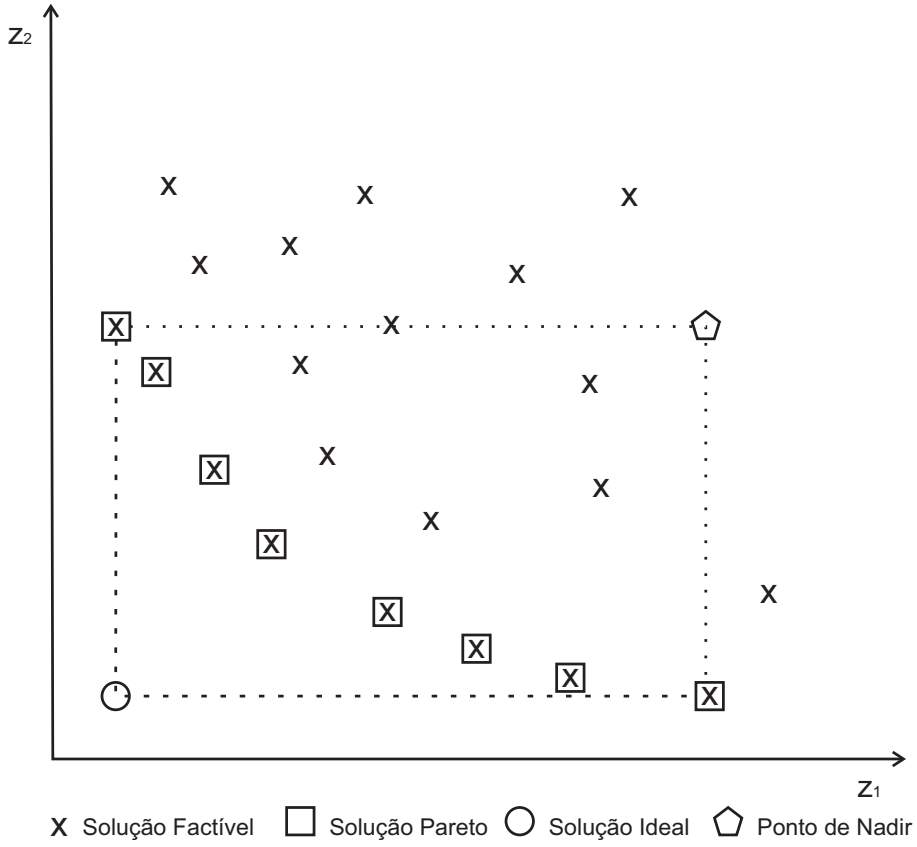


Figura 2.3: Solução Utópica e Ponto de Nadir. Fonte: Coello et al. (2010)

2.2 Uma Revisão de literatura de Métodos de Otimização Multiobjetivo

Li et al. (2015) apresentaram uma revisão da literatura a respeito de problemas de otimização com mais de um objetivo. Inicialmente, nesta Seção será apresentado alguns trabalhos que abordaram problemas multiobjetivo com aplicação de métodos evolutivos. Posteriormente, serão apresentados trabalhos que utilizarão técnicas de busca local.

Fonseca et al. (1993) foram os primeiros autores a propor um Algoritmo Genético Multiobjetivo, o MOGA, do inglês *Multi-Objective Genetic Algorithm*. Horn et al. (1994) propuseram o NPGA, do inglês *Niched Pareto Genetic Algorithm*. Srinivas e Deb (1995) propuseram o NSGA, do inglês *Nondominated Sorting Genetic Algorithm*. Neste último algoritmo a população é classificada em camadas de não-dominância. Ishibuchi e Murata (1996) apresentaram o MOGLS (do inglês *Multi-Objective Genetic Local Search*). Depois disso, outras variações desse algoritmo foram propostas. Foi o caso do C-MOGLS (do inglês *Celular* MOGLS), proposta por Murata et al. (2000) e do MOGLS modificado, proposto por Jaszkiewicz (2002). Esses métodos propostos são chamados de algoritmos de primeira geração.

A segunda geração de algoritmos multiobjetivos começa com a proposta do *Strength Pareto Evolutionary Algorithm* – SPEA (Zitzler e Thiele, 1998) e, depois, pelo seu sucessor, o SPEA2 (Zitzler et al., 2002). O SPEA foi o primeiro algoritmo multiobjetivo a utilizar arquivo; os indivíduos não-dominados são copiados para o arquivo e recebem um valor de força. Knowles e Corne (2000) propõem uma estratégia evolutiva, outro algoritmo de segunda geração, chamada em inglês de *Pareto Archived Evolution Strategy* (PAES). Deb et al. (2002) propuseram o NSGA-II, adicionando ao NSGA o operador de distância de aglomeração.

Tanto os algoritmos de primeira quanto os de segunda geração são métodos que encontram dificuldades à medida que o número de objetivos cresce. Em geral, eles são bons para problemas biobjetivos e não produzem bons resultados em problemas nos quais o número de objetivos é maior que três.

Ishibuchi et al. (2008) apontam três razões pelas quais os algoritmos evolutivos de primeira e segunda geração não se comportam bem em problemas com mais de três objetivos. O primeiro motivo é a deterioração da seleção por dominância baseada em Pareto, isto é, quando o número de objetivos aumenta, quase todas as soluções de uma população tornam-se não-dominadas e isso enfraquece a seleção baseada na dominância de Pareto. Outra razão é o aumento exponencial do número de soluções necessárias para aproximar a frente de Pareto ótima. O terceiro motivo é a dificuldade de visualização das soluções em um espaço de objetivos com mais de duas dimensões.

Para resolver algumas dessas dificuldades, em Dugardin et al. (2010) é apresentado um algoritmo, chamado L-NSGA, no qual o relacionamento de Dominância de Pareto é substituído pelo relacionamento de Dominância de Lorenz. No mesmo sentido, Deb e Jain (2014) propuseram um algoritmo genético chamado NSGA-III, que representa uma melhora do NSGA-II para o tratamento de problemas com mais de três objetivos. No NSGA-III o cálculo da distância de aglomeração é realizado em relação a alguns pontos de referência. Yuan et al. (2014) melhoraram o NSGA-III incluindo normalização adaptativa e preservação da diversidade dos pontos de referência.

Algoritmos de busca local também têm sido propostos para resolução de problemas multiobjetivo. Como exemplo, citamos o MOSA (do inglês *Multi-Objective Simulated Annealing*) proposto por Serafini (1994), *Multi-Objective Tabu Search* (MOTS) proposto por Hansen (1997), o *Pareto Simulated Annealing* – PSA (Czyzak, 1998), o GRASP-MULTI proposto por Vianna e Arroyo (2004), o *Pareto Local Search* (PLS) proposto por Paquete et al. (2004), o P-ILS (do inglês *Pareto Iterated Local Search*) proposto por Geiger (2007), o MO-VNS proposto por Geiger (2008) e o *Restarted Iterated Pareto Greedy* (RIPG) apresentado em Minella et al. (2011); Ciavotta et al. (2013).

Algoritmos baseados em busca de vizinhança variável, isto é, na metaheurística MO-VNS, foram aplicados com sucesso para resolver problemas combinatórios multiobjetivos, desde sua primeira versão proposta por Geiger (2004). Schilde et al. (2009) estenderam a estrutura do método VNS para o caso multiobjetivo, a fim de planejar rotas turísticas em uma cidade. Liang et al. (2009) apresentaram um algoritmo MO-VNS para resolver um problema de máquina paralela idênticas. Arroyo et al. (2011) compararam três algoritmos multiobjetivos baseados no MO-VNS aplicado ao problema de sequenciamento em máquina única com tempos de *setup* dependentes da sequência. Liang e Chuang (2013) abordaram um problema de alocação de recursos com dois objetivos aplicando um algoritmo MO-VNS. Rego et al. (2014) aplicaram o MO-VNS para resolver um problema de sequenciamento em máquina única bi-objetivo. Gomes et al. (2014) aplicaram o MO-VNS para resolver um problema de agendamento de projetos com recursos restritos bi-objetivo com relações de precedência.

Duarte et al. (2015) apresentaram uma adaptação de três variantes da metaheurística VNS, as quais foram utilizadas para resolver dois problemas de otimização combinatória multiobjetivos. Os resultados mostraram que a variante MO-GVNS foi mais eficiente para resolver os problemas apresentados. Esta variante usa o MO-VND (do inglês *Multi-objective Variable Neighborhood Descent*) como busca local. Segundo os autores, ao final do procedimento, o MO-VND garante que cada ponto da solução foi melhorado em relação a cada função objetivo e a cada vizinhança do problema

tratado.

Em Masri et al. (2019), um método híbrido que usa MO-VNS para explorar e intensificar a busca em regiões específicas é aplicado para resolver um problema de fluxo de comunicação. López-Sánchez et al. (2018) propuseram quatro variantes de um algoritmo híbrido que combina GRASP com VND em versões multiobjetivo. O VND é utilizado para melhorar a solução gerada pela fase de construção do GRASP. Essas variantes são usadas para resolver um problema real de coleta de lixo.

2.3 Métodos de Resolução

Esta seção discute sobre os métodos de resolução de problemas de otimização combinatória multiobjetivo. Na Subseção 2.3.1 é apresentado a classificação dos métodos de resolução. Na Subseção 2.3.2 são apresentados os algoritmos genéticos NSGA-II e NSGA-III, na Subseção 2.3.3 é feita uma apresentação de métodos de busca em vizinhança variável e na Subseção 2.3.4 é abordado sobre o método *Pareto Iterated Local Search*. Por fim, na Subseção 2.4 são explicadas as medidas de desempenho em problemas de otimização multiobjetivo.

2.3.1 Classificação de Algoritmos

O principal desafio em otimização combinatória multiobjetivo é a busca pelo conjunto de soluções mais eficientes de um problema. Para problemas mais complexos, essa busca pode ter um custo computacional muito alto, até mesmo para instâncias de pequeno porte em muitos casos, em especial se envolver mais de três objetivos a serem satisfeitos conjuntamente. Em virtude disso, métodos heurísticos são normalmente utilizados para resolver essa classe de problemas (Ehrgott, 2005). É importante ressaltar, no entanto, para efeitos de completude desse tese, que métodos exatos foram e continuam sendo extensamente desenvolvidos e utilizados para a solução de problemas dessa classe, como mostram, por exemplo, Ehrgott et al. (2016) e Przybylski e Gandibleux (2017). Esta tese se atém, contudo, ao estudo da utilização de metaheurísticas para a solução desta classe de problemas.

Conforme Dorigo e Stützle (2004), os métodos heurísticos são algoritmos que buscam obter boas soluções com baixo custo computacional, porém sem garantir que essas boas soluções obtidas sejam ótimas. Os métodos heurísticos estão divididos em duas classes: heurísticas construtivas e de refinamento. As heurísticas construtivas são métodos que constroem uma solução para o problema, sendo que a cada passo um único elemento da solução é inserido. O processo de construção de uma solução pode ser feito, em geral, de forma aleatória ou gulosa, ou ainda, de forma parcialmente

gulosa. As heurísticas de refinamento partem de uma ou mais soluções construídas no intuito de melhorá-las iterativamente.

As heurísticas de refinamento, por sua vez, se dividem em outras duas subclasses: heurísticas clássicas e metaheurísticas. As heurísticas clássicas trabalham com o conceito de vizinhança e consistem na exploração de forma iterativa sobre os vizinhos da solução corrente por meio de pequenas alterações nessa solução. Em geral, esses algoritmos encontram a melhor solução para aquele conjunto de soluções gerado pelas alterações consideradas. Em outras palavras, as heurísticas clássicas de refinamento encontram uma solução ótima local. Já as metaheurísticas, por outro lado, são métodos genéricos, que têm a capacidade de explorar o espaço de soluções sem ficarem presas a soluções ótimas locais. Cada metaheurística se diferencia por esse mecanismo de escape.

Uma solução é ótima local quando ela não possui nenhum vizinho melhor do que ela em relação à vizinhança adotada para explorar o espaço de soluções. Assim, em problemas de otimização combinatória multiobjetivo, o conceito de otimalidade local deve estendido para a noção de um conjunto de soluções Pareto ótimo local. Esse conjunto é composto pelas soluções que não contêm nenhum vizinho dominante, ou seja, as soluções cujos vizinhos são dominados ou incomparáveis. Da mesma forma que não há garantia de que uma solução ótima local é a solução ótima global de um problema mono-objetivo, também, o conjunto de soluções Pareto ótimo local não é necessariamente o conjunto Pareto ótimo global de um problema multiobjetivo.

Segundo Coello et al. (2010) metaheurísticas multiobjetivas podem ser classificadas em quatro classes de abordagem:

- Escalar: consiste em transformar o problema multiobjetivo em um ou vários problemas mono-objetivo.
- Populacional: consiste em explorar uma população de soluções, dividindo-a em subpopulações para combinar vários procedimentos de busca mono-objetiva. Cada subpopulação seleciona os melhores indivíduos com base em um único objetivo. Então, todas as subpopulações são mescladas.
- Baseada em Pareto: consiste em adotar o mecanismo de seleção que incorpora o conceito de otimização de Pareto.
- Baseada em indicadores: consiste em adotar uma medida de avaliação de desempenho na seleção de soluções.

2.3.2 NSGA-II e NSGA-III

A metaheurística mais difundida na resolução de problemas de otimização multi-objetivo é o NSGA-II, a qual é um algoritmo de busca populacional, isto é, trabalha com um conjunto de soluções a cada iteração. Ele foi proposto por Deb et al. (2002) como uma melhoria do NSGA (Srinivas e Deb, 1995), adicionando ao algoritmo um operador de distância de aglomeração, o que proporciona uma melhor diversidade das soluções não-dominadas. O Algoritmo 1 esquematiza o funcionamento do NSGA-II e está explicado a seguir.

Primeiramente, o algoritmo gera uma população inicial com $nPop$ indivíduos (linhas 1 a 3 do Algoritmo 1). Após essa primeira fase, o algoritmo passa para uma fase iterativa (linhas 4 a 13), que consiste em aplicar iterativamente sobre a população de indivíduos os procedimentos de cruzamento, mutação e seleção até que um critério de parada seja atendido. Na linha 7 do Algoritmo 1 é aplicado o processo de cruzamento sobre a população. Nesse procedimento, são gerados $nPop$ filhos por meio da escolha de dois pais para cada filho (linha 6). Após a seleção dos pais, o cruzamento é aplicado para gerar um filho com uma probabilidade $probCross$. A seleção dos pais e a aplicação do cruzamento são repetidas até que sejam gerados $nPop$ filhos.

Após o cruzamento, os filhos gerados são submetidos ao procedimento de mutação com probabilidade $probMut$. Na linha 12 é aplicado o método de seleção dos indivíduos sobreviventes para a próxima geração. Inicialmente, uma população expandida com $2nPop$ indivíduos é criada como resultado da união da população de pais e filhos. Todos os indivíduos da população expandida são avaliados com base na ordenação por fronteiras não-dominadas. A seguir, os indivíduos das fronteiras mais baixas irão formar a nova população até um limite máximo de $nPop$ indivíduos. Essa nova população é iniciada com os indivíduos da melhor frente não-dominada e continua com as soluções da segunda frente, depois com a terceira e assim por diante. Como o tamanho da população é fixo, nem todas as frentes poderão estar presentes na população sobrevivente. Sendo assim, quando a última frente é considerada para formar a população, pode existir um número de indivíduos nessa frente que ultrapasse o tama nho $nPop$. Caso isto ocorra, os indivíduos da última frente selecionada são eliminados por meio da distância de aglomeração, que é dada pelo perímetro do hipercubo formado pelas soluções vizinhas ao indivíduo que estão localizadas na mesma frente de dominância. Os indivíduos com menor distância de aglomeração são selecionados.

Em 2014, Deb e Jain (2014) propuseram o NSGA-III, como uma evolução do NSGA-II para resolução de problemas com mais de três objetivos. O algoritmo NSGA-III tem a mesma estrutura do algoritmo NSGA-II, diferindo deste em relação

Algoritmo 1 NSGA-II

Entrada: $nPop$, $probCross$, $probMut$

1: **para** $k \leftarrow 1$ to $nPop$ **faça**
2: $Pop_k \leftarrow$ gerarSoluçãoInicial();
3: **fim para**
4: **repita**
5: **para** $k \leftarrow 1$ to $nPop$ **faça**
6: $[Parent1, Parent2] \leftarrow$ selecionarPai(Pop)
7: $Offsprings_k \leftarrow$ aplicarCruzamento($Parent1, Parent2, probCross$);
8: **fim para**
9: **para** $k \leftarrow 1$ to $nPop$ **faça**
10: $Offsprings_k \leftarrow$ aplicarMutação($Offsprings_k, probMut$);
11: **fim para**
12: $Pop \leftarrow Selecao(Pop, Offsprings)$
13: **até** Critério de parada satisfeito

Saída: Pop;

Algoritmo 2 NSGA-III

Entrada: $nPop$, P, $probCross$, $probMut$

1: $Ref \leftarrow$ gerarPontosdeReferencia(P);
2: **para** $k \leftarrow 1$ to $nPop$ **faça**
3: $Pop_k \leftarrow$ gerarSoluçãoInicial();
4: **fim para**
5: **repita**
6: **para** $k \leftarrow 1$ to $nPop$ **faça**
7: $[Parent1, Parent2] \leftarrow$ selecionarPai(Pop)
8: $Offsprings_k \leftarrow$ aplicarCruzamento($Parent1, Parent2, probCross$);
9: **fim para**
10: **para** $k \leftarrow 1$ to $nPop$ **faça**
11: $Offsprings_k \leftarrow$ aplicarMutação($Offsprings_k, probMut$);
12: **fim para**
13: $Pop \leftarrow Selecao(Pop, Offsprings, Ref)$
14: **até** Critério de parada satisfeito

Saída: Pop;

ao mecanismo de seleção e está esquematizado no Algoritmo 2. Primeiramente, o algoritmo cria P pontos de referência (linha 1) e logo depois (linhas 2 a 4 do Algoritmo 2) gera uma população inicial com M indivíduos. Após essa primeira fase, o algoritmo passa para uma fase iterativa (linhas 5 a 14), que consiste em aplicar iterativamente sobre a população de indivíduos os procedimentos de cruzamento, mutação e seleção até que um critério de parada seja atendido. As operações de cruzamento e mutação são exatamente iguais aos do NSGA-II.

No procedimento de seleção do NSGA-III existe um conjunto de pontos de referência, os quais são parâmetros do algoritmo, e cada indivíduo da população é

associado ao ponto de referência com menor distância perpendicular. Os pontos que não possuem membros associados são excluídos. Logo após, o procedimento identifica o ponto de referência com o menor número de indivíduos associados a ele. Caso já existam indivíduos sobreviventes associado a esse ponto, o método seleciona aleatoriamente um outro indivíduo da última frente de dominância que esteja associado ao ponto de referência. Caso contrário, o indivíduo com menor distância perpendicular ao ponto é selecionado. O processo é repetido até que a população sobrevivente esteja completa.

Os pontos de referência são gerados sobre um hiperplano normalizado e igualmente inclinado para todos os eixos de objetivo. Esta metodologia busca evitar a aglomeração dos pontos de referência. Cada eixo é dividido em σ partes simétricas. Dessa forma, o número total de pontos de referência para um problema com ρ objetivos é dado por:

$$N_r = \begin{pmatrix} \rho + \sigma - 1 \\ \sigma \end{pmatrix}$$

Como exemplo, dado $\rho = 3$, os pontos são criados sobre um triângulo equilátero. Assim, para, por exemplo, $\sigma = 3$, tem-se um total de $N_r = 10$ pontos de referência para 3 divisões em cada eixo de objetivo, sendo eles:

$$(0.0000, 1.0000, 0.0000), (0.0000, 0.6667, 0.3333), (0.0000, 0.3333, 0.667),$$
$$(0.0000, 0.0000, 1.0000), (0.3333, 0.6667, 0.0000), (0.3333, 0.3333, 0.3333),$$
$$(0.3333, 0.0000, 0.6667), (0.6667, 0.3333, 0.0000), (0.6667, 0.0000, 0.6667),$$
$$(1.0000, 0.0000, 0.0000)$$

Com essa estratégia, os pontos de referência permanecem sempre espalhados no espaço de objetivos, preservando sua diversidade.

2.3.3 MO-VNS

Os algoritmos de busca local também tem sido utilizados para resolução de problemas combinatórios multiobjetivo, como por exemplo, a metaheurística MO-VNS (do inglês *Multi-Objective Variable Neighborhood Search*) (Geiger, 2008). Esse método é baseado na ideia de mudanças de vizinhanças sistemáticas com objetivo de encontrar uma solução ótima local.

O algoritmo 3 esquematiza o funcionamento desse método e está explicado a seguir. Inicialmente é gerado um conjunto D de soluções não-dominadas, o qual, posteriormente, passa por procedimentos de perturbação, busca local e alteração da vizinhança adotada, repetidamente, até que o critério de parada seja satisfeito. Isto é, a cada iteração do algoritmo são feitas modificações nas soluções, as quais passam

por mecanismos de refinamento e caso não haja melhoria nas soluções uma nova estrutura de vizinhança é considerada.

Algoritmo 3 MO-VNS

1: Gera conjunto inicial D de soluções não-dominadas
2: **repita**
3: $D' \leftarrow Perturbacao(D)$
4: $D'' \leftarrow BuscaLocal(D')$
5: Altera Vizinhança Adotada
6: **até** Critério de parada ser satisfeito

Saída: D;

A Figura 2.4 mostra o funcionamento básico do algortimo MO-VNS em duas dimensões. Inicialmente, são criadas soluções não-dominadas, representadas na figura em azul, a partir delas, através de um mecanismo de perturbação, são geradas novas soluções, representadas na figura em laranja. Assim, através de uma busca local, chega-se até um novo conjunto de soluções não-dominadas, representados na figura com a cor preta.

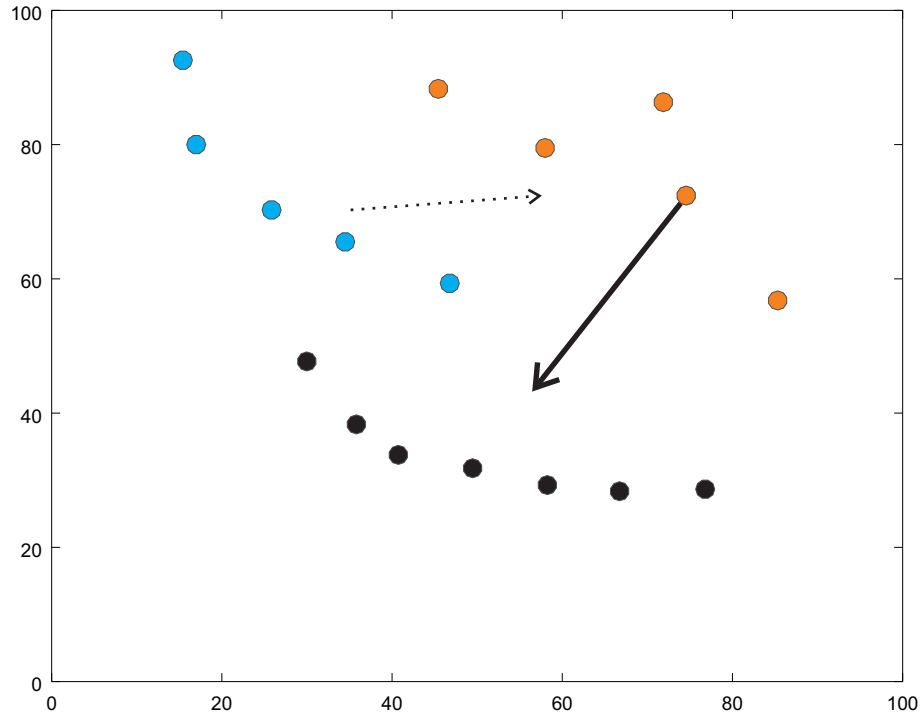


Figura 2.4: Funcionamento do MO-VNS

2.3.4 P-ILS

Outro algoritmo utilizado para resolver problemas multiobjetivo é o P-ILS (do inglês *Pareto Iterated Local Search*), o qual foi proposto por Geiger (2007) e está

esquematizado no algoritmo 4. Esse método inicia gerando uma solução inicial e adicionando a um conjunto D de soluções não-dominadas. Posteriormente, é aplicado uma busca local baseada numa vizinhança selecionada $\mathcal{N}_i$, gerando assim um novo conjunto D' de solução não-dominadas. Se houver melhora nesse conjunto uma nova solução s é considerada. Isso se repete até que todas as vizinhanças sejam visitadas. Depois disso, um procedimento de perturbação é aplicado ao conjunto D e procedimento de busca local é repetido até que o critério de parada seja satisfeito.

Algoritmo 4 P-ILS

```
1: Gera solução inicial s
2: D ← {s}
3: repita
4:    repita
5:       Selecione uma vizinhança N_i
6:       D' ← N_i(s)
7:       se houver melhora em D' faça
8:          Selecione uma nova solução s ∈ D'
9:       fim se
10:   até todas as vizinhanças sejam visitadas
11:   D' ← Perturbacao(D)
12:   Selecione uma solução s ∈ D'
13: até Critério de parada seja satisfeito
Saída: D;
```

A Figura 2.5 mostra o funcionamento básico do algoritmo P-ILS em duas dimensões. Inicialmente, é criada uma solução, representada na figura na cor azul, assim, através de busca local, chega-se a um conjunto de soluções não-dominadas, representadas na figura em branco. Depois, uma das soluções é escolhida, e então aplicado a ela um mecanismo de perturbação, a solução resultante está representada na figura na cor laranja, a partir dessa solução, através de uma busca local, chega-se até um novo conjunto de soluções não-dominadas, representados na figura com a cor preta.

2.4 Medidas de Desempenho

A desvantagem da aplicação de uma metaheurística é o fato dessa técnica não garantir otimalidade, ou seja, o resultado obtido pelo algoritmo heurístico não é necessariamente o conjunto de soluções mais eficientes do problema. Porém, é possível ter uma estimativa de quão próximo o resultado de metaheurística possa estar das soluções Pareto ótimo. A qualidade dos resultados encontradas por uma heurística pode ser medido de diversas formas, e existem muitas métricas de qualidades diferen-

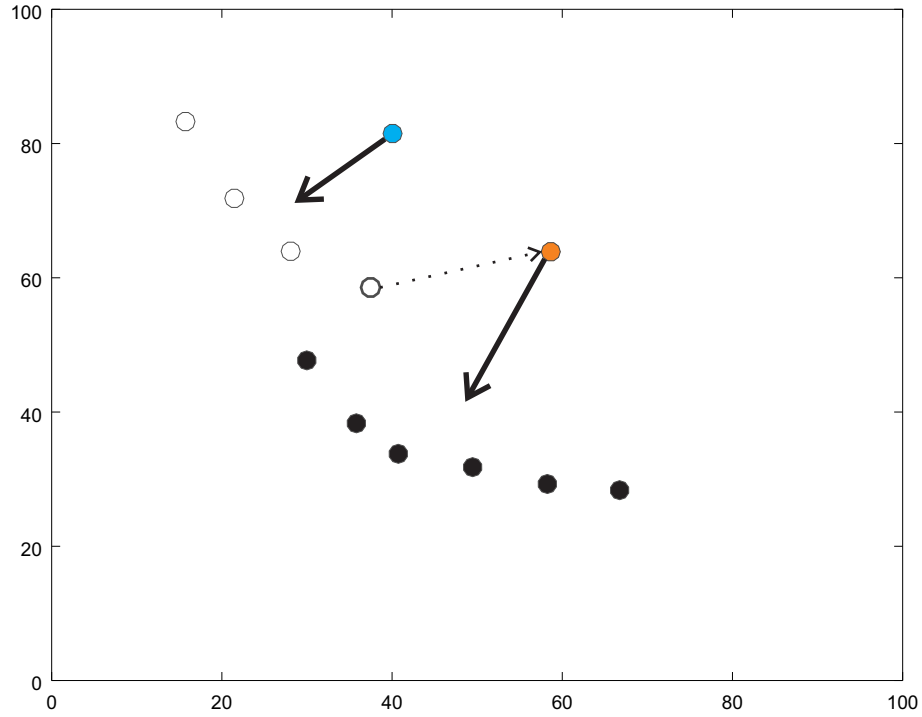


Figura 2.5: Funcionamento do P-ILS

tes na literatura. Em geral, as métricas medem dois aspectos: a convergência para o Pareto ótimo e a diversidade entre as soluções.

A Figura 2.6 mostra quais são as aplicações dos aspectos considerados pelas métricas de qualidade. Considerando nesta figura que os pequenos pontos representam a fronteira de Pareto ótimo para um problema com dois objetivos conflitantes, os círculos representam as soluções encontradas pelo Algoritmo A e os asteriscos pelo Algoritmo B, pode-se dizer que A é melhor B no aspecto de convergência, pois suas soluções estão mais próximas das soluções de Pareto. Porém, no aspecto diversidade, B é melhor que A, pois suas soluções estão mais bem espaçadas entre si.

Assim, existem diversas métricas de qualidades propostas na literatura. Algumas delas são medidas de diversidade, como por exemplo, o indicador *Spacing*, proposto por Schott (1995) e a métrica HCC, do inglês *Hierarquical Cluster Counting*, proposta por (Guimaraes et al., 2009). Existem também métricas que medem a convergência das soluções, como por exemplo, o indicador *Epsilon* proposto por Zitzler et al. (2003). Outras métricas são tanto medidas de diversidade quanto convergência, é o caso da métrica *Hypervolume*, proposta por Zitzler e Thiele (1998).

Segundo Ehrgott e Gandibleux (2008), não foi encontrada uma métrica universalmente adotada na literatura de otimização multiobjetivo, porém os autores deixam claro que é necessário mais investigação.

As medidas de desempenho utilizadas neste trabalho para avaliar as soluções

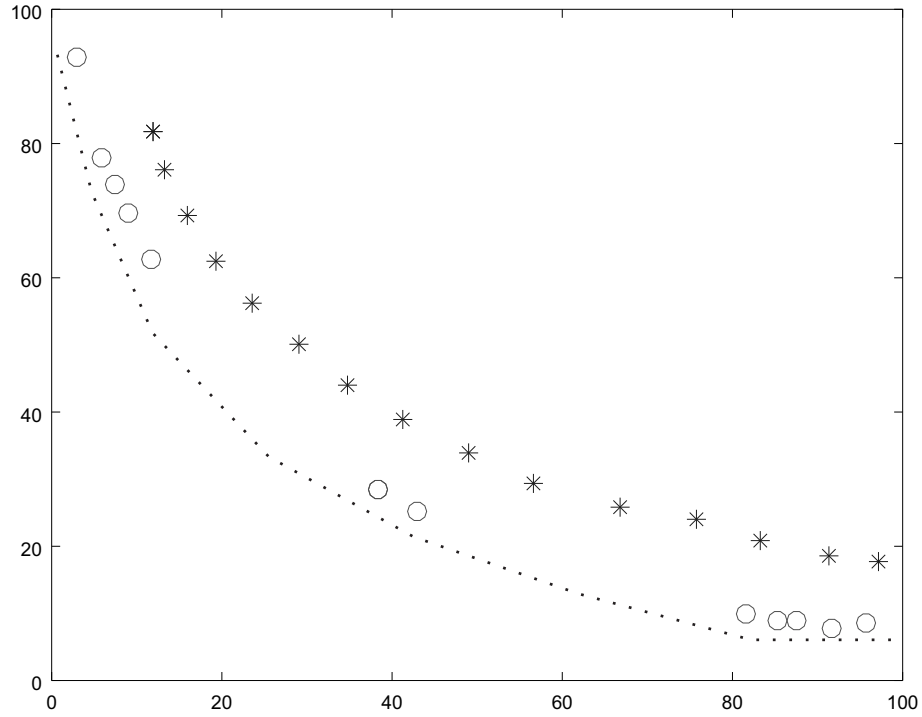


Figura 2.6: Convergência e Diversidade

geradas pelos algoritmos foram:

- *Hypervolume*, $H(Q_{alg}, R_0)$: Proposta por Zitzler e Thiele (1998), essa métrica mede o volume entre a fronteira de Pareto Q_{alg} obtida pelo algoritmo alg e um ponto de referência R_0. Uma área com maior volume indica tanto um maior espalhamento das soluções quanto uma maior convergência das soluções. Assim, quanto maior o *Hypervolume*, melhor o algoritmo. O ponto de referência R_0 foi definido, para cada instância, como o maior valor dos objetivos contidos no conjunto de união de todas as soluções encontradas após todos os experimentos realizados neste trabalho.

- *Epsilon*, $I_e(Q_{alg}, Q_{ref_i})$: Desenvolvido por Zitzler et al. (2003), esse indicador determina um fator mínimo e que, se multiplicado por cada ponto do conjunto Q_{ref_i}, torna o conjunto de aproximações resultante fracamente dominado por Q_{alg}. Seja Q_{ref_i} a fronteira ideal para Pareto da instância i. Portanto, quanto menor o valor de I_e, maior a convergência do algoritmo. O conhecimento prévio da fronteira ótima de Pareto é necessário para o cálculo deste indicador. No entanto, em muitos problemas apresentados na literatura, esse conjunto não é conhecido. Assim, para avaliar a convergência dos algoritmos, um conjunto de referência é geralmente definido e usado como a fronteira ideal para Pareto. Neste trabalho, o conjunto de referência Q_{ref_i} para cada instância i contém os

pontos não-dominados do conjunto de união de todos os experimentos realizados com a instância i.

- *Spacing*: Proposto por Schott (1995), esse indicador estima a variância das distâncias das soluções vizinhas pertencentes à frente Q_{alg} das soluções não-dominadas obtidas pelo algoritmo alg (Okabe et al., 2003; Yen e He, 2014). Para cada solução, a distância entre a mais próxima é encontrada. O valor retornado por essa métrica é o desvio-padrão das distâncias encontradas. Valores próximos de zero indicam altos níveis de uniformidade entre as soluções.

- *Hierarquical Cluster Counting* (HCC): Nessa métrica, os elementos do conjunto de soluções não dominadas são agrupados em *clusters*, que são unidos até que todos os pontos estejam em apenas uma classe (Guimaraes et al., 2009). As distâncias de fusão dos *clusters* são somadas em um único valor. A métrica HCC é capaz de avaliar tanto a extensão quanto a uniformidade do conjunto de estimativas, tornando-se uma métrica de diversidade adequada para problemas multiobjetivo.

Capítulo 3

Problemas de Sequenciamento de Tarefas

Este capítulo apresenta a notação utilizada e descreve os principais problemas de sequenciamento de tarefas, entre eles, o problema *Flow Shop* Híbrido. Na Seção 3.1 é apresentada uma notação geral. As Seções 3.2, 3.3 e 3.4 tratam dos ambientes de máquinas, das características e restrições dos ambientes e dos critérios de avaliação, respectivamente. A Seção 3.5 apresenta exemplos de problemas clássicos. Por fim, a Seção 3.6 descreve o problema *Flow Shop* Híbrido, enquanto a Seção 3.7 apresenta uma revisão bibliográfica sobre problemas dessa natureza.

3.1 Notações

Problemas de sequenciamento de tarefas tratam da alocação temporal de recursos escassos para a realização de um conjunto de tarefas. Dado um conjunto de tarefas que necessitam ser processadas em um determinado ambiente, um problema dessa natureza consiste em sequenciar as tarefas, respeitando-se um conjunto de restrições, de tal modo que uma ou mais medidas de desempenho sejam otimizadas.

Esses recursos escassos, que são genericamente chamados de máquinas, podem ser equipamentos em uma indústria, computadores em uma rede, equipes de trabalho, dentre outros. Do mesmo modo, as tarefas podem ser operações em um processo de fabricação, programas a serem executados, decolagens e pousos em aeroportos, cirurgias em centro cirúrgico, etc.

Segundo Pinedo (2008), sequenciamento de tarefas é um processo de decisão que é usado em muitas indústrias e serviços. Esse problema trata da alocação de recursos para tarefas durante períodos de tempo dados e envolve a otimização de um ou mais objetivos.

Será utilizada a seguinte notação geral:

(i) m: número de máquinas (ou estágios);

(ii) n: número de tarefas;

(iii) (i, j): índices para o processamento da tarefa j na máquina i (operação);

(iv) p_{ij}: tempo de processamento da operação (i, j);

(v) p_j: tempo de processamento da tarefa j;

(vi) d_j: data requerida para a entrega (*due date*) da tarefa j;

(vii) w_j: custo por unidade de atraso da tarefa j;

(viii) u_j: custo por unidade de antecipação da tarefa j;

(ix) C_{ij}: instante de conclusão da i-ésima operação da tarefa j;

(x) C_j: instante de conclusão da tarefa j (última operação);

(xi) $L_j = C_j - d_j$: afastamento (*lateness*) da tarefa j;

(xii) $T_j = \max(C_j - d_j, 0)$: atraso (*tardiness*) da tarefa j;

(xiii) $E_j = \max(d_j - C_j, 0)$: antecipação (*earliness*) da tarefa j.

Os problemas de sequenciamento de tarefas podem ser descritos por uma tripla $\alpha \mid \beta \mid \gamma$. O campo α descreve o ambiente de máquinas e contém apenas uma entrada. O campo β fornece detalhes sobre as características e restrições de processamento e pode conter nenhuma entrada, uma única entrada, ou entradas múltiplas. O campo γ, por sua vez, descreve o objetivo a ser otimizado, e pode conter uma única entrada (mono-objetivo) ou mais de uma entrada (multiobjetivo). As seções 3.2, 3.3 e 3.4 a seguir, mostram entradas possíveis para os campos α, β e γ, respectivamente:

3.2 Ambientes de Máquinas

De acordo com Pinedo (2008), são os seguintes os possíveis ambientes de máquinas:

(i) Máquina única (1): o caso de uma única máquina é o mais simples de todos os ambientes de máquina possível e é um caso especial dentre todos os outros ambientes de máquinas. A Figura 3.1 mostra as n tarefas em fila para o processamento na única máquina disponível. Cada tarefa é representada por uma elipse colorida e a máquina é representada por um quadrado cinza;

Figura 3.1: Ambiente de Máquina Única

(ii) Máquinas paralelas idênticas (P_m): há m máquinas idênticas em paralelo. Cada tarefa requer uma operação e podem ser executadas em qualquer uma das máquinas ou em qualquer uma que pertença a um dado subconjunto de máquinas;

(iii) Máquinas paralelas uniformes (Q_m): há m máquinas em paralelo com velocidades diferentes. A velocidade da máquina l é denotada por v_l. O tempo que a tarefa j gasta na máquina l é nomeado como p_{lj}, e é igual a p_j/v_i. Se todas as máquinas têm a mesma velocidade, então o ambiente é de máquinas paralelas idênticas;

(iv) Máquinas paralelas não-relacionadas (R_m): há m máquinas diferentes em paralelo. A máquina l pode processar a tarefa j na velocidade v_{lj}. O tempo de processamento p_{lj} da tarefa j na máquina l é igual a p_j/v_{ij}. Se as velocidades das máquinas são independentes das tarefas, então o ambiente é de máquinas paralelas uniformes. A Figura 3.2 ilustra um ambiente com m máquinas paralelas e n tarefas. Cada tarefa é representada por uma elipse colorida e as máquinas são representadas por quadrados cinzas;

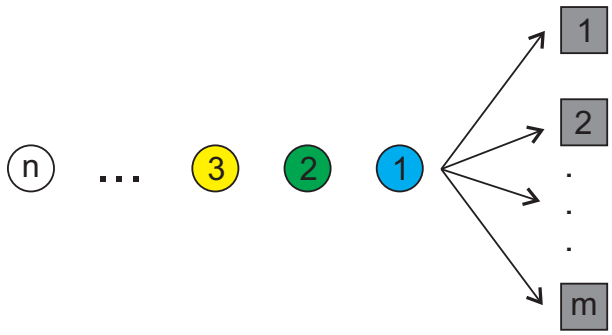


Figura 3.2: Ambiente de Máquinas Paralelas

(v) *Flow Shop* (F_m): existem m máquinas em série. Cada tarefa deve ser executada em cada uma das máquinas. Todas as tarefas seguem o mesmo fluxo, ou seja, são processadas primeiro na máquina 1, em seguida na máquina 2, e assim por diante. A Figura 3.3 ilustra esse ambiente com as tarefas (elipses coloridas) passando por todas as máquinas (quadrados cinzas) na mesma rota, uma após a outra;

(vi) *Flow Shop* Híbrido (FF_c): o ambiente *Flow Shop* Híbrido é uma generalização do *Flow Shop* e dos ambientes de máquinas paralelas. Ao invés de máquinas em

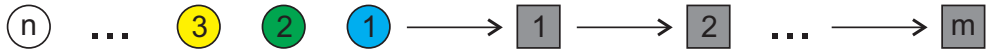


Figura 3.3: Ambiente *Flow Shop*

série, existem c estágios em série, de modo que cada estágio possui um número de máquinas em paralelo. Cada tarefa é executada em primeiro lugar no estágio 1, em seguida no estágio 2, e assim por diante. Conforme mostra a Figura 3.4, toda tarefa (elipse colorida) segue o mesmo fluxo de estágios e em cada estágio existe um conjunto de máquinas (quadrados cinzas) paralelas;

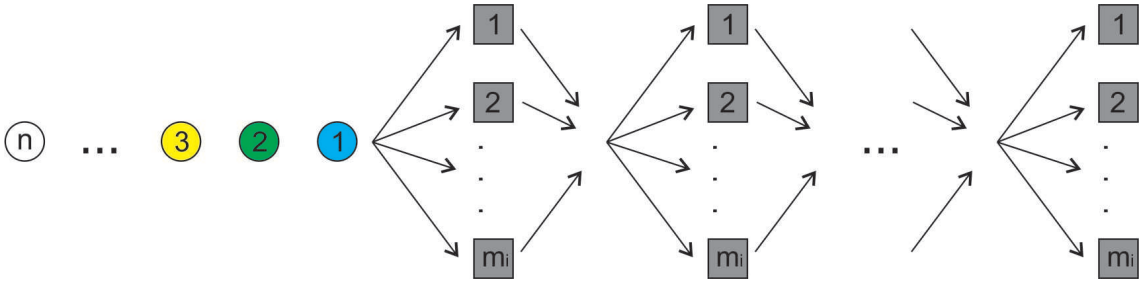


Figura 3.4: Ambiente *Flow Shop* Híbrido

(vii) *Job Shop* (J_m): neste ambiente cada tarefa deve ser executada em cada uma das m máquinas, e tem o seu próprio fluxo de máquinas pré-determinado;

(viii) *Job Shop* Híbrido (FJ_c): é uma generalização do ambiente *Job Shop* e dos ambientes de máquinas paralelas. Em vez de máquinas em série, existem c estágios em série, de modo que cada estágio possui um número de máquinas idênticas em paralelo;

(ix) Open Shop (O_m): No *Open Shop* cada tarefa deve ser executada em cada uma das m máquinas. No entanto, não existem restrições sobre o fluxo de máquinas em que as tarefas devem ser executadas.

3.3 Características e Restrições

Alguns exemplos de características e restrições de processamento que o campo β pode conter são listados a seguir:

(i) Tempos de *Release* (r_j): a tarefa j não pode iniciar o seu processamento antes do seu tempo de *release* r_j;

(ii) Restrições de precedência ($prec$): as restrições de precedência exigem que uma ou mais tarefas estejam concluídas antes do inicio do processamento de uma

outra tarefa. Existem várias formas especiais de restrições de precedência: se cada tarefa tem, no máximo, um antecessor e, no máximo, um sucessor, as restrições são chamadas de cadeias. Se cada tarefa tem no máximo um sucessor, as restrições são chamadas de *intree*. Se cada operação tem no máximo um antecessor, as restrições são chamadas de *outtree*;

(iii) Tempos de preparação (*Setup*) dependentes da Sequência (s_{ljk}): s_{ljk} representa o tempo de preparação (*setup*) da máquina l quando a tarefa j precede a tarefa k. s_{ljk} pode ser diferente de s_{lkj}, pois o tempo de preparação depende da sequência que está sendo executada. Quando os tempos de preparação são independentes da sequência, eles são incluídos nos tempos de processamento das tarefas;

(iv) Elegibilidade de Máquinas (M_j): a elegibilidade de máquinas indica que nem todas as máquinas são capazes de processar a tarefa j. M_j é o conjunto de máquinas que podem executar a tarefa j;

(v) Permutação (*prmu*): essa restrição só aparece em ambientes *Flow Shop*. Ela implica que a ordem (permutação) em que as tarefas passam pela primeira máquina (ou estágio) é mantida em todas as outras máquinas(ou estágios);

(vi) Bloqueio (*block*): o bloqueio também é uma situação que só pode ocorrer em *Flow Shop*. Essa restrição implica que a máquina que está executando uma operação de uma dada tarefa não é liberada enquanto a mesma não inicie a sua próxima operação, ou seja, a máquina fica bloqueada;

(vii) *no-Wait* (*nwt*): essa característica também só ocorre em ambientes *Flow Shop*, e significa que as tarefas não podem ficar esperando entre duas operações sucessivas, ou seja, assim que terminar uma operação, a próxima deverá iniciar.

3.4 Critérios de Avaliação

O campo γ descreve os critérios de avaliação na definição de um problema de sequenciamento de tarefas. Existem diversos critérios que podem ser objeto de interesse em problemas dessa classe, como, por exemplo:

(i) Minimização do *Makespan*: o *makespan* é o instante de tempo em que a última tarefa é completada e é dado por $C_{\max} = \max C_1, C_2, \ldots, C_n$;

(ii) Minimização do somatório ponderado do atraso: uma tarefa sofre um atraso caso seu tempo de conclusão seja maior que a data de entrega desejável. Nesse caso, a cada tarefa é atribuído um custo w_j, de acordo com a importância da tarefa. Esse critério é dado por $\sum w_j T_j$ onde $T_j = max\{C_j - d_j, 0\}$;

(iii) Minimização do somatório ponderado da antecipação: uma tarefa sofre um atraso caso seu tempo de conclusão seja menor que a data de entrega desejável. . Nesse caso, a cada tarefa é atribuído um custo u_j, de acordo com a importância da tarefa. Esse critério é dado por $\sum u_j E_j$ onde $E_j = max\{d_j - C_j, 0\}$;

(iv) Minimização do somatório do tempo de ociosidade: a ociosidade de cada máquina é a quantidade de tempo durante o processamento no qual ela ficou sem executar nenhuma tarefa até a conclusão da última tarefa. Esse critério é dado por $\sum I_l$ sendo $I_l = (\max C_{jl}) - \sum p_{jl}$.

(v) Minimização do número de tarefas atrasadas: o número de tarefas atrasadas é quantidade de tarefas que tiveram o seu tempo de conclusão maior que sua data entrega. Esse critério é dado por $\sum U_j$ onde $U_j = 1$ se $T_j > 0$ e $U_j = 0$, caso contrário;

(vi) Minimização do Maior afastamento: o afastamento de uma tarefa é a quantidade de tempo que separa seu instante de conclusão de sua data de entrega. Esse critério é definido por $L_{\max} = \max L_1, L_2, \ldots, L_n$ onde $L_j = C_j - d_j$;

3.5 Exemplos de problemas clássicos

Esta seção apresenta exemplos de problemas clássicos, combinando os ambientes de máquinas, com as características e restrições, e critérios de avaliação.

Exemplo 1 *O primeiro exemplo é de um ambiente de máquina única com objetivo de minimização da soma ponderada do atraso. Consideram-se quatro tarefas com os seguintes tempos de processamento:* 6, 8, 11 *e* 12, *e as seguintes datas de entrega* (d_j) *e custos* (w_j), *respectivamente:* 14 *e* 5, 24 *e* 7, 27 *e* 4 *e* 32 *e* 2.

Uma possível solução para esse problema está esquematizada na Figura 3.5. Nesse caso, $\sum w_j T_j = w_2 \times T_2 + w_4 + T_4 = 7 \times 1 + 2 \times 5 = 17$.

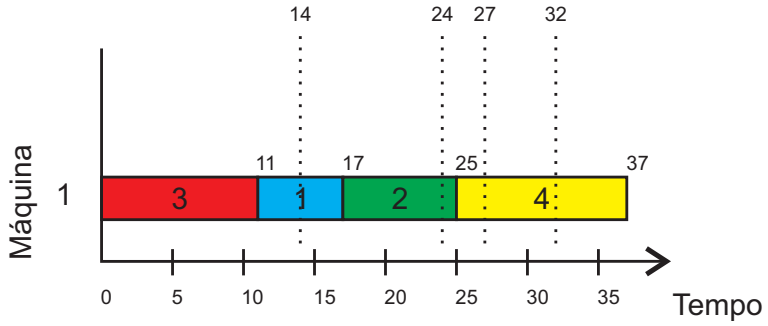


Figura 3.5: Diagrama de Gantt - Máquina única. $\sum w_j T_j = 17$

Exemplo 2 *O segundo exemplo é de um ambiente com duas máquinas paralelas e cinco tarefas para minimização do* makespan. *A Tabela 3.1 mostra os tempos de processamento de cada tarefa j em cada máquina i. Nesta tabela pode-se verificar que o tempo de processamento da tarefa* 3 *na máquina* 2 *é de* 9 *unidades. A Figura 3.7 mostra uma possível solução para esse exemplo. Nesse caso,* $\sum C_{\max} = 27$.

Tabela 3.1: Tempo de processamento - Exemplo 2

	i	1	2
j	1	12	4
	2	3	10
	3	6	9
	4	8	8
	5	11	6

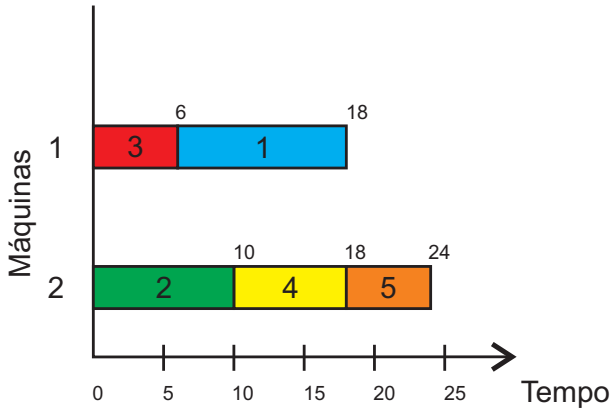


Figura 3.6: Diagrama de Gantt - Máquinas Paralelas. $\sum C_{max} = 27$

Exemplo 3 *Outro exemplo é de um ambiente* Flow Shop *com a minimização da soma ponderada dos tempos de conclusão. Considere três tarefas e dois estágios, as tarefas têm os seguintes custos (v_j):* 5, 8 *e* 3. *A Tabela 3.2 mostra os tempos de processamento de cada tarefa j em cada máquina i . Nesta tabela pode-se verificar que o tempo de processamento da tarefa* 1 *na máquina* 2 *é de* 7 *unidades.*

A Figura 3.7 representa um possível sequenciamento para a situação apresentada. Nesse caso $\sum v_j C_j = v_1 \times C_1 + v_2 \times C_2 + v_3 \times C_3 = 5 \times 20 + 8 \times 27 + 3 \times 16 = 100 + 216 + 48 = 364$.

3.6 Problema *Flow Shop* Híbrido

Segundo Pinedo (2008), em muitos processos de produção, cada tarefa deve passar por uma série de operações, as quais devem ser realizadas na mesma ordem de

Tabela 3.2: Tempo de processamento - Exemplo 3

	i	1	2
j	1	5	4
	2	3	7
	3	8	8

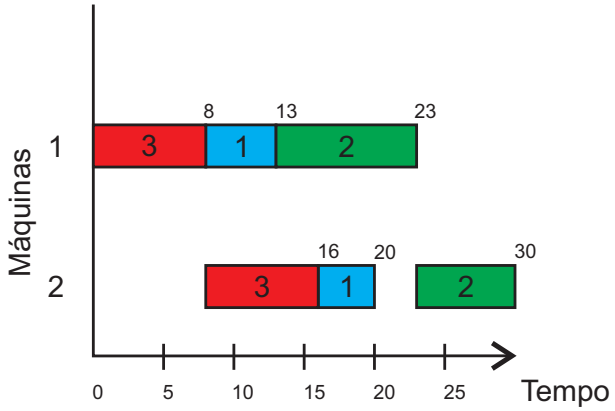


Figura 3.7: Diagrama de Gantt - *Flow Shop*. $\sum v_j C_j = 364$

produção, implicando que as tarefas tenham que seguir a mesma rota. Esse tipo de ambiente de máquinas em série é denominado *Flow Shop*.

Em um problema *Flow Shop*, tem-se um conjunto de tarefas, que devem ser processadas em um conjunto de máquinas em série. Nesse ambiente, o fluxo de máquinas é sempre o mesmo para todas as tarefas do sistema, isto é, cada tarefa deve ser processada em estágios. No *Flow Shop* clássico, cada estágio possui apenas uma máquina para realizar o processamento de todas as tarefas, as quais visitam o estágio exatamente uma vez.

Uma característica importante de ambientes de máquinas em série é a necessidade de armazenamento (em inglês *buffer*) entre estágios, isto é, caso uma tarefa termine no estágio anterior antes que o próximo estágio esteja disponível, existe, então, a necessidade de que o produto seja armazenado em algum lugar até que se possa ser processado. A Figura 3.8 ilustra um caso no qual a tarefa k não precisa de armazenamento, enquanto a Figura 3.9 ilustra um caso no qual a tarefa k precisa de armazenamento por 3 unidades de tempo.

No mundo real, se uma etapa em um processo de produção é um gargalo, pode-se adicionar, a essa etapa, outras máquinas paralelas, podendo assim agilizar o processo e impedir a sobrecarga em uma única máquina. Esse ambiente de máquinas é muitas vezes referido na literatura como *Flow Shop* Híbrido ou *Flow Shop* Flexível.

O ambiente *Flow Shop* Híbrido (HFS, das iniciais em inglês *Hybrid Flow Shop*), por sua vez, traz em suas características a possibilidade de existir mais de uma

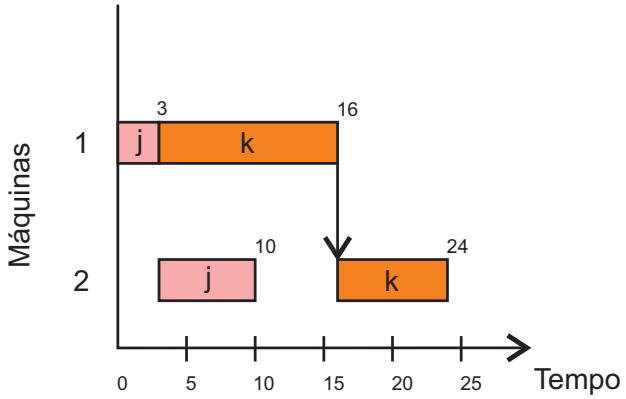


Figura 3.8: Diagrama de GANTT - Sem *buffer*

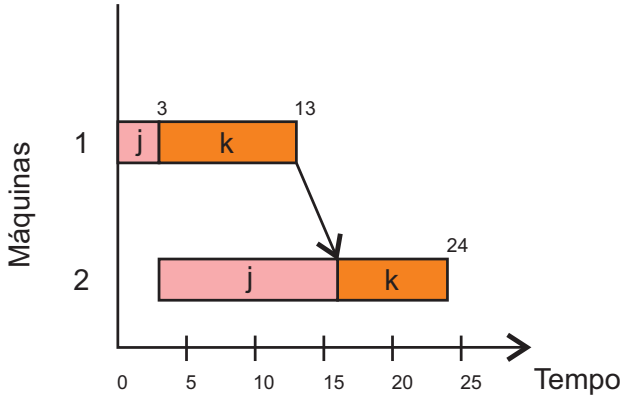


Figura 3.9: Diagrama de GANTT - Com *buffer*

máquina em cada estágio. Cada tarefa deve ser processada em cada estágio somente em uma das máquinas. Além disso, é possível também que tarefas saltem estágios, isto é, algumas tarefas específicas podem não visitar todos os estágios.

Portanto, em um problema HFS, tem-se um conjunto de tarefas que devem ser executadas em um conjunto de estágios, os quais possuem um conjunto de máquinas paralelas. Nota-se que o HFS é uma generalização do ambiente *Flow Shop*, que combina características dos ambientes de máquinas em série e paralelas. O Exemplo 4 caracteriza essa formulação de problema.

Exemplo 4 *Considere o seguinte exemplo, com quatro tarefas, dois estágios e duas máquinas em cada estágio. Seja* p_{ilj} *o tempo de processamento da tarefa* j *na máquina* l *e estágio* i*. A Tabela 3.3 mostra os tempos de processamento para este exemplo. O tempo de processamento da tarefa* 1 *na máquina* 1 *no primeiro estágio é* 6 *unidades de tempo, enquanto na máquina* 2 *é de* 18 *unidades de tempo.*

A Figura 3.10 mostra uma possível sequência para este exemplo.

Tabela 3.3: Tempos de Processamento - Exemplo 4.

	i	1		2	
	l	1	2	3	4
j	1	6	18	20	14
	2	8	20	10	16
	3	14	20	6	10
	4	16	6	8	20

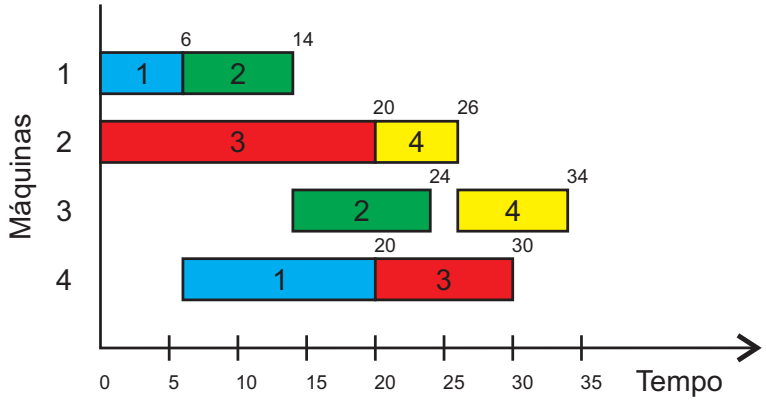


Figura 3.10: Diagrama de GANTT - Exemplo 4.

3.7 Problemas *Flow Shop* Híbrido: revisão bibliográfica

O primeiro trabalho sobre problemas *Flow Shop* híbrido (HFS, do inglês *Hybrid Flow Shop*) surgiu na década de 1950, com Johnson (1954). Desde então, esse problema de sequenciamento de tarefas mono-objetivo é abordado em vários trabalhos da literatura. Por exemplo, Sriskandarajah e Sethi (1989), Gupta e Tunc (1991) e Hoogeveen et al. (1996) tratam o HFS com dois estágios para minimização do *makespan*, e propõem heurísticas de construção para resolução do problema.

O primeiro trabalho a tratar o HFS com um número variável de estágios foi Wittrock (1985). Outros autores fizeram o mesmo, Rajendran e Chaudhuri (1992) desenvolveram algoritmos *Branch and Bound* para minimizar o *makespan* e Vignier et al. (1996) trataram o HFS para minimização do tempo total de conclusão das tarefas. O HFS é tratado com máquinas parelelas não-relacionadas em trabalhos como os de Hayrinen et al. (2000), Jungwattanakit et al. (2005), Low (2005) e Jenabi et al. (2007).

Naderi et al. (2010) apresentam um algoritmo baseado na metaheurística *Iterated Local Search* – ILS (Lourenço et al., 2003), para minimizar o *makespan*. Urlings e Ruiz (2010) propuseram Algoritmos Genéticos (Holland, 1975) para solucionar o HFS, com as características propostas por Ruiz et al. (2008), com objetivo de minimizar

do *makespan*. Zandieh et al. (2010) também trabalharam com o mesmo problema e propuseram outro Algoritmo Genético. Em Defersha e Chen (2012), os autores tratam o HFS com Algoritmos Genéticos em plataformas de computação paralelas e sequenciais. Os testes mostraram que a implementação paralela melhorou muito o desempenho computacional das heurísticas desenvolvidas. Em de Siqueira et al. (2013), um algoritmo baseado em Estratégias Evolutivas é proposto para tratar o problema HFS, também para a minimização do *makespan*. Nesses trabalhos, diversas características de problemas reais são tratadas, como restrições de precedência e tempos de prep aração dependentes da sequência.

Chamnanlor et al. (2017) abordaram a minimização do *makespan* em um problema HFS com restrições de janela de tempo e, para resolvê-lo, propuseram um algoritmo híbrido que combina métodos de algoritmos genéticos e colônia de formigas (Dorigo e Stützle, 2004). Em Dios et al. (2018), o problema HFS é abordado, considerando em especial o salto de estágios. Os autores propuseram um conjunto de heurísticas construtivas para minimizar o *makespan*.

Capítulo 4

Problema Flow Shop Híbrido multiobjetivo

Neste capítulo o problema abordado é apresentado e detalhado, apresentando-se também uma revisão da literatura. A Seção 4.1 apresenta a descrição do problema *Flow Shop* Híbrido. A Seção 4.2 traz uma formulação de programação matemática para esse problema. As Seções 4.3 e 4.4 apresenta os problemas tratados nesta Tese, envolvendo três objetivos e cinco objetivos, respectivamente. A Seção 4.5 traz uma revisão bibliográfica a respeito de problemas *Flow Shop* multiobjetivo. Por fim, a Seção 4.6 mostra exemplos do problema *Flow Shop* híbrido multiobjetivo.

4.1 Problema *Flow Shop* Híbrido multiobjetivo

O problema aqui tratado, denominado Problema *Flow Shop* Híbrido Multiobjetivo (MOHFS, das iniciais em inglês *Multi-Objective Hybrid Flow Shop*), é definido a seguir. Seja $N = \{1, 2, 3, \cdots, n\}$ um conjunto de tarefas que devem ser executadas em um conjunto de estágios $M = \{1, 2, 3, \cdots, m\}$. Para cada estágio i, há um conjunto de máquinas paralelas não-relacionadas. Algumas tarefas podem saltar estágios e essa é uma propriedade importante desse problema. O processamento da tarefa j no estágio i é chamado de operação. Uma situação comum na prática é abordada, na qual algumas tarefas só podem ser executadas em determinadas máquinas especializadas, as quais, por sua vez, só podem executar um determinado grupo de tarefas. Essa característica é chamada de elegibilidade de máquinas. As principais características do problema são:

(i) F_j: conjunto de estágios visitados pela tarefa j, $1 < |F_j| < m$;

(ii) p_{ilj}: tempo de processamento da tarefa j na máquina l e estágio i;

(iii) G_{ij}: conjunto de máquinas elegíveis para a tarefa j no estágio i;

(iv) d_j: data de entrega (*due date*) da tarefa j;

(v) w_j: custo por tempo de atraso da tarefa j;

(vi) u_j: custo por tempo de antecipação da tarefa j;

Os itens (i), (ii) e (iii) indicam que as máquinas disponíveis em cada estágio são independentes entre si. Além disso, algumas tarefas podem saltar estágios, o que é muito comum em vários processos industriais. Se uma etapa em um processo de produção é um gargalo, os gerentes geralmente adicionam máquinas paralelas a essa etapa para diminuir esse gargalo. Essas máquinas geralmente são diferentes umas das outras e algumas não executam todos os tipos de operações. Da mesma forma, alguns produtos específicos podem pular etapas do processo. Os itens (iv), (v) e (vi) consideram outra situação encontrada na prática, na qual os produtos têm prazos de entrega desejáveis. Se a tarefa terminar após o prazo, haverá um custo normalmente relacionado às multas contratuais. Por outro lado, se a tarefa terminar antes do prazo, o custo geralmente se refere ao armazenamento dos produtos. Os custos de estocagem são geralmente mais baixos do que os valores de multa por atraso.

Essas características estão formuladas matematicamente a seguir, na Seção 4.2.

4.2 Formulação Matemática

A formulação matemática do MOHFS apresentada a seguir foi adaptada de Urlings (2010), adicionando a minimização da soma ponderada da antecipação. Os parâmetros do modelo são definidos como:

$$
\begin{aligned}
N &= \text{conjunto de tarefas;} \\
M &= \text{conjunto de estágios} \\
M_i &= \text{conjunto de máquinas no estágio } i; \\
n &= \text{quantidade de tarefas, isto é, } n = |N|; \\
m &= \text{quantidade de estágios, isto é, } m = |M|; \\
m_i &= \text{quantidade de máquinas no estágio } i, \text{ isto é, } m_i = |M_i|; \\
p_{ilj} &= \text{tempo de processamento da tarefa } j \text{ na máquina } l \text{ e estágio } i; \\
d_j &= \text{data de entrega da tarefa } j;
\end{aligned}
$$

As variáveis de decisão são definidas como:

$$
\begin{aligned}
X_{jki} &= \begin{cases} 1, & \text{se a tarefa } j \text{ precede a tarefa } k \text{ no estágio } i; \\ 0, & \text{caso contrário} \end{cases} \\
Y_{jil} &= \begin{cases} 1, & \text{se a tarefa } j \text{ é executada na máquina } l \text{ no estágio } i; \\ 0, & \text{caso contrário} \end{cases} \\
C_{ij} &= \text{instante de conclusão da tarefa } j \text{ no estágio } i \\
C_{\max} &= \text{maior tempo de conclusão} \\
I_l &= \text{tempo de ociosidade da máquina } l; \\
U_j &= 1, \text{se } T_j > 0; \text{ e } U_j = 0, \text{ caso contrário.}
\end{aligned}
$$

O modelo de programação matemática do MOHFS pode ser representado pelas Equações (4.1) a (4.14), mostradas a seguir:

$$
\begin{aligned}
\min \quad & C_{\max} && (4.1)\\
\min \quad & \sum_{j=1}^{n} w_j T_j && (4.2)\\
\min \quad & \sum_{j=1}^{n} u_j E_j && (4.3)\\
\min \quad & \sum_{l=1}^{m_i} I_l && (4.4)\\
\min \quad & \sum_{j=1}^{n} U_j && (4.5)\\
\text{sujeito a} \quad & \sum_{l=1}^{m_i} Y_{jil} = 1, j \in N, i \in M && (4.6)\\
& \sum_{l=1}^{m_i} Y_{jil} . p_{ilj} \leq C_{ij} - C_{i-1,j}, j \in N, i \in M && (4.7)\\
& V(2 - Y_{jil} - Y_{kil} + X_{jki}) + C_{ij} - C_{i,k} \geq p_{ilj}, j, k \in N, j < k && (4.8)\\
& V(3 - Y_{jil} - Y_{kil} - X_{jki}) + C_{ij} - C_{i,k} \geq p_{ilj}, j, k \in N, j < k && (4.9)\\
& C_{0j} = 0, j \in N && (4.10)\\
& Y_{jil} \in \{0, 1\}, j \in N, i \in M, l \in M_i && (4.11)\\
& X_{jki} \in \{0, 1\}, j, k \in N, i \in M && (4.12)\\
& C_{ij} \geq 0, , j \in N, i \in M && (4.13)\\
& C_{\max} \geq C_{mj}, j \in N && (4.14)
\end{aligned}
$$

no qual V é um valor positivo muito alto.

As Expressões (4.1) a (4.5) definem os objetivos do problema tratado. Estas funções-objetivo serão discutidas na Seção 4.3, em que se discute a situação em que três objetivos são considerados, e na Seção 4.4, que apresenta a situação envolvendo cinco objetivos. O conjunto de restrições (4.6) garante que cada tarefa seja executada

em exatamente uma máquina em todos os estágios. O conjunto de restrições (4.7) garante que uma tarefa não se inicia em um determinado estágio, enquanto não concluir o estágio anterior. Já os conjuntos de restrições (4.8) e (4.9) impedem que duas tarefas sejam executadas na mesma máquina no mesmo instante. O conjunto de restrições (4.10) representa que todas as tarefas já podem ser executadas a partir do instante zero. Os conjuntos de restrições (4.11), (4.12) e (4.13) definem o domínio das variáveis de decisão. Finalmente, o conjunto (4.14) é necessário para que o valor do *makespan* seja o maior valor dentre os instantes de conclusão de todas as tarefas.

4.3 MOHFS com três objetivos

Esta seção apresenta o Problema *Flow Shop* Híbrido com três objetivos, o qual é definido nos mesmos termos do MOHFS, apresentado na Seção 4.1. Na situação aqui tratada, a minimização dos seguintes objetivos é considerada:

(i) *Makespan* ($C_{\max}$);

(ii) Soma ponderada dos atrasos ($\sum w_j T_j$);

(iii) Soma ponderada das antecipações ($\sum u_j E_j$).

O tempo de conclusão C_j de uma tarefa j é o instante em que a última operação dessa tarefa é concluída. O *makespan* $C_{\max}$, por sua vez, é o tempo de conclusão da última operação do sistema, ou seja, $C_{\max} = \max_{j \in N}\{C_j\}$. O atraso T_j de uma tarefa é definida como $\max\{(C_j - d_j), 0\}$. Além disso, a antecipação de uma tarefa é definida como $\max\{(d_j - C_j), 0\}$. Cada tarefa está associada a um custo por unidade de atraso w_j e a um custo por unidade de antecipação u_j. Tipicamente, $w_j > u_j$. As Figuras 4.1 e 4.2 mostram o caso de uma tarefa j, cuja data de entrega $d_j = 20$, $w_j = 5$, $u_j = 2$ e $C_j > d_j$, isto é, na Figura 4.1, a tarefa j está atrasada em 8 unidades de tempo e o custo por atraso para essa tarefa é $w_j T_j = 40$. Já na Figura 4.2 a tarefa j está antecipada em 10 unidades de tempo e o custo por antecipação para essa tarefa é $u_j E_j = 20$.

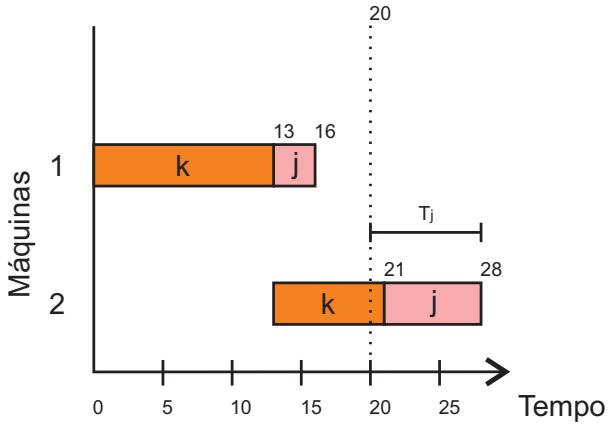


Figura 4.1: Exemplo de atraso.

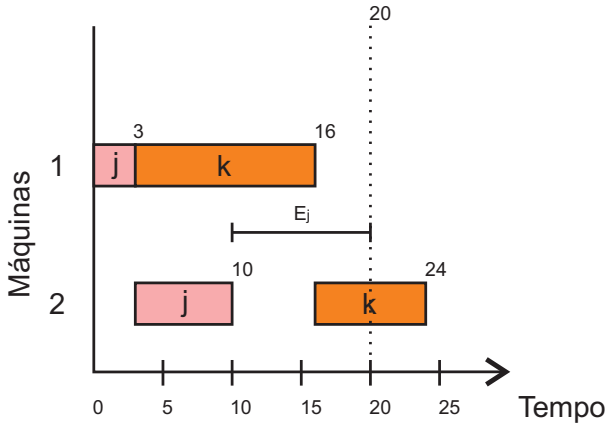


Figura 4.2: Exemplo de antecipação.

A Figura 4.3 mostra uma ilustração do ambiente de máquinas do problema. Cada tarefa, representada na Figura como uma elipse colorida, segue o mesmo fluxo de estágios e, em cada estágio, existe um conjunto de máquinas paralelas não-relacionadas. Algumas tarefas podem saltar estágios específicos.

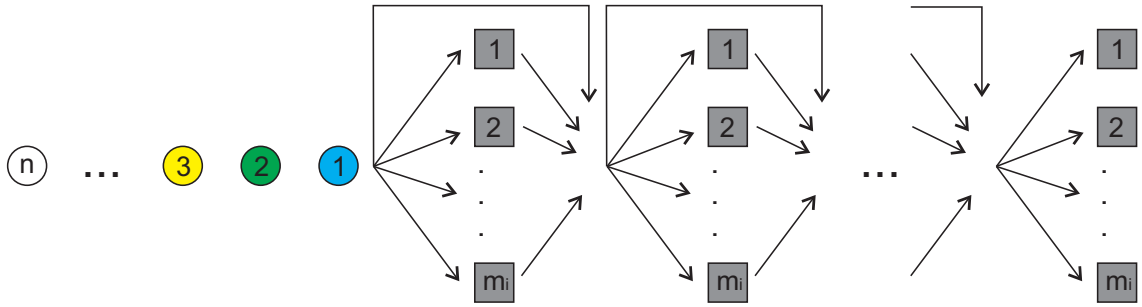


Figura 4.3: Ambiente *Flow Shop* Híbrido com Salto de Estágios.

4.4 MOHFS com cinco objetivos

Esta Seção introduz o Problema *Flow Shop* Híbrido com cinco objetivos, o qual é definido a seguir nos mesmos termos do MOHFS com três objetivos. Os dois novos objetivos considerados são:

(i) I_l: tempo de ociosidade da máquina l;

(ii) U_j: $U_j = 1$, se $T_j > 0$; e $U_j = 0$, caso contrário.

Portanto, nessa abordagem serão tratados cinco objetivos, a serem minimizados simultaneamente:

(i) *Makespan* ($C_{\max}$);

(ii) Soma ponderada dos atrasos ($\sum w_j T_j$);

(iii) Soma ponderada das antecipações ($\sum u_j E_j$);

(iv) Soma dos tempos de ociosidade ($\sum I_l$);

(v) Número de tarefas atrasadas ($\sum U_j$).

Os objetivos (i), (ii) e (iii) são os mesmos já expostos para o casso do MOHFS com três objetivos. No objetivo (iv), $I_l = (\max C_{jl}) - \sum p_{jl}$, isto é, a ociosidade de cada máquina é a quantidade de tempo durante o processamento no qual ela ficou sem executar nenhuma tarefa até a conclusão da última tarefa. Já no objetivo (v), $U_j = 1$ se $T_j > 0$ e $U_j = 0$, caso contrário, isto é, U_j é uma valor binário que indica se a tarefa j atrasou ou não.

A Figura 4.4 mostra um exemplo de ociosidade de máquinas. Nessa figura, a máquina 1 não possui ociosidade, enquanto a máquina 2 tem uma ociosidade de 12 unidades de tempo ($= C_{k2} - (p_{j2} + p_{k2}) = 25 - 8 - 5$).

A Figura 4.5 ilustra o número de tarefas atrasadas. Nesse caso, as datas de entrega das tarefas são $d_1 = 14$, $d_2 = 25$ e $d_3 = 28$ e os tempos de conclusão são $C_1 = 15$, $C_2 = 25$ e $C_3 = 28$, ou seja, duas tarefas estão atrasadas.

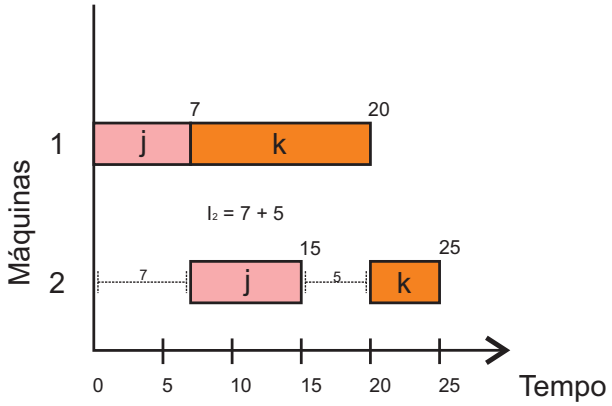


Figura 4.4: Exemplo de ociosidade.

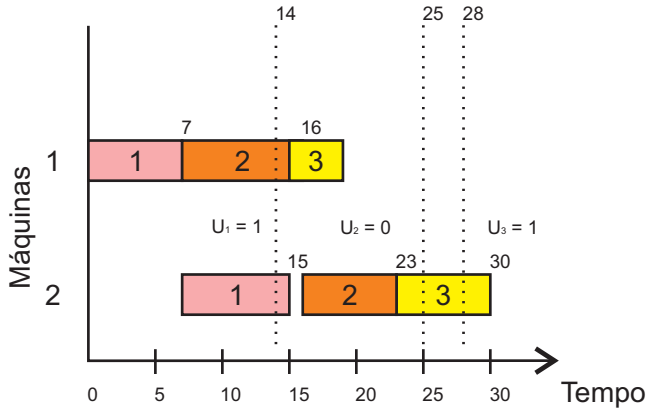


Figura 4.5: Exemplo de número de tarefas atrasadas.

4.5 Problemas *Flow Shop* multiobjetivo: revisão da literatura

Esta seção traz uma revisão bibliográfica a respeito de problemas *Flow Shop* multiobjetivo. Inicialmente, na Subseção 4.5.1, são apresentados trabalhos relacionados a problemas de *Flow Shop* multiobjetivo em geral, enquanto na Subseção 4.5.2 a abordagem é mais específica ao MOHFS.

4.5.1 Revisão bibliográfica: Problemas *Flow Shop* multiobjetivo

O primeiro trabalho sobre problemas *Flow Shop* Multiobjetivo (MO-FS, do inglês *Multi-objective Flow Shop*) surgiu no final da década de 1990, com Ishibuchi e Murata (1998). Em 2005, Arroyo e Armentano (2005) apresentam um algoritmo genético

aplicados ao MO-FS para a minimização de dois pares de objetivos: *makespan* e atraso máximo; *makespan* e atraso total.

A partir daí outros autores abordaram o MO-FS. Minella et al. (2008) trazem uma avaliação de algoritmos de otimização multiobjetivo para problemas *Flow Shop*.

Um novo algoritmo, chamado *Restarted Iterated Pareto Greedy* (RIPG), é apresentado em Minella et al. (2011); Ciavotta et al. (2013), para ser aplicado ao problema *Flow Shop* permutacional com *setup* dependente da sequência. O mesmo problema é abordado em Li e Li (2015) por meio de uma busca local baseada em decomposição, e em Li e Ma (2016), ele é resolvido por meio de Algoritmos Meméticos. Campos e Arroyo (2014) propuseram a aplicação do NSGA-II, com busca local, para resolver um problema *Flow Shop* com três estágios, tendo como objetivos a minimização do total *flowtime* e do atraso total.

Em Torkashvand et al. (2017), aborda-se um problema *Flow Shop* multiobjetivo no qual existem dois conjuntos de tarefas. Os objetivos considerados são a minimização do *makespan* e a minimização do atraso total. Os autores desenvolveram um novo método baseado em biogeografia e os resultados mostraram que o algoritmo proposto supera os outros algoritmos da literatura.

Em Wang e Tang (2017), um algoritmo multiobjetivo baseado em aprendizado de máquinas é proposto para um problema MO-FS permutacional. Três objetivos foram considerados: minimização do *makespan*, minimização do total *flowtime* e minimização do atraso total. Os resultados mostraram que o algoritmo melhora vários resultados na literatura. Xu et al. (2017) propuseram um algoritmo baseado em ILS multiobjetivo para resolver um problema MO-FS permutacional, a fim de minimizar o *makespan* e a soma ponderada do atraso. O algoritmo foi comparado com vários algoritmos evolutivos e superou os resultados de todos eles. Deng e Wang (2017) propuseram um Algoritmo Memético para resolver um problema MO-FS para minimizar o *makespan* e o atraso total. Gong et al. (2018) abordaram um problema MO-FS e apresentaram um algoritmo híbrido de colônia artificial de abelhas para resolver dois objetivos conflitantes: o *makespan* e a antecipação total. Em Lu et al. (2017), um problema MO-FS com *setup* dependente da sequência e tempo de transporte é investigado. O problema considera tanto o *makespan* quanto o consumo de total de *setup* e é resolvido por um algoritmo híbrido de busca *backtracking*.

4.5.2 Revisão bibliográfica: Problemas *Flow Shop* Híbrido multiobjetivo

Alguns trabalhos da literatura tratam de problemas *Flow Shop* Híbrido multiobjetivo (MOHFS, do inglês *Multi-objective Hybrid Flow Shop*) com dois objetivos.

Eles estão apresentados a seguir. Behnamian et al. (2009) implementaram uma metaheurística de três fases para resolver um problema MOHFS com máquinas idênticas em cada estágio e com tempos de *setup*. Em Dugardin et al. (2010) é apresentado um algoritmo, chamado L-NSGA, no qual o relacionamento de Dominância de Pareto é substituído pelo relacionamento de Dominância de Lorenz (Kostreva e Ogryczak, 1999; Kostreva et al., 2004). Os autores compararam os resultados obtidos com os encontrados pela enumeração completa e pelas adaptações dos algoritmos NSGA-II e SPEA2 (do inglês *Strength Pareto Evolutionary Algorithm 2*), este último proposto por Zitzler et al. (2002).

Asefi et al. (2014) resolveram um problema MOHFS aplicando uma abordagem que combina as características do NSGA-II e da metaheurística VNS (do inglês *Variable Neighborhood Search*) para a minimização do *makespan* e o atraso médio. Já em Ying et al. (2014), um algoritmo baseado em IPG (do inglês *Iterated Pareto Greedy*) é aplicado para resolver o problema MOHFS com o objetivo de minimizar o *makespan* e o atraso total. Os resultados mostraram grande eficiência do algoritmo proposto.

Wang et al. (2017) propuseram um algoritmo multiobjetivo para resolver um problema MOHFS. Os objetivos são minimizar o total *flowtime* e o número de tarefas atrasadas. Li et al. (2018) propuseram um algoritmo de otimização multiobjetivo, chamado *Energy-Aware* (EA-MOA), para resolver problemas MOHFS. Dois objetivos são considerados simultaneamente: a minimização do *makespan* e a minimização do consumo de *setup*. Mousavi et al. (2018) trataram um problema MOHFS biobjetivo, tendo como objetivos minimizar o *makespan* e a soma dos atrasos. O método de solução desenvolvido é baseado em algoritmos genéticos. Para validar o algoritmo proposto, um algoritmo de enumeração completa é usado para encontrar a fronteira de Pareto para problemas de pequeno porte. Os resultados mostraram que o algoritmo proposto é eficiente.

Pargar et al. (2018) estudaram o efeito de técnicas de aprendizagem de máquinas em um problema MOHFS com dois objetivos conflitantes: *makespan* e atraso total. Em de Siqueira et al. (2018), um algoritmo baseado no método MO-VNS (do inglês *Multi-objective Variable Neighborhood Search*) é proposto para resolver o problema MOHFS com dois objetivos de minimização: o *makespan* e a soma ponderada de atrasos.

Poucos artigos tratando o problema MOHFS com três objetivos são encontrados na literatura. Marichelvam e Geetha (2014) trataram um problema MOHFS com máquinas paralelas idênticas em cada estágio para minimizar o *makespan*, o total *flowtime* e o tempo de ociosidade das máquinas, usando um método chamado *Discrete Firefly Algorithm* (Yang, 2010). Xu et al. (2016) propuseram uma adaptação do algoritmo de Colônia de Abelhas Artificiais, baseado em um método de busca

de vizinhança adaptativa, para resolver o problema MOHFS com máquinas paralelas não-relacionadas. Os três objetivos a serem minimizados foram o *makespan*, a soma ponderada dos atrasos e antecipação e o tempo total de espera. Lei e Zheng (2017) trataram o problema do MOHFS com três objetivos o atraso total, o atraso máximo e o *makespan*. Os autores adotaram uma metodologia de priorização dos objetivos considerados. Assim, o problema resolvido é reduzido a um problema bi-objetivo. Eles propuseram um novo método de busca local, chamado *Neighborhood Structures and Global Exchange* (NSG). Os resultados mostraram que o algoritmo proposto é competitivo quando comparado a outros algoritmos da literatura como Busca Tabu Multiobjetivo (Wang e Liu, 2014) e NSGA-II. Li et al. (2017) apresentaram um algoritmo híbrido de Colônia de Abelhas Artificiais (Li et al., 2011) para resolver o problema de sequenciamento de tarefas multiobjetivo em um sistema de computação em nuvem, modelado como um problema MOHFS. Os autores abordaram a otimização de três objetivos simultaneamente: minimização do *makespan*, a maximização da carga de trabalho e a maximização da carga de trabalho total.

4.6 Exemplos de MOHFS multiobjetivo

Esta seção apresenta três exemplos do problema MOHFS. O primeiro exemplo, apresentado na Subseção 4.6.1, mostra as característica com três objetivos, enquanto o segundo exemplo, apresentado na Subseção 4.6.2, mostra uma situação na qual esses objetivos são conflitantes. Por fim, na Subseção 4.6.3 é apresentado um exemplo com cinco objetivos.

4.6.1 Exemplo do MOHFS com três objetivos

Nesta Seção apresentamos um exemplo que mostra todas as características do MOHFS com os três objetivos apresentados anteriormente.

Exemplo 5 *Considere uma instância com* 5 *tarefas e* 2 *estágios, com* 2 *máquinas em cada estágio. A Tabela 4.1 mostra quais máquinas l são elegíveis para cada tarefa j em cada estágio i. A partir desta tabela, pode-se verificar, por exemplo, que a tarefa* 1 *pode ser executada nas máquinas* 1 *e* 2 *no estágio* 1 *e na máquina* 4 *no estágio* 2. *Pode-se verificar, também, que as tarefas* $1, 2$ *e* 4 *visitam todos os estágios, enquanto a tarefa* 3 *pula o estágio* 1 *e a tarefa* 5 *pula o estágio* 2.

A Tabela 4.2 mostra o tempo de processamento (p_{ilj}) de cada tarefa j em cada máquina l de cada estágio i. Nessa tabela pode-se concluir que o tempo de processamento do tarefa 1 *na máquina* 2 *e no estágio* 1 *é de* 18 *unidades de tempo. O tempo de processamento de uma tarefa em uma máquina não elegível é nulo. A Tabela 4.2*

Tabela 4.1: Elegibilidade de Máquinas – Exemplo 5.

	i	1	2
j	1	{1,2}	{4}
	2	{1,2}	{3}
	3	-	{3,4}
	4	{2}	{3,4}
	5	{1, 2}	-

também mostra os valores das datas de entrega (d_j) e os custos unitários w_j e u_j de cada tarefa j. A partir desta tabela, a data de entrega da tarefa 4 é $d_4 = 51$, seu custo unitário de atraso é $w_4 = 4$ e seu custo unitário de antecipação é $u_4 = 3$.

Tabela 4.2: Tempo de processamento, data de entrega d_j, custo de atraso w_j e custo de antecipação u_j - Exemplo 5.

	i	1		2				
	l	1	2	3	4	d_j	w_j	u_j
j	1	10	18	0	21	35	4	2
	2	13	15	45	0	60	3	1
	3	0	0	15	22	48	1	1
	4	0	31	17	12	51	4	3
	5	17	29	0	0	30	5	5

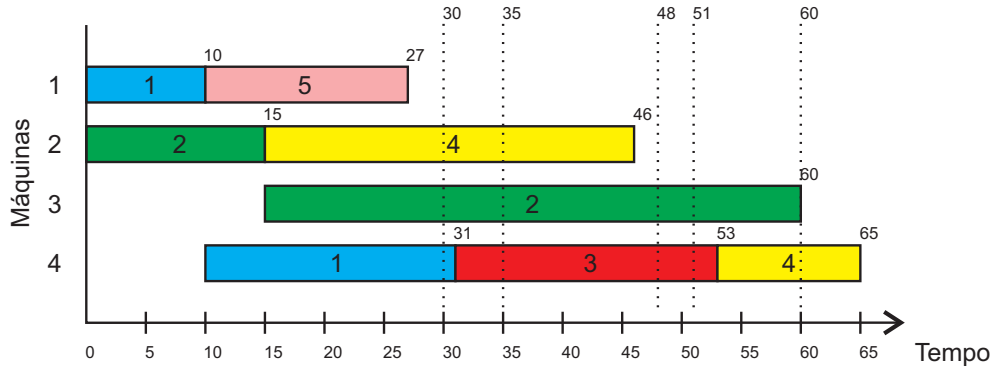


Figura 4.6: Diagrama de GANTT - Exemplo 5. $C_{\max} = 65$. $\sum w_j T_j = 61$. $\sum u_j E_j = 23$

A Figura 4.6 mostra uma possível sequência para este exemplo. Nota-se, nesse caso, que o makespan *para esta sequência é* 65 *unidades de tempo e a soma ponderada do atraso é igual a* 61 $(= w_3T_3 + w_4T_4 = 1 \times 5 + 4 \times 14)$. *A soma ponderada da antecipação é dada por* 23 $(= u_1E_1 + u_5E_5 = 2 \times 4 + 5 \times 3)$.

4.6.2 Uma situação em que os três objetivos são conflitantes

O exemplo anterior mostrou várias características do problema. Para frisar que os objetivos são conflitantes, outro exemplo será apresentado.

Exemplo 6 *Seja uma instância com* 4 *tarefas e* 3 *estágios, com uma máquina em cada estágio. Neste exemplo, para simplificar, um caso mais específico será considerado, no qual há apenas uma permutação de tarefas para todos os estágios. Além disso, todas as tarefas passam por todos os estágios e todas as máquinas são elegíveis para todas as tarefas.*

A Tabela 4.3 mostra o tempo de processamento (p_{ilj}) de cada tarefa j no estágio i. Esta tabela também mostra a data de entrega (d_j) e os custos w_j e u_j de cada tarefa j. Nesta Tabela, pode-se verificar que o tempo de processamento do trabalho 1 *é de* 6 *unidades de tempo nas três máquinas. A partir desta Tabela, também pode ser verificado que a data de entrega da tarefa* 2 *é* $d_2 = 45$ *e seus custos por atraso e antecipação são* $w_2 = 5$ *e* $u_2 = 3$*, respectivamente.*

Tabela 4.3: Tempos de Processamento, Data de entrega d_j, custo de atraso w_j e custo de antecipação u_j - Exemplo 6.

	i	1	2	3			
	l	1	2	3	d_j	w_j	u_j
j	1	6	6	6	32	3	1
	2	9	12	6	45	5	3
	3	15	18	3	40	10	2
	4	12	15	15	73	2	1

Inicialmente, a permutação $[3, 1, 2, 4]$ *foi considerada. O sequenciamento para esta solução está esquematizado na Figura 4.7. A solução apresentada produz um* makespan *de* $C_{\max} = 81$*, a soma ponderada do atraso é* $\sum w_j T_j = 115$ *(*$= w_1T_1 + w_2T_2 + w_4T_4 = 3\times13+5\times12+2\times8$*) e a soma ponderada da antecipação é* $\sum u_j E_j = 2$ *(*$= u_3E_3 = 2 \times 1$*).*

Com a configuração apresentada, a permutação que produz o menor makespan *é* $[1, 2, 4, 3]$*. A Figura 4.8 mostra o sequenciamento dessa solução. Para esse caso, temos:* $C_{max} = 63$*,* $\sum w_j T_j = 230$ *(*$= w_3T_3 = 10 \times 23$*) e* $\sum u_j E_j = 13$ *(*$= u_4E_4 = 1 \times 13$*). Assim, pode-se notar que, ao minimizar o* makespan*, tanto o atraso quanto a antecipação são comprometidos.*

A mesma permutação será agora considerada, mas a tarefa 4 *será deslocada para ser concluída exatamente na data prevista, isto é, não haverá assim nenhuma antecipação. Para que isso aconteça, a tarefa de* 3 *também será deslocada e, portanto, atrasará sua conclusão. Essa situação está ilustrada na Figura 4.9. Assim, essa alteração produziu um* makespan *de* $C_{\max} = 76$*, ou seja, um aumento de quase* 21%*; a*

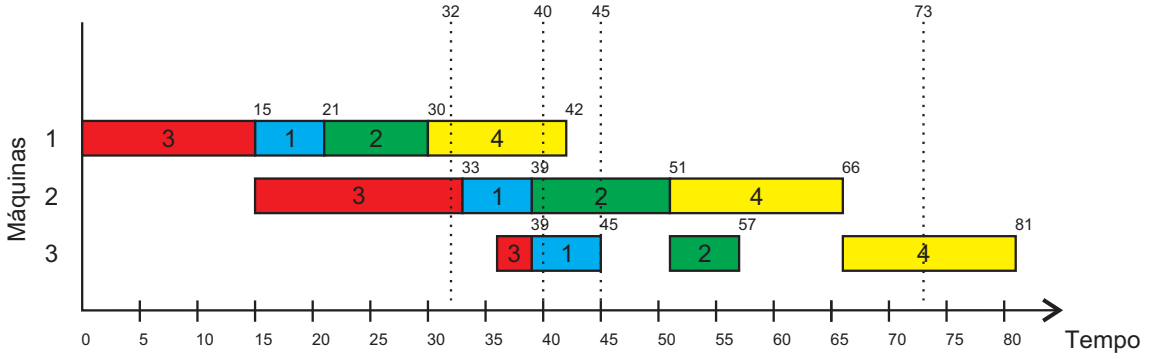


Figura 4.7: Diagrama de GANTT - Exemplo 6 - Caso 1. $C_{max} = 81$. $\sum w_j T_j = 115$. $\sum u_j E_j = 2$.

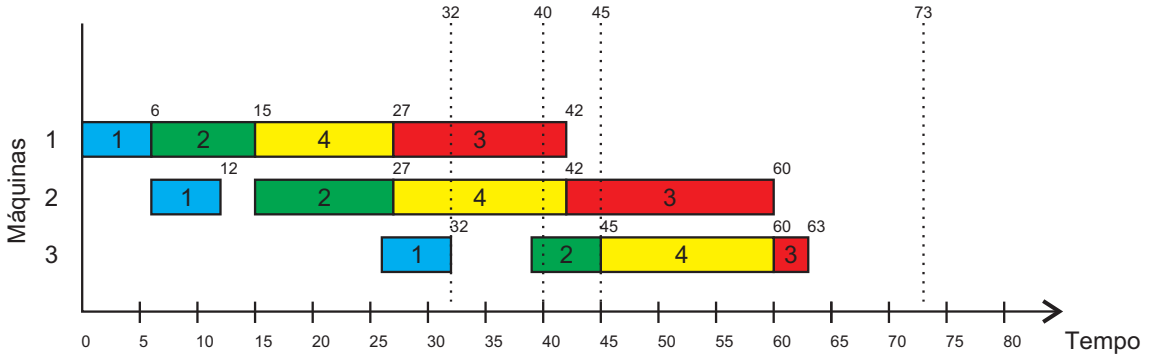


Figura 4.8: Diagrama de GANTT - Exemplo 6 - Caso 2. $C_{\max} = 63$. $\sum w_j T_j = 230$. $\sum u_j E_j = 13$.

soma ponderada de atrasos tornou-se $\sum w_j T_j = 360$ *(*$= w_3 T_3 = 10 \times 36$*), isto é, um aumento de mais de* 55%*; e a soma ponderada da antecipação é zero, já que nenhuma tarefa foi concluída antecipadamente.*

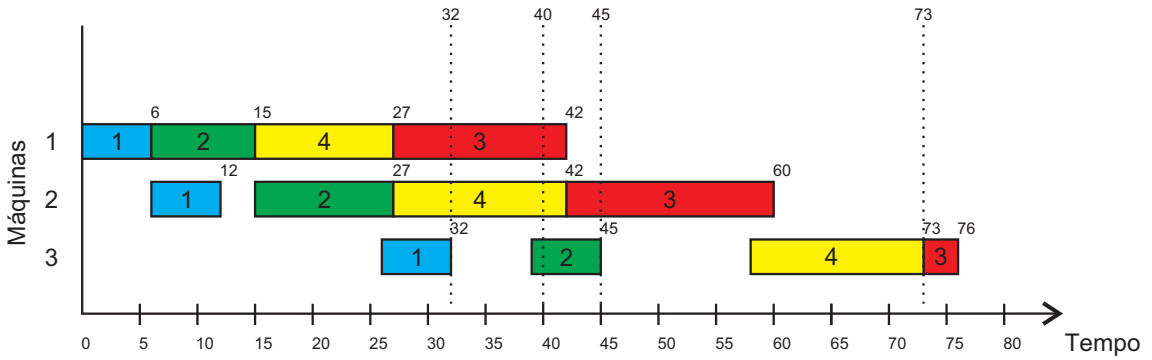


Figura 4.9: Diagrama de GANTT - Exemplo 6 - Caso 3. $C_{max} = 76$. $\sum w_j T_j = 360$. $\sum u_j E_j = 0$.

Situações como essas mostram que minimizar um critério pode comprometer seriamente os outros dois. Os problemas-teste considerados neste trabalho também

seguem essa característica, e estão exemplificados no Capítulo 6, Seção 6.1. Isso significa que os objetivos são fortemente conflitantes, por isso é necessário estudar e aplicar técnicas de otimização multiobjetivo para encontrar soluções de compromisso.

4.6.3 Exemplo do MOFHS com cinco objetivos

Nesta Seção apresentamos um exemplo do MOFHS com cinco objetivos.

Exemplo 7 *Seja um problema-teste com* 6 *tarefas e* 2 *estágios, com* 2 *máquinas em cada estágio. A Tabela 4.4 mostra quais máquinas l são elegíveis para cada tarefa j em cada estágio i. A partir desta tabela, pode-se verificar, por exemplo, que a tarefa* 1 *pode ser executada nas máquinas* 1 *e* 2 *no estágio* 1 *e na máquina* 4 *no estágio* 2. *Pode-se verificar, também, que todas as tarefas visitam todos os estágios.*

Tabela 4.4: Elegibilidade de Máquinas – Exemplo 7.

	i	1	2
j	1	{1,2}	{4}
	2	{1}	{3}
	3	{1, 2}	{3,4}
	4	{1, 2}	{3}
	5	{1}	{3,4}
	6	{2}	{4}

A Tabela 4.5 mostra o tempo de processamento (p_{ilj}) de cada tarefa j em cada máquina l de cada estágio i. Nessa Tabela pode-se concluir que o tempo de processamento do tarefa 2 *na máquina* 1 *e no estágio* 1 *é de* 23 *unidades de tempo. O tempo de processamento de uma tarefa em uma máquina não elegível é nulo. A Tabela 4.5 também mostra os valores das datas de entrega (d_j) e os custos w_j e u_j de cada tarefa j. A partir desta tabela, a data de entrega da tarefa* 3 *é $d_3 = 81$, o custo de atraso é $w_3 = 2$ e o custo de antecipação é $u_3 = 1$.*

Tabela 4.5: Tempos de Processamento, data de entrega d_j, custo de atraso w_j e custo de antecipação u_j - Exemplo 7.

	i	1		2				
	l	1	2	3	4	d_j	w_j	u_j
j	1	10	29	0	7	77	5	3
	2	23	0	10	0	93	4	2
	3	17	20	32	25	81	2	1
	4	28	30	14	0	67	7	7
	5	34	0	30	37	86	6	5
	6	0	34	0	27	71	5	4

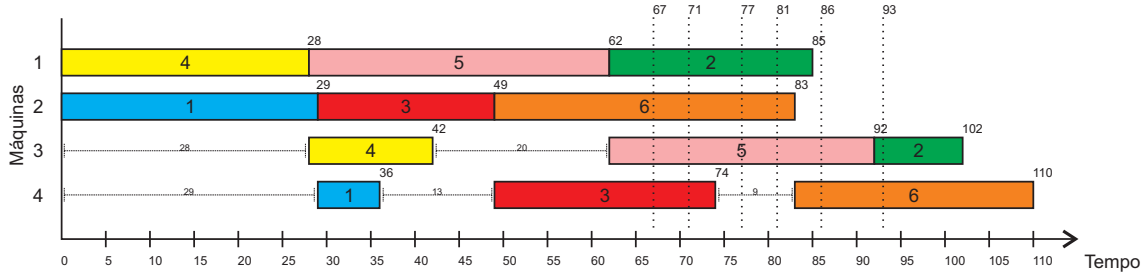


Figura 4.10: Diagrama de GANTT - Exemplo 7. $C_{max} = 110$. $\sum w_j T_j = 267$. $\sum u_j E_j = 305$. $I = 99$, $U = 3$

A Figura 4.10 mostra uma possível sequência para este exemplo. Nota-se, nesse caso, que o makespan *para esta sequência é* 110 *unidades de tempo e a soma ponderada do atraso é igual a* 267 $(= w_2T_2 + w_5T_5 + w_6T_6)$. *A soma ponderada da antecipação é dada por* 23 $(= u_1E_1 + u_3E_3 + u_4E_4)$. *O somatório do tempo de ociosidade é igual a* 99 $(= I_3 + I_4)$. *O número de tarefas atrasadas é* 3, *são elas* 2, 5 *e* 6.

Capítulo 5

Algoritmos Propostos para o problema MOHFS

Este capítulo apresenta os dois algoritmos propostos para aplicação ao problema definido no Capítulo 4. Na Seção 5.1 é mostrado como uma solução é representada nos algoritmos propostos. As Seções 5.2 e 5.3 apresentam os algoritmos propostos MO-GVNS e P-ILS, respectivamente. A Seção 5.4 apresenta o método de construção GRASP. A Seção 5.5 explica como as tarefas são sequenciadas nas máquinas, enquanto a Seção 5.6 apresenta os tipos de movimentos considerados para formação das estruturas de vizinhança. As Seções 5.7, 5.8 e 5.9 apresentam o método MO-VND, os algoritmos de busca local e o procedimento de perturbação das soluções, respectivamente. A Seção 5.10 define as denominações dos algoritmos para utilização no próximo capítulo.

5.1 Representação da Solução

Uma solução s do problema é representada por um vetor. Cada posição desse vetor indica a ordem em que cada tarefa é executada. A Figura 5.1 ilustra a representação de uma solução. Nessa figura, por exemplo, é mostrado que a primeira tarefa a ser executada é tarefa 1; então, as tarefas 2, 5, 3 e 4 são executadas, nessa ordem. Essa solução representa o sequenciamento mostrado na Figura 4.6 da Seção 4.6 do Capítulo 4. Essa é uma representação simples, porém eficiente, devido às características do problema *Flow Shop* Híbrido, em que a sequência executada no primeiro estágio interfere no processamento dos estágios seguintes.

Figura 5.1: Representação da Solução s.

5.2 MO-GVNS Proposto

Para resolver o MOHFS apresentado no Capítulo 4, é proposto um algoritmo baseado na metaheurística MO-GVNS (do inglês *Multi-Objective General Variable Neighborhood Search*) (Duarte et al., 2015). O MO-GVNS é uma versão multiobjetivo para o VNS (do inglês *Variable Neighborhood Search*) (Mladenović e Hansen, 1997; Hansen et al., 2008). A versão proposta na presente tese é denominada MO-GVNS Adaptada, considerando as modificações incluídas na versão original do algoritmo MO-GVNS.

O algoritmo proposto se difere da metaheurística original de Duarte et al. (2015) em, basicamente, quatro pontos. Primeiro, a geração de soluções iniciais do MO-GVNS Adaptado é feita por um método construtivo parcialmente guloso, enquanto em Duarte et al. (2015) as soluções iniciais são construídas aleatoriamente. Segundo, outra diferença é que a busca local do algoritmo original trabalha com um objetivo por cada vez, enquanto o MO-GVNS Adaptado trabalha com todos os objetivos simultaneamente. Terceiro, além disso, como outra diferença importante entre o algoritmo proposto e o método de Duarte et al. (2015), o MO-GVNS Adaptado inclui um segundo método de busca local, que é um procedimento de intensificação baseado na abordagem de dominância de Pareto introduzida em Arroyo et al. (2011). Quarto, além disso, um novo método de atribuição de tarefas também é proposto.

O pseudocódigo do MO-GVNS Adaptado é mostrado no Algoritmo 5. Esse algoritmo recebe como parâmetros o tamanho máximo (Max) do conjunto de soluções não-dominadas; o nível máximo ($levelMax$) de perturbação das soluções; e o critério de parada. Primeiramente, nas linhas 1–4, as soluções iniciais s são geradas pela fase de construção do método GRASP (explicada na Seção 5.4). As soluções geradas são inseridas no conjunto D caso sejam soluções não-dominadas. O procedimento para verificar e inserir uma solução não dominada no conjunto D é chamado *addSolution* e é mostrado no Algoritmo 6. Como mostra a Seção 5.1, uma solução é representada como uma permutação de tarefas. Quando uma nova solução é gerada, o método de alocação de tarefas é executado; esse método é detalhado na Seção 5.5. Depois disso, a variável *level* é inicializada na linha 5. A seguir, o algoritmo entra em seu laço principal, entre as linhas 6 e 18.

Em cada iteração do laço de repetição principal, uma das vizinhanças descritas na Seção 5.6 é selecionada (linha 7). Logo após, o procedimento de perturbação (linha 8), descrito na Seção 5.9, é aplicado. O resultado desse procedimento de perturbação é um novo conjunto D' de soluções não-dominadas. Esse conjunto passa pelo procedimento de busca local MO-VND Adaptado (linha 9), descrito na Seção 5.7. O resultado da busca local é um novo conjunto D'', possivelmente diferente de D'.

Algoritmo 5 MO-GVNS Adaptado

Entrada: *Max*, *levelMax*, *StopCriterion*

1: **para** $k \leftarrow 1$ to Max **faça**
2: $s \leftarrow$ geraSolucaoGRASP();
3: $D \leftarrow addSolution(D, s)$
4: **fim para**
5: $level \leftarrow 1$
6: **repita**
7: Selecione uma vizinhança aleatória $\mathcal{N}_i$
8: $D' \leftarrow Perturbation(D, level, \mathcal{N}_i)$
9: $D'' \leftarrow$ MO-VND Adaptado(D')
10: $flag \leftarrow falso$
11: $[flag, D] \leftarrow addSet(D, D'')$
12: **se** $flag = falso$ **faça**
13: $level \leftarrow level + 1$
14: **fim se**
15: **se** $level > levelMax$ **faça**
16: $level \leftarrow 1$
17: **fim se**
18: **até** Critério de parada ser satisfeito

Saída: D;

Algoritmo 6 *addSolution*

Entrada: D, s

1: **se** solução s é não-dominada em D **faça**
2: Insira s em D
3: Se uma solução x em D é dominada por s, então remova x de D.
4: **fim se**

Saída: D;

Algoritmo 7 *addSet*

Entrada: D, D'

1: **para** cada $s \in D'$ **faça**
2: $flag \leftarrow falso$
3: **se** solução s é não-dominada em D **faça**
4: Insira s em D
5: Se uma solução x em D é dominada por s, então remova x de D.
6: $flag \leftarrow verdadeiro$
7: **fim se**
8: **fim para**

Saída: $flag$, D;

Se houver uma melhora no conjunto D'', quando comparado ao conjunto D, ou seja, se houver pelo menos uma solução não-dominada nova no conjunto D'' que não esteja no conjunto antigo D', então o valor da variável *level* permanece o mesmo;

caso contrário, essa variável é incrementada em uma unidade, aumentando, assim, a intensidade da perturbação das soluções na próxima iteração. Se a variável *level* atingir seu valor máximo, ela é reiniciada e recebe seu menor valor (linha 15). O procedimento *addSet* (linha 11) compara os dois conjuntos, adiciona as soluções não-dominadas do novo conjunto e exclui as soluções dominadas do conjunto D. Esse procedimento retorna o valor "verdadeiro" se houver alguma nova solução no conjunto inicial e retorna o valor "falso", caso contrário.

O procedimento continua até que o critério de parada seja satisfeito e, em seguida, o conjunto de soluções não-dominadas D é retornado. As próximas seções explicam cada um dos módulos do algoritmo proposto.

5.3 P-ILS Proposto

Outro algoritmo proposto para resolver o MOHFS é o P-ILS (do inglês *Pareto Iterated Local Search*). Esse algoritmo foi proposto por Geiger (2007), e é uma versão multiobjetivo do algoritmo *Iterated Local Search* (ILS) de Lourenço et al. (2003).

A adaptação proposta para o P-ILS está esquematizada no Algoritmo 8. Inicialmente, é gerada uma solução s pela fase de construção do método GRASP, explicado na Subseção 5.4, e ela é adicionada ao conjunto de soluções não-dominadas D. Logo após, o algoritmo entra em dois laços de repetição, o primeiro (linhas 3 até 23) é executado até que o critério de parada seja atingido. No segundo laço de repetição (linhas 4 até 13), é selecionada aleatoriamente uma estrutura de vizinhança, e assim todos os vizinhos da solução s são avaliados e inseridos no conjunto D conforme a função *addSet*. Se houver melhora no conjunto D, uma nova solução vizinha de s é selecionada como nova solução corrente (linha 9) e todas as estruturas de vizinhanças são marcadas como não visitadas. O procedimento é repetido, selecionando uma vizinhança, gerando vizinhos e avaliando as novas soluções até que todas as estruturas de vizinhanças sejam visitadas.

Após isso, se ainda houver solução em D que ainda não foi escolhida, então é selecionada uma solução aleatoriamente, sem repetição. Caso não haja nenhuma solução não visitada em D, então é executado o procedimento de perturbação em uma solução aleatória, e essa nova solução perturbada é escolhida. O método é repetido, e quando o critério de parada for atingido, o conjunto D resultante é retornado.

Algoritmo 8 P-ILS Adaptado

Entrada: *levelMax*, *StopCriterion*

$s \leftarrow$ geraSolucaoGRASP();
$D \leftarrow \{s\}$
repita
  repita
    Selecione uma vizinhança não-visitada $\mathcal{N}_i$
    $D' \leftarrow \mathcal{N}_i(s)$
    $[flag, D] \leftarrow addSet(D, D')$
    se $flag = true$ **faça**
      Selecione uma solução $s' \in D'$
      $s \leftarrow s'$
      Redefina todas as vizinhanças como não-visitadas
    fim se
  até todas as vizinhanças sejam visitadas
  se existe uma solução não-visitada em D **faça**
    Selecione uma solução não-visitada $s \in D$
  senão
    Selecione uma solução $s' \in D$
    Selecione uma vizinhança aleatória $\mathcal{N}_i$
    $level \leftarrow$ Gere um número aleatório entre 1 e $levelMax$
    $D' \leftarrow Perturbacao(D, level, \mathcal{N}_i)$
    Selecione uma solução $s \in D'$
  fim se
até Critério de parada seja satisfeito

Saída: D;

5.4 Construção GRASP multiobjetivo

Para gerar soluções iniciais, um algoritmo baseado na fase de construção da metaheurística *Greedy Randomized Adaptive Search Procedures* – GRASP (Feo e Resende, 1995) foi adaptado.

O procedimento de construção de soluções iniciais funciona como segue. Para cada posição do vetor solução s, um dos três critérios a seguir é escolhido aleatoriamente: (i) menor tempo de processamento; (ii) menor data de entrega; e (iii) maior data de entrega. Cada um desses critérios privilegia uma das funções objetivo abordadas: (i) *makespan*, (ii) atraso e (iii) antecipação, respectivamente. De acordo com o critério escolhido, as tarefas com a melhores avaliações são inseridas em uma Lista de Candidatos Restritos (LCR), dada por:

$$LCR = \{j \in J \,|\, g^m(j) \leq g^m_{\min} + \alpha\,(g^m_{\max} - g^m_{\min})\}$$

em que g^m é a função de avaliação do critério escolhido, J é o conjunto com todas

as tarefas que ainda não foram sequenciadas, e $g^m_{\max}$ e $g^m_{\min}$ são, respectivamente, os valores máximo e mínimo da função de avaliação em relação ao critério escolhido aplicado à tarefa $j \in J$. Depois disso, uma tarefa é escolhida aleatoriamente na lista LCR e inserida no vetor s. O parâmetro α controla a aleatoriedade do procedimento, de forma que quanto maior o valor de α, mais aleatório é o método apresentado. Essas etapas são repetidas até que todas as tarefas sejam sequenciadas.

5.5 Alocação das tarefas

No Algoritmo 5, sempre que uma nova solução é gerada, o procedimento que atribui as tarefas às respectivas máquinas é executado e calcula os valores dos objetivos. Esse procedimento de atribuição de tarefas nas máquinas dos respectivos estágios é mostrado no Algoritmo 9 e funciona da seguinte maneira.

O método recebe uma solução s como parâmetro de entrada. Na linha 1, essa solução é atualizada com o procedimento *updateSolution*. Dada uma sequência de tarefas $s = (\ldots, a, b, \ldots)$, o procedimento *updateSolution* altera as posições de duas tarefas subsequentes para que a primeira da sequência visite mais estágios futuros do que a segunda. Assim, após essa aplicação, esse procedimento retorna outra sequência $s' = (\ldots, b, a, \ldots)$ satisfazendo a essa condição. Para evitar o crescimento do valor do atraso e mudanças muito drásticas, essa troca não é realizada sempre, isto é, se o tempo de conclusão da primeira tarefa no último estágio visitado for maior do que sua data de entrega, então a troca não é feita. Além disso, cada tarefa participa dessa alteração de posições apenas uma vez em cada estágio.

Algoritmo 9 Alocação de Tarefas

Entrada: s

1: $s' \leftarrow updateSolution(s)$
2: **para** cada estágio i **faça**
3: **para** $k \leftarrow 1$ to n **faça**
4: $j \leftarrow s'_k$
5: **se** $i \in F_j$ **faça**
6: Selecione uma máquina l do estágio i na qual a tarefa j possa concluir mais cedo, tal que $l \in E_{ij}$.
7: Sequencie a tarefa j na próxima posição livre da máquina l .
8: **fim se**
9: **fim para**
10: $s' \leftarrow updateSolution(s')$
11: **fim para**
12: Calcule $C_{\max}$, $\sum w_j T_j$ and $\sum u_j E_j$.

Saída: $C_{\max}$, $\sum w_j T_j$, $\sum u_j E_j$;

Após a atualização do vetor inicial, para cada estágio i (linhas 2 - 11) e em cada posição k do vetor s' (linhas 3 - 9), a tarefa colocada na posição k, se visitar o estágio i, é executada pela máquina que pode concluí-la o mais rápido possível dentre as máquinas elegíveis. Depois de executar todas as tarefas, o vetor s' é atualizado pelo método *updateSolution* (linhas 10). O procedimento continua até que todos os estágios sejam considerados. Finalmente, os valores dos objetivos são calculados e retornados.

As Figuras 5.2(a) até 5.2(j) ilustram a alocação das tarefas para a solução apresentada na Seção 5.1.

Na Figura 5.2(a), a tarefa 1 é a primeira a ser sequenciada, pois ela está na primeira posição do vetor. Nesse caso, a máquina 1 foi escolhida, pois essa máquina pode encerrar a tarefa no instante 10, enquanto a máquina 2 conclui a tarefa 1 no instante 18. A próxima tarefa a ser sequenciada no estágio 1 é a tarefa 2, isso está representado na Figura 5.2(b). Como a máquina 1 está ocupada até o instante 10 e liberaria a tarefa 2 no instante 23, então essa tarefa é alocada à máquina 2, sendo concluída no instante 15. Do mesmo modo, a tarefa 5 é sequenciada conforme a Figura 5.2(c) e a tarefa 3 salta o primeiro estágio, sendo então desconsiderada (Figura 5.2(d)). A última tarefa a ser alocada é a tarefa 4 (Figura 5.2(e)).

Para alocar as tarefas do segundo estágio deve se levar em conta os instantes de liberação no estágio anterior. Assim, a tarefa 1 que está liberada no instante 10 e só pode ser executada na máquina 4 no estágio 2, é alocada a essa máquina, conforme Figura 5.2(f). As outras tarefas são alocadas em ordem, com exceção da tarefa 5 que não passa por esse estágio.

5.6 Estruturas de Vizinhança

Quatro tipos de movimentos são usados para explorar o espaço de soluções do problema, cada um definindo uma estrutura de vizinhança, nessa ordem:

(a) Inserção: consiste em mudar a posição de uma tarefa. A Figura 5.3(a) ilustra a inserção da tarefa 2 em uma nova posição na sequência (nesse caso, após a tarefa 3);

(b) Troca: consiste em trocar a posição de duas tarefas entre si. A Figura 5.3(b) ilustra a troca entre as tarefas 1 e 3, ou seja, a tarefa 1 ocupa a posição da tarefa 3 e, por sua vez, a tarefa 3 ocupa a posição de 1;

(c) Inversão: consiste em inverter a ordem de tarefas em um determinado intervalo. A Figura 5.3(c) mostra a inversão da sequência de execução entre as tarefas 2 e

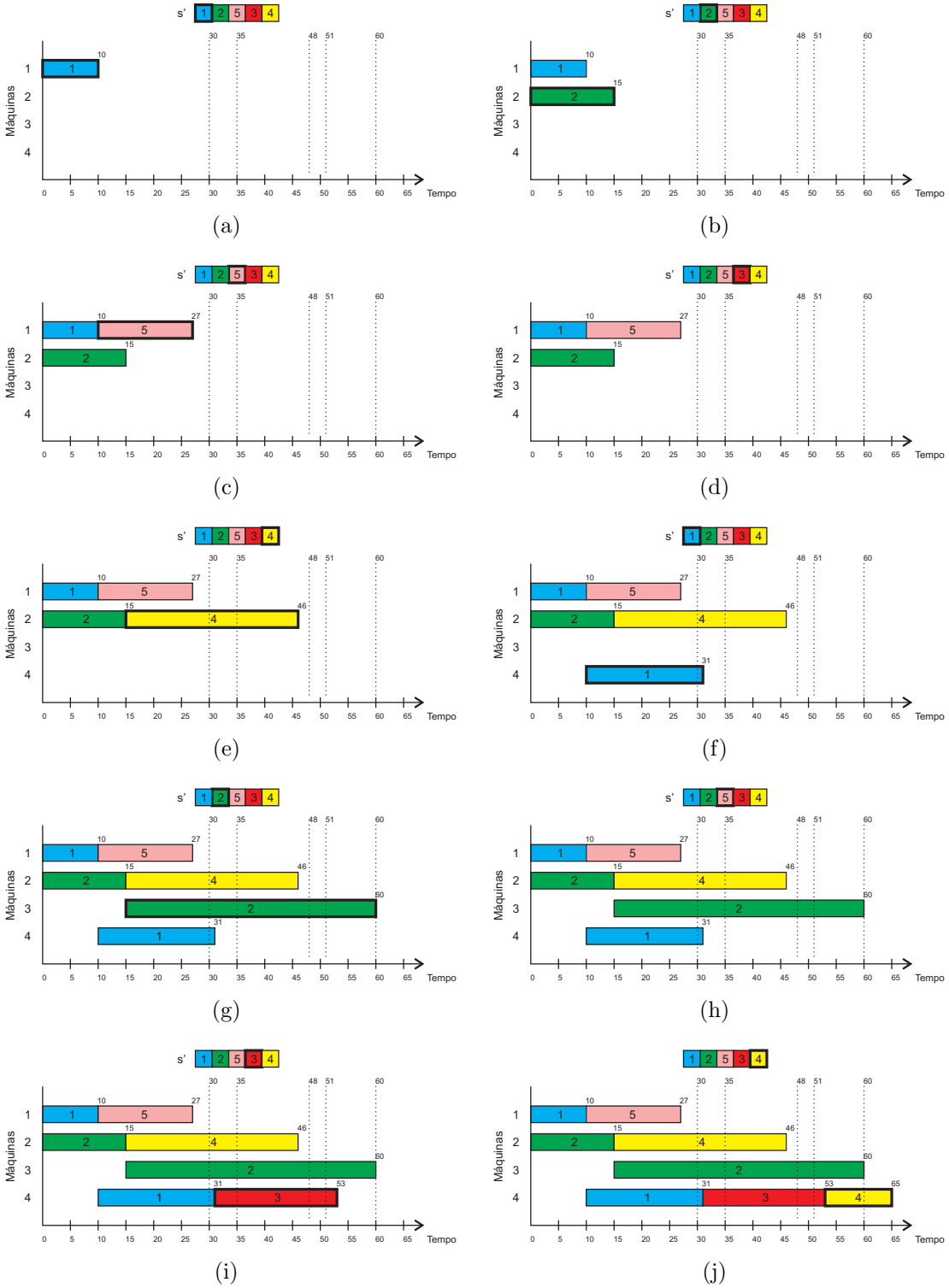
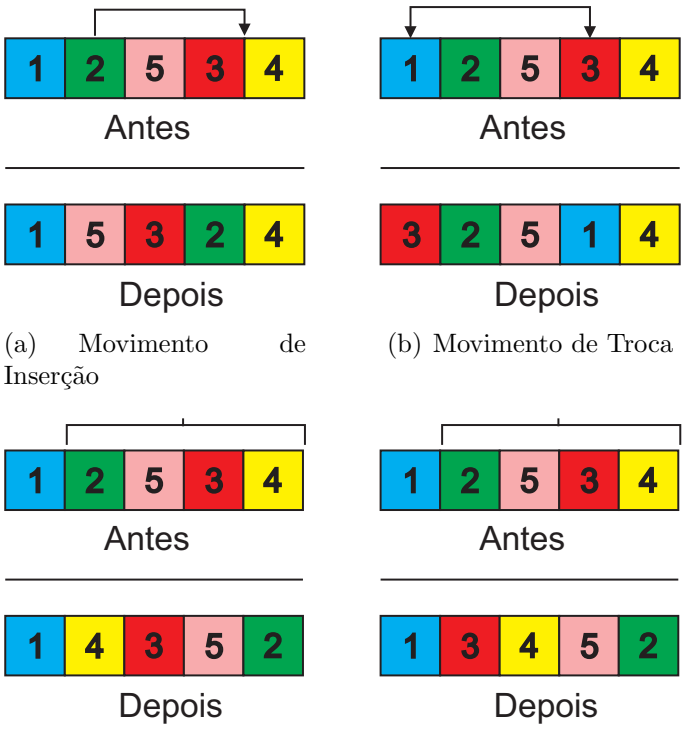


Figura 5.2: Procedimento de Alocação de Tarefas.

4, incluindo-as. Neste caso, a sequência nesse intervalo foi 2, 5, 3 e 4 e, após a mudança, tornou-se 4, 3, 5 e 2.

(d) *Scramble*: consiste em misturar aleatoriamente a ordem de todas as tarefas dentro



(a) Movimento de Inserção

(b) Movimento de Troca

(c) Movimento de Inversão

(d) Movimento *Scramble*

Figura 5.3: Tipos de movimento.

de um determinado intervalo. Esse tipo de movimento é usado apenas na fase de perturbação. Neste movimento, a princípio, duas tarefas diferentes são escolhidas aleatoriamente; então, as tarefas que estão no intervalo entre elas (incluindo-as) são embaralhadas. Portanto, o tamanho do intervalo pode variar de 2 a n, sendo n o número de tarefas. A Figura 5.3(d) mostra o embaralhamento da sequência no intervalo entre as tarefas 2 e 4.

5.7 MO-VND Adaptado

O Algoritmo 10 mostra o método MO-VND adaptado, usado como busca local para a metaheurística MO-GVNS Adaptada apresentada no Algoritmo 5. O MO-VND Adaptado proposto recebe um parâmetro inicial, o conjunto de soluções não-dominadas D, e realiza uma busca sistêmica na vizinhança das soluções usando dois métodos de busca local, a saber: *LocalSearch-r* e *Arroyo-LocalSearch*. O método *LocalSearch-r* realiza a busca com os três primeiros tipos de movimentos apresentados na Subseção 5.6: (a) inserção, (b) troca e (c) inversão. O método *Arroyo-LocalSearch* é proposto em Arroyo et al. (2011). Ambos os métodos de busca local são descritos na Seção 5.8.

Algoritmo 10 MO-VND adaptado

Entrada: D

1: $r \leftarrow 1$
2: **repita**
3: **se** $r < 3$ **faça**
4: $D' \leftarrow LocalSearch\text{-}r(D)$
5: **senão**
6: $D' \leftarrow Arroyo\text{-}LocalSearch(D)$
7: **fim se**
8: $[flag, D] \leftarrow addSet(D, D')$
9: **se** $flag = false$ **faça**
10: $r \leftarrow r + 1$
11: **senão**
12: $r \leftarrow 1$
13: **fim se**
14: **até** $r = 4$

Saída: D;

Alguns detalhes a seguir mostram como o Algoritmo 10 funciona. Inicialmente, na linha 1, a primeira vizinhança é definida pela junção dos movimentos (a)inserção e (b) troca ($r = 1$). Dubois-Lacoste et al. (2009) mostraram que a união desses movimentos é mais eficiente na busca local multiobjetivo do que quando usados separadamente para resolver problemas *Flow Shop*. Depois disso, o processo de busca local é repetido até que todas as vizinhanças consideradas tenham sido exploradas. Nesse processo, se $r = 1$, a busca local é aplicada considerando os movimentos de inserção e troca; e se $r = 2$, o movimento de inversão é aplicado na busca local. O Algoritmo 11 apresenta como esses operadores de busca local são aplicados. Por outro lado, se $r = 3$, então o método *Arroyo-LocalSearch* é executado. O Algoritmo 12 mostra o pseudocódigo desse método. Na linha 8, tal como acontece no Algoritmo 5, o procedimento *addSet* verifica se houve melhora no conjunto D. Se não houver adição de novas soluções não-dominadas, o valor de r é incrementado e o algoritmo explora uma nova vizinhança; caso contrário, o valor de r retornará ao seu valor mínimo, ou seja, $r \leftarrow 1$.

5.8 Algoritmos de Busca Local

Esta seção apresenta os dois métodos de busca local usados para o refinamento das soluções. O primeiro, dado pelo Algoritmo 11, e nomeado aqui *LocalSearch-r*, é mostrado no Algoritmo 10 e explicado na Seção 5.7. Esse método aplica apenas um dos seguintes movimentos durante todo o processo: (a) inserir, (b) trocar e (c) inversão. A ideia desse método é baseada na busca local proposta por Paquete et al. (2004). O valor da variável r define o movimento a ser aplicado. Isto é, se $r = 1$, o

Algoritmo 11 *LocalSearch-r*

Entrada: D

1: **repita**
2: Selecione uma solução s não-visitada em D
3: $S \leftarrow \{s\}$
4: **repita**
5: Selecione uma solução s não-visitada em Y
6: $Y' \leftarrow$ Toda vizinhança-r de s
7: $[flag, Y] \leftarrow addSet(Y, Y')$
8: **até** $flag = falso$
9: $[flag, D] \leftarrow addSet(D, Y)$
10: **até** Soluções não-visitadas em $D = \emptyset$

Saída: D;

refinamento é executado pelos movimentos de inserção e troca, em conjunto; se $r = 2$, o refinamento é realizado pelo movimento de inversão. O segundo método de busca local, aqui chamado *LocalSearch-Arroyo* e apresentado pelo Algoritmo 12, consiste em aplicar um procedimento de intensificação proposto por Arroyo et al. (2011), que é, por sua vez, baseado no método *Iterated Greedy Search* – IGS (Ruiz e Stützle, 2007). Esses dois métodos são descritos a seguir.

O Algoritmo 11 mostra o pseudocódigo do método *LocalSearch-r*. Esse método começa selecionando uma solução s do conjunto D de soluções não-dominadas (linha 2). Essa solução é inserida em um novo conjunto Y (linha 3). Depois disso, o algoritmo entra em um laço de repetição (linhas 4 - 8). Em cada iteração desse laço, uma solução não-visitada do conjunto Y é selecionada e todos os vizinhos de s são gerados com o movimento r, os quais formarão um novo conjunto Y' e, se houver uma melhora nesse conjunto quando comparado ao conjunto Y, o procedimento é repetido; caso contrário, a variável *flag* assume o valor *falso* e o laço é interrompido. Ao final das iterações, o conjunto Y é adicionado ao conjunto D com o procedimento *addSet*. Todas essas etapas são repetidas até que todas as soluções pertencentes a D sejam visitadas e, portanto, o conjunto D seja retornado.

O Algoritmo 12 descreve o pseudocódigo do método *Arroyo-LocalSearch*. Esse método começa selecionando uma solução s do conjunto D de soluções não-dominadas (linha 1). Então p tarefas são removidas dessa solução e inseridas em um conjunto X (linha 2). A sequência parcial v gerada pela remoção de p tarefas da solução s é inserida em um novo conjunto Y (linha 3). Para cada tarefa removida da solução s inicialmente selecionada (e agora pertencente ao conjunto X), um conjunto Y' é criado e, para cada solução parcial v pertencente ao conjunto Y, a tarefa $j \in X$ é inserida em todas as posições de v, gerando assim um novo conjunto Y'' com $(n-p+j)$ sequências resultantes de cada inserção. O conjunto Y'' é adicionado ao conjunto Y'

Algoritmo 12 *Arroyo-LocalSearch*

Entrada: D

1: Selecione uma solução s em D
2: $v \leftarrow$ Remova p tarefas aleatórias de s e insere essas tarefas em X , isso é, v é uma sequência parcial com $n - p$ tarefas
3: $Y \leftarrow \{v\}$
4: **para** $j \leftarrow 1$ to p **faça**
5: $\quad Y' \leftarrow \emptyset$
6: $\quad$ **para** cada sequência parcial $v \in Y$ **faça**
7: $\quad\quad$ Insira a tarefa $j \in X$ em todas as posições de v, gerando o conjunto Y'' de $(n - p + j)$ sequências
8: $\quad\quad$ $[flag, Y'] \leftarrow addSet(Y', Y'')$
9: $\quad$ **fim para**
10: $\quad Y \leftarrow Y'$
11: **fim para**

Saída: Y;

Algoritmo 13 Perturbação

Entrada: D, $level$, $\mathcal{N}_i$

1: $D' \leftarrow \emptyset$
2: **para** cada solução $s \in D$ **faça**
3: $\quad$ **para** $q \leftarrow 1$ to $level$ **faça**
4: $\quad\quad$ Aplique o movimento $\mathcal{N}_i$ em s aleatoriamente
5: $\quad$ **fim para**
6: $\quad D' \leftarrow addSolution(D', s)$
7: **fim para**

Saída: D';

com o procedimento *addSet* (linha 8). No final do segundo laço, o conjunto Y recebe o conjunto Y' (linha 10). O algoritmo retorna o conjunto Y ao final do procedimento.

5.9 Procedimento de Perturbação

O procedimento de perturbação, incluído na linha 8 do Algoritmo 5 e na linha 20 do Algoritmo 5.3, é mostrado no Algoritmo 13. Esse procedimento consiste em aplicar o movimento escolhido em todas as soluções do conjunto D em *level* vezes. Portanto, se *level* assumir o valor 2, o movimento escolhido será aplicado duas vezes. Todas as tarefas envolvidas nesse procedimento são escolhidas aleatoriamente.

5.10 Denominações de Algoritmos

Nos testes finais, seis algoritmos foram avaliados. O primeiro é o P-ILS adaptado, mostrado no Algoritmo 8, denominado P-ILS. O segundo é o MO-GVNS Adaptado, mostrado no Algoritmo 5, doravante denominado MO-GVNS1. O terceiro é o mesmo Algoritmo 5 sem a fase de intensificação do Algoritmo 12, denominado MO-GVNS2. Os outros são o algoritmo MO-GVNS, proposto por Duarte et al. (2015), denominado MO-GVNS3; o algoritmo MO-RVNS (do inglês *Multi-Objective Reduced Variable Neighborhood Search*), que se difere do MO-GVNS1 pelo fato de não possuir busca local, ou seja, o método MO-VND Adaptado (linha 9 do Algoritmo 5) não é executado; e o NSGA-III de Deb e Jain (2014) e Jain e Deb (2014).

O NSGA-III implementado é aquele proposto em Yuan et al. (2015) e descrito pelo Algoritmo 2. O método de geração de soluções iniciais aplicado ao NSGA-III é o mesmo do algoritmo de construção usado no MO-GVNS1.

Capítulo 6

Experimentos computacionais

Este capítulo apresenta os resultados obtidos pela execução dos algoritmos propostos. Na Seção 6.1 as configurações iniciais são apresentadas. A Seção 6.2, por sua vez, mostra como foram realizados os testes para calibração dos parâmetros. A Seções 6.3 trazem os resultados para o problema com três objetivos, a análise estatística e as discussões. Por fim, a Seção 6.4 mostra o resultado da aplicação dos algoritmos ao problema com cinco objetivos.

6.1 Configurações

Os algoritmos apresentados no Capítulo 5 foram implementados em C++, utilizando-se o ambiente IDE Netbeans 6. Os testes foram executados em um computador Intel Core i7, 2.00 GHz, com 16 GB de memória RAM, sob sistema operacional Linux Ubuntu 64 bits.

O conjunto de instâncias utilizadas nos experimentos foram adaptadas de Urlings (2010), uma vez que os instâncias desse autor não continham penalizações por antecipação. Assim, foram incluídas os custos unitários u_j por antecipação de uma tarefa j, sendo u_j um número inteiro aleatório no intervalo $[0, w_j]$. Esse conjunto é composto por 432 instâncias de pequeno porte e outras 144 instâncias maiores. Foram geradas outras 432 instâncias com $n > 50$, pois o conjunto original não continha instâncias dessa magnitude. Assim, os instâncias estão subdivididos pelo número de tarefas (n), máquinas (m) e máquinas por estágio (m_i). As instâncias menores possuem as seguintes configurações: $n \in \{5, 7, 9, 11, 13, 15\}$, $m \in \{2, 3\}$ e $m_i = 3$. As instâncias maiores têm $n \in \{50, 100, 150, 200\}$ e $m \in \{4, 8\}$ and $m_i \in \{2, 4\}$. Assim, tem-se um total de 1008 instâncias divididas em 28 grupos de 36 instância cada.

As Figuras 6.1, 6.2 e 6.3 mostram os sequenciamentos ótimos para cada um dos objetivos em uma instância do conjunto, com $n = 5$, $m = 2$ e $m_i = 3$. Nessa instância,

todas as tarefas passam por todos estágios e todas as máquinas são elegíveis para todas as tarefas e as datas de entrega das tarefas são $d_1 = 77$, $d_2 = 63$, $d_3 = 81$, $d_4 = 67$ e $d_5 = 66$. A Figura 6.1 mostra o sequenciamento no qual o *makespan* é mínimo, nessa situação $C_{\max} = 90$, $\sum w_j T_j = 310$ e $\sum u_j E_j = 75$. A Figura 6.2 mostra o diagrama de Gantt, no qual a soma ponderada do atraso é minimizada. Pode-se notar que nesse sequenciamento o valor de $C_{\max}$ e $\sum u_j E_j$ são maiores. Já a Figura 6.2 mostra o sequenciamento no qual não há antecipação; logo, $\sum u_j E_j = 0$. Nessa última situação os valores de $C_{\max}$ e $\sum w_j T_j$ são comprometidos.

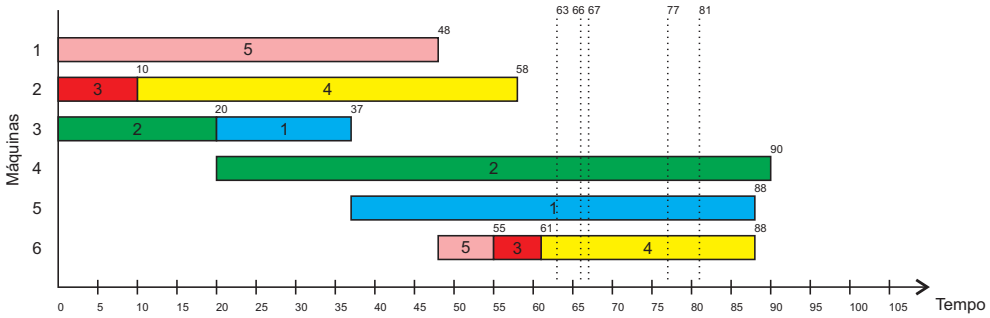


Figura 6.1: Diagrama de GANTT - Solução ótima para o *Makespan*. $C_{\max} = 90$. $\sum w_j T_j = 310$. $\sum u_j E_j = 75$

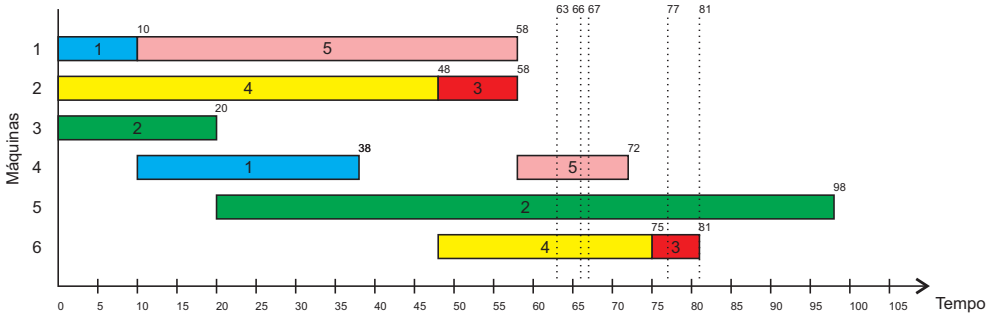


Figura 6.2: Diagrama de GANTT - Solução ótima para o atraso. $C_{\max} = 98$. $\sum w_j T_j = 232$. $\sum u_j E_j = 117$

Para o cálculo dessas métricas, foi utilizado o pacote *Evolutionary Multiobjective Algoritms Optimization Algorithms* – EMOA (Mersmann, 2012), disponível no *software* de estatística *R*. Além disso, com relação a este cálculo das métricas, os valores da função objetivo ϕ_ι, obtidos para a instância λ e a solução χ_κ e contidos no conjunto de soluções não-dominadas, foram normalizados de acordo com a Equação (6.1):

$$norm_{\lambda,\iota} = \frac{\phi_\iota^\lambda(\chi_\kappa) - \min_{\lambda\iota}}{\max_{i\iota} - \min_{\lambda\iota}} \tag{6.1}$$

em que $\min_{\lambda\iota}$ e $\max_{\lambda\iota}$ são os menores e maiores valores encontrados para o objetivo ι na instância λ, respectivamente, em todos os experimentos realizados neste

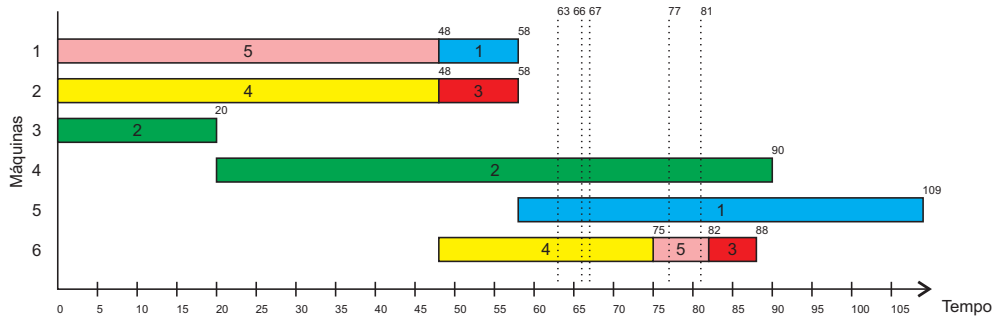


Figura 6.3: Diagrama de GANTT - Solução ótima para a antecipação. $C_{\max} = 109$. $\sum w_j T_j = 434$. $\sum u_j E_j = 0$

trabalho. O resultado final de cada métrica é a média de todas as execuções em cada instância. Essas médias foram agrupadas de acordo com as características das instâncias. Portanto, existem 36 valores médios para cada versão do algoritmo em cada grupo.

6.2 Testes de calibração

Inicialmente, os parâmetros *Max* e *levelMax* dos métodos MO-GVNS1 e P-ILS e a variável p do método *Arroyo-LocalSearch* foram calibrados usando-se o software IRACE (López-Ibáñez et al., 2016; López-Ibáñez et al., 2011). Esse pacote só funciona com um valor de avaliação por vez. Portanto, como o algoritmo proposto é multiobjetivo e retorna um conjunto de soluções não-dominadas, utilizamos a métrica *Hypervolume* explicada na Seção 2.4 do Capítulo 2, para avaliar a qualidade dos conjuntos. Os valores usados como base para esse teste foram $Max \in \{50, 100, 150, 200\}$, *levelMax* no intervalo de números inteiros entre 3 e 9 e p no intervalo de números inteiros entre 4 e 8. O valor de $p > 4$ foi desconsiderado para instâncias com $n < 11$.

Para esses testes de calibração foram geradas outras 252 instâncias de treinamento. Esses testes não lidaram com o número total de instâncias para evitar a execução exaustiva dos algoritmos. Para testar cada grupo de instâncias, 9 delas foram escolhidas aleatoriamente, totalizando 252 instâncias, os quais foram modificados para formar um novo conjunto de instâncias de treinamento. Cada uma das instâncias modificadas foi testada 30 vezes em todos algoritmos. A partir dos resultados de todas as execuções por instância, foram gerados conjuntos de soluções não-dominadas normalizadas, as quais foram avaliadas pela métrica Hypervolume. Assim, para cada uma das instâncias do conjunto de treinamento, temos um valor de medida de desempenho adequado para calibração via Pacote IRACE.

Os valores resultantes apresentados pelo software IRACE estão apresentados na Tabela 6.1. Nessa Tabela cada linha representa um Algoritmo e cada coluna repre-

senta os valores de *Max*, *levelMax* e *p*, respectivamente. Os algoritmos 5 e 8 foram denominados MO-GVNS1 e P-ILS, respectivamente. Os valores dos parâmetros foram utilizados para testar os algoritmos na segunda bateria de testes apresentados na Seção 6.3.

Tabela 6.1: Pârametros dos Algoritmos

Algoritmo	*Max*	*levelMax*	*p*
P-ILS	-	6	-
MO-GVNS1	150	3	8

Os parâmetros dos algoritmos MO-GVNS2, MO-GVNS3, MO-RVNS e NSGA-III também foram calibrados pelo pacote IRACE, conforme apresentado na Subseção 6.2, ou seja, o mesmo procedimento usado para calibrar os parâmetros do MO-GVNS1 e do P-ILS é usado para calibrar os parâmetros dos algoritmos MO-GVNS2, MO-GVNS3 de Duarte et al. (2015), MO-RVNS e NSGA-III. Os parâmetros calibrados do IRACE para esses quatro algoritmos são mostrados na Tabela 6.2. A notação adotada nesta tabela é: MP_Level = nível máximo de perturbação; $nPop$ = tamanho da população; $probCross$ = probabilidade de cruzamento; $probMut$ = probabilidade de mutação e N_r = número de pontos de referência.

Tabela 6.2: Parâmetros dos Algoritmos MO-GVNS2, MO-GVNS3, MO-RVNS e NSGA-III.

Algoritmo	*Max*	*levelMax*	MP_Level	$nPop$	$probCross$	$probMut$	N_r
MO-GVNS2	150	3	-	-	-	-	-
MO-GVNS3	-	-	3	-	-	-	-
MO-RVNS	150	3	-	-	-	-	-
NSGA-III	-	-	-	100	0.85	0.05	15

Para comparar os seis algoritmos considerados, o mesmo critério de parada foi estabelecido para todos eles. O critério de parada adotado foi o tempo de execução, dado por $25 \times n \times m \times m_i$ milissegundos. Nos testes finais, o conjunto total com 1008 instâncias foi considerado. Cada algoritmo foi executado 30 vezes para cada instância, gerando, para cada uma dessas combinações, um conjunto de soluções não-dominadas normalizadas. Os resultados foram compilados para cada um dos 28 grupos de instância e, em seguida, os valores das métricas *Hypervolume*, *Epsilon*, *Spacing* e HCC apresentados foram calculados pelo pacote *Evolutionary Multiobjective Algoritms Optimization Algorithms* – EMOA (Mersmann, 2012), disponível no software de estatística *R*. Esses resultados são apresentados e analisados na Seção 6.3
.

6.3 Resultados do MOHFS com Três objetivos

Nesta seção são apresentados os resultados obtidos após a execução dos testes de calibração explicados na Seção 2.4 do Capítulo 2. Os valores médios das métricas *Hypervolume*, *Epsilon*, *Spacing* e HCC estão mostrados nas Tabelas a seguir.

As Tabelas 6.3, 6.4, 6.5 e 6.6 mostram os valores médios para instâncias pequenas de cada um dos algoritmos nas quatro métricas consideradas. As Tabelas 6.7, 6.8, 6.9 e 6.10 mostram os valores médios para instâncias grandes de cada um dos algoritmos nas quatro métricas consideradas. Nessas tabelas, a primeira coluna identifica o algoritmo; as próximas colunas trazem os valores médios da métrica *hypervolume*, *epsilon*, *spacing* e *hcc*, respectivamente, para os diferentes grupos de instâncias, separados pelo número de tarefas.

Tabela 6.3: Valores médios para a métrica *Hypervolume* nas instâncias pequenas ($n \in \{5, 7, 9, 11, 13, 15\}$).

Algoritmo	$n = 5$	$n = 7$	$n = 9$	$n = 11$	$n = 13$	$n = 15$
P-ILS	0.80	0.86	0.85	0.88	0.89	0.85
MO-GVNS1	0.80	0.86	0.85	0.89	0.89	0.85
MO-GVNS2	0.77	0.82	0.78	0.79	0.78	0.75
MO-GVNS3	0.71	0.77	0.69	0.72	0.70	0.65
MO-RVNS	0.36	0.30	0.25	0.20	0.19	0.19
NSGA-III	0.59	0.60	0.50	0.47	0.44	0.35

Tabela 6.4: Valores médios para a métrica *Epsilon* nas instâncias pequenas ($n \in \{5, 7, 9, 11, 13, 15\}$).

Algoritmo	$n = 5$	$n = 7$	$n = 9$	$n = 11$	$n = 13$	$n = 15$
P-ILS	0.17	0.10	0.09	0.08	0.08	0.08
MO-GVNS1	0.17	0.10	0.09	0.07	0.08	0.09
MO-GVNS2	0.20	0.14	0.15	0.15	0.15	0.15
MO-GVNS3	0.23	0.17	0.21	0.19	0.20	0.20
MO-RVNS	0.53	0.53	0.55	0.58	0.58	0.58
NSGA-III	0.31	0.30	0.36	0.39	0.42	0.48

O teste de Levene (Levene, 1960) foi aplicado para analisar os dados relacionados aos resultados obtidos. Esse teste avalia a igualdade da variância entre dados de diferentes amostras. Se o valor de p-valor gerado pela estatística for maior que o nível de significância de 5%, a hipótese de igualdade de variância não é rejeitada.

O teste aplicado mostrou diferenças estatísticas dos algoritmos MO-GVNS1 e P-ILS em relação a todos os outros algoritmos nas métricas *Hypervolume* e *Epsilon*, para todas as classes de instâncias. No entanto, para a métrica *Spacing*, as diferenças estatisticamente significativas entre os algoritmos só foram notadas a partir de $n > 11$.

Tabela 6.5: Valores médios para a métrica *Spacing* (%) nas instâncias pequenas ($n \in \{5, 7, 9, 11, 13, 15\}$).

Algoritmo	$n = 5$	$n = 7$	$n = 9$	$n = 11$	$n = 13$	$n = 15$
P-ILS	4.32	1.63	0.74	0.84	0.44	0.32
MO-GVNS1	4.50	1.37	0.76	0.67	0.44	0.46
MO-GVNS2	5.07	1.97	1.29	1.05	0.88	0.66
MO-GVNS3	6.45	2.74	2.74	2.43	2.27	1.52
MO-RVNS	5.83	3.99	3.39	2.92	2.24	2.32
NSGA-III	4.10	2.51	2.25	1.94	1.25	1.37

Tabela 6.6: Valores médios para a métrica *HCC* nas instâncias pequenas ($n \in \{5, 7, 9, 11, 13, 15\}$).

Algoritmo	$n = 5$	$n = 7$	$n = 9$	$n = 11$	$n = 13$	$n = 15$
P-ILS	2.52	5.84	11.47	12.26	17.73	18.50
MO-GVNS1	2.49	5.86	11.47	12.83	17.69	18.39
MO-GVNS2	3.51	8.16	13.65	13.32	16.93	17.92
MO-GVNS3	3.30	6.25	8.85	8.00	8.72	8.79
MO-RVNS	3.84	4.14	3.77	3.38	3.49	3.31
NSGA-III	2.72	3.40	3.65	3.50	3.67	3.61

Tabela 6.7: Valores médios para a métrica *Hypervolume* nas instâncias grandes ($n \in \{50, 100, 150, 200\}$).

Algoritmo	$n = 50$	$n = 100$	$n = 150$	$n = 200$
P-ILS	0.80	0.82	0.80	0.81
MO-GVNS1	0.80	0.83	0.80	0.81
MO-GVNS2	0.73	0.75	0.73	0.73
MO-GVNS3	0.61	0.65	0.59	0.61
MO-RVNS	0.18	0.18	0.19	0.14
NSGA-III	0.04	0.06	0.05	0.05

Tabela 6.8: Valores médios para a métrica *Epsilon* nas instâncias grandes ($n \in \{50, 100, 150, 200\}$).

Algoritmo	$n = 50$	$n = 100$	$n = 150$	$n = 200$
P-ILS	0.14	0.13	0.13	0.13
MO-GVNS1	0.14	0.13	0.14	0.13
MO-GVNS2	0.19	0.18	0.19	0.19
MO-GVNS3	0.28	0.26	0.29	0.29
MO-RVNS	0.66	0.68	0.69	0.64
NSGA-III	0.87	0.86	0.88	0.88

Como este indicador estima a variância das distâncias de soluções não-dominadas e os valores obtidos são muito próximos de zero, isso significa que as soluções resultantes dos algoritmos estão bem distribuídas. Já para a métrica HCC, os algoritmos MO-GVNS1 e P-ILS não se diferem estatisticamente do algoritmo MO-GVNS2, à exceção

Tabela 6.9: Valores médios para a métrica *Spacing* (%) nas instâncias grandes ($n \in \{50, 100, 150, 200\}$).

Algoritmo	$n = 50$	$n = 100$	$n = 150$	$n = 200$
P-ILS	0.04	0.04	0.03	0.03
MO-GVNS1	0.03	0.01	0.03	0.03
MO-GVNS2	0.04	0.08	0.05	0.04
MO-GVNS3	0.19	0.18	0.19	0.19
MO-RVNS	0.35	0.32	0.35	0.34
NSGA-III	0.23	0.23	0.25	0.25

Tabela 6.10: Valores médios para a métrica *HCC* nas instâncias grandes ($n \in \{50, 100, 150, 200\}$).

Algoritmo	$n = 50$	$n = 100$	$n = 150$	$n = 200$
P-ILS	27.39	23.05	21.33	26.20
MO-GVNS1	27.75	24.13	21.87	26.63
MO-GVNS2	26.61	22.98	19.17	26.09
MO-GVNS3	11.30	10.18	10.59	11.09
MO-RVNS	2.01	2.08	2.59	1.94
NSGA-III	2.75	2.46	2.78	2.83

nas instâncias com $n = 5$, $n = 7$ e $n = 9$.

Na aplicação desse teste não foram encontradas diferenças significativas entre os resultados dos algoritmos MO-GVNS1 e P-ILS.

O Anexo I traz as Figuras que representam os gráficos do intervalo de confiança para os valores médios para cada uma das métricas, nos quais o eixo horizontal identifica cada algoritmo abordado e o eixo vertical representa o valor médio da métrica. Nesse anexo, a Seção I.1 mostra os gráficos referentes a métrica *Hypervolume*, a Seção I.2 apresenta as figuras referentes ao *Epsilon*, as figuras da Seção I.3 representam a métrica *Spacing* e a Seção I.4 mostra os gráficos referentes a métrica HCC. O nível de confiança dos intervalos mostrados em todos esses gráficos é 95%.

As Figuras I.1(a) – I.1(f) mostram os gráficos do intervalo de confiança para os valores médios da métrica *Hypervolume* para cada grupo de instâncias pequenas, separados pelo número de tarefas. As Figuras I.2(a) – I.5(d), por sua vez, trazem os gráficos para as instâncias maiores para os valores médios da mesma métrica.

Da mesma forma, as Figuras I.6(a) – I.6(f) do Anexo I mostram o gráfico do intervalo de confiança para os valores médios da métrica *Epsilon* para instâncias pequenas. Além disso, as Figuras I.7(a) – I.10(d) trazem os gráficos para instâncias maiores referentes aos valores médios da mesma métrica.

Adicionalmente, as Figuras I.11(a) – I.11(f) do Anexo I mostram o gráfico do intervalo de confiança para os valores médios relativos à métrica *Spacing* para as

instâncias menores. As Figuras I.12(a) – I.15(d), por sua vez, trazem o gráfico do intervalo de confiança para os valores médios da mesma métrica considerando instâncias grandes.

Por fim, as Figuras I.16(a) – I.16(f) do Anexo I trazem o gráfico do intervalo de confiança para os valores médios relativos à métrica *HCC* para as instâncias de menor porte. Já as Figuras I.17(a) – I.20(d) trazem o gráfico do intervalo de confiança para os valores médios da mesma métrica considerando instâncias de grande porte.

Observa-se, pelas métricas *Hypervolume* e *Epsilon*, que para todas as classes de instâncias os algoritmos propostos P-ILS e MO-GVNS1 obtiveram melhor desempenho quando comparados a outros algoritmos da literatura. Como esses indicadores são métricas de convergência, pode-se notar que as estratégias abordadas são melhores que a dos outros métodos mais comuns aplicados na literatura, isto é, as soluções obtidas pelos algoritmos propostos estão mais próximas das soluções de Pareto Ótima. Os algoritmos propostos P-ILS e MO-GVNS1 possuem mecanismos de busca local, portanto esses resultados mostram a importância da aplicação desses mecanismos propostos.

Pela métrica *Spacing*, é possível notar que o desempenho dos algoritmos propostos P-ILS e MO-GVNS1 melhora na medida em que o tamanho das instâncias cresce. Nas instâncias menores, isto é, nos instâncias com $n = 5$, quase não existe diferença significativa entre os algoritmos nessa métrica. A partir das instâncias com $n \geq 7$, já é possível verificar uma diferença maior, que fica mais evidente nas instâncias com $n > 9$. A métrica *Spacing* mede a diversidade das soluções, portanto os algoritmos propostos possuem um bom espalhamento das soluções não-dominadas e isso fica mais evidente nas instâncias de grande porte.

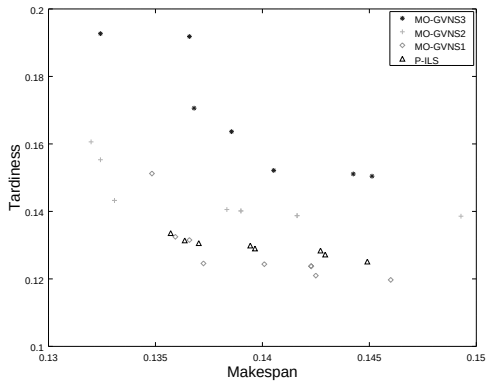
Para a métrica HCC, nota-se que para instâncias pequenas, isto é, $n < 9$, os algoritmos propostos P-ILS e MO-GVNS1 obtêm resultados com pouca diversidade de soluções, quando comparados com outros algoritmos da literatura. Nesse quesito, o algoritmo MO-GVNS2 se mostra melhor nessas instâncias. À medida que o tamanho da instância aumenta, a diversidade dos algoritmos propostos P-ILS e MO-GVNS1 melhora, e passa a ter um desempenho semelhante ao do algoritmo MO-GVNS2 e superior aos outros algoritmos da literatura.

Avaliando-se os resultados médios dos métodos abordados e as análises estatísticas realizadas, observa-se que os algoritmos MO-GVNS1 e P-ILS superam os outros métodos abordados neste experimento (algoritmo MO-GVNS de Duarte et al. (2015), MO-RVNS e NSGA-III) na maior parte dos casos. No entanto, não foram encontradas diferenças significativas entre esses dois algoritmos propostos. Logo, pode-se dizer que os algoritmos MO-GVNS1 e P-ILS são métodos competitivos em problemas de otimização combinatória multiobjetivo, tanto no quesito convergência quanto

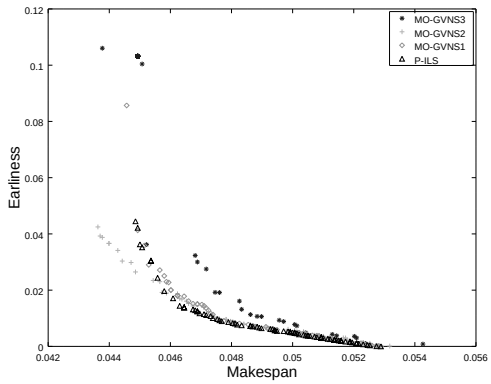
nas medidas de diversidade.

Para ilustrar a convergência e a diversidade das soluções não-dominadas obtidas, as fronteiras de Pareto em duas dimensões normalizadas obtidas por esses algoritmos são mostradas nas Figuras 6.4(a) – 6.4(c). Nessa análise foram considerados apenas os algoritmos P-ILS, MO-GVNS1, MO-GVNS2 e NSGA-III, pois já foi verificado que esses métodos superam os demais algoritmos. Em cada uma das figuras, um dos três objetivos foi fixado e os outros dois considerados.

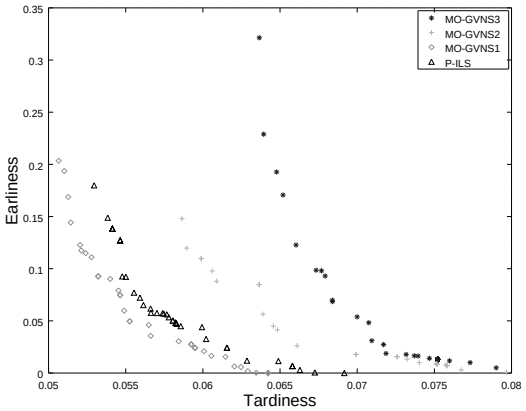
Na Figura 6.4(a), a soma ponderada da antecipação está fixada; o *makespan* está no eixo horizontal e a soma ponderada do atraso está no eixo vertical. Na Figura 6.4(b), a soma ponderada dos atrasos é fixada; o *makespan* está no eixo horizontal e a soma ponderada das antecipações está no eixo vertical. Por outro lado, na Figura 6.4(c), o *makespan* é fixo; a soma ponderada dos atrasos está no eixo horizontal e a soma ponderada das antecipações está no eixo vertical.



(a) C_{max} versus $\sum w_j T_j$



(b) C_{max} versus $\sum u_j E_j$



(c) $\sum w_j T_j$ versus $\sum u_j E_j$

Figura 6.4: Fronteira de Pareto normalizada de uma instância com 50 tarefas, 4 estágios e 2 máquinas em cada estágio.

Os dados utilizados nesses gráficos foram obtidos através da execução dos algoritmos citados acima, e representam a Fronteira de Pareto normalizada de uma instância com $n = 50$, $m = 4$ e $m_i = 2$. Comportamento semelhante foi observado em todos os instâncias de grande porte. Vale a pena notar que nem todos os pontos da borda tridimensional estão presentes em algum desses gráficos.

6.4 Resultados do MOHFS com cinco objetivos

Esta seção mostra os resultados obtidos considerando o problema com cinco objetivos. Os algoritmos propostos no Capítulo 5 foram adaptados para o problema com cinco objetivos. Os testes foram executados em um computador Intel Core i7, 2.00 GHz, com 16 GB de memória RAM sob sistema operacional Linux Ubuntu 64 bits. A Subseção 6.4.1 traz os resultados com agregação de objetivos e a Subseção 6.4.2 apresenta os resultados com os cinco objetivos simultaneamente.

O conjunto de instâncias utilizadas nos experimentos foi o mesmo descrito na Seção 6.1. Esse conjunto é composto por 432 instâncias de pequeno porte e outras 576 instâncias maiores. Os instâncias estão subdivididos pelo número de tarefas (n), máquinas (m) e máquinas por estágio (m_i). As instâncias menores possuem as seguintes configurações: $n \in \{5, 7, 9, 11, 13, 15\}$, $m \in \{2, 3\}$ e $m_i = 3$. As instâncias maiores tem $n \in \{50, 100, 150, 200\}$ e $m \in \{4, 8\}$ and $m_i \in \{2, 4\}$. Assim, tem-se um total de 1008 instâncias divididas em 28 grupos de 36.

6.4.1 Agregação e redução para três objetivos

Inicialmente, os algoritmos foram executados para o problema com os objetivos agregados. Isso foi feito combinando-se os objetivos (i) e (iv) e os objetivos (ii) e (iii), isto é, o problema reduzido considera apenas três objetivos de minimização: *makespan* e somatório dos tempos de ociosidade ($C_{\max} + \sum I_j$); somatórios ponderados dos atrasos e das antecipações ($\sum w_j T_j + \sum u_j E_j$) e, finalmente, número de tarefas atrasadas ($\sum U_j$). Essas junções de objetivos é possível, pois o *makespan* e a ociosidade são medidas de mesma grandeza, e o mesmo acontece com a combinação do atraso e da antecipação.

Cada algoritmo foi executado 30 vezes para cada instância, gerando, para cada uma dessas combinações, um conjunto de soluções não-dominadas. Ao final das execuções os conjuntos para o problema reduzido foi comparado com os conjuntos para o problema completo.

O Anexo II na Seção II.1 traz as Figuras que representam os gráficos do intervalo de confiança para os valores médios para cada uma das métricas. Nessa seção, a

Subseção II.1.1 mostra as figuras referentes a métrica *Hypervolume*, a Subseção II.1.2 apresenta os gráficos referentes ao *Epsilon*, a Subseção II.1.3 mostra os gráficos quem representam a métrica *Spacing* e a Subseção II.1.4 mostra os gráficos referentes a métrica HCC. O nível de confiança dos intervalos é 95%.

As Figuras II.1(a) – II.1(f) trazem os gráficos da métrica *Hypervolume* para o grupo de instâncias pequenas, separados pelo número de tarefas. Enquanto, as Figuras II.2(a) – II.5(d), por sua vez, trazem os gráficos para as instâncias maiores.

As Figuras II.6(a) – II.6(f) mostram o gráfico dos valores médios da métrica *Epsilon* para instâncias pequenas e as Figuras II.7(a) – II.10(d) trazem os gráficos para instâncias maiores.

Pelas métricas *Hypervolume* e *Epsilon*, os algoritmos propostos P-ILS, MO-GVNS1 e MO-GVNS2 obtiveram um ótimo desempenho quando comparados aos outros algoritmos da literatura. Esse comportamento é muito semelhante ao observado nos resultados da Seção 6.3.

As Figuras II.11(a) – II.11(f) mostram o gráfico do intervalo de confiança para os valores médios relativos à métrica *Spacing* para as instâncias menores e as Figuras II.12(a) – II.15(d) mostram o gráfico do intervalo de confiança para os valores médios considerando instâncias grandes.

Da mesma forma, as Figuras II.16(a) – II.16(f) trazem o gráfico para os valores médios relativos à métrica *HCC* para as instâncias de menor porte. Já as Figuras II.17(a) – II.20(d) trazem o gráfico do intervalo de confiança para os valores médios da mesma métrica considerando instâncias de grande porte.

Assim, as métricas *Spacing* e HCC mostram comportamento semelhante aos apresentados na Seção 6.3, isto é, para essas métricas os resultados dos algoritmos propostos P-ILS e MO-GVNS1 melhora na medida em que o tamanho das instâncias cresce.

6.4.2 Cinco objetivos simultâneos

Posteriormente, o problema foi considerado de forma completa, ou seja, os cinco objetivos foram considerados simultaneamente. Assim, cada algoritmo foi executado 30 vezes para cada instância, gerando novamente, para cada uma dessas combinações, um novo conjunto de soluções não-dominadas.

No Anexo II a Seção II.2 apresenta as Figuras que representam os gráficos do intervalo de confiança para os valores médios para cada uma das métricas, nos quais o eixo horizontal identifica cada algoritmo abordado e o eixo vertical representa o valor médio da métrica. Nesse anexo, as figuras da Subseção II.2.1 representam os gráficos referentes a métrica *Hypervolume*, a Subseção II.2.2 apresenta as figuras referentes

a métrica *Epsilon*, a Subseção II.2.3 apresenta os gráficos da métrica *Spacing* e as figuras da Seção II.2.4 mostram os gráficos referentes a métrica HCC. O nível de confiança dos intervalos desses gráficos, assim como nos demais, é 95%.

As Figuras II.21(a) – II.21(f) trazem os gráficos do intervalo de confiança para os valores médios da métrica *Hypervolume* para o grupo de instâncias pequenas, separados pelo número de tarefas. As Figuras II.22(a) – II.25(d), por sua vez, trazem os gráficos para as instâncias maiores.

Da mesma forma, as Figuras II.26(a) – II.26(f) mostram o gráfico dos valores médios da métrica *Epsilon* para instâncias pequenas e as Figuras II.27(a) – II.30(d) trazem os gráficos para instâncias maiores.

Pelas métricas *Hypervolume* e *Epsilon*, nota-se um comportamento muito semelhante ao observado nos resultados com três objetivos para os algoritmos P-ILS, MO-GVNS1, MO-GVNS2, MO-GVNS3 e MO-RVNS, porém o NSGA-III mostrou um desempenho melhor com um número maior de objetivos. Isto é, os algoritmos propostos MO-GVNS1 e MO-GVNS2 é superior aos outros algoritmos com exceção do NSGA-III, o qual possui desempenho igual aos dois.

As Figuras II.31(a) – II.31(f) mostram o gráfico do intervalo de confiança para os valores médios relativos à métrica *Spacing* para as instâncias menores e as Figuras II.32(a) – II.35(d) mostram o gráfico do intervalo de confiança para os valores médios considerando instâncias grandes.

Já em relação métrica *Spacing* os algoritmos mostram um comportamento mostram semelhante aos resultados com três objetivos, isto é, os resultados dos algoritmos propostos P-ILS e MO-GVNS1 melhora na medida em que o tamanho das instâncias cresce, enquanto os resultados do NSGA-III se mantém estável.

Por fim, as Figuras II.36(a) – II.36(f) trazem o gráfico para os valores médios relativos à métrica *HCC* para as instâncias de menor porte. Já as Figuras II.37(a) – II.40(d) trazem o gráfico do intervalo de confiança para os valores médios da mesma métrica considerando instâncias de grande porte.

Na métrica HCC, pode-se notar, mais uma vez, que os algoritmos P-ILS, MO-GVNS1, MO-GVNS2 e NSGA-III são superiores aos demais em relação instâncias com $n > 7$, e entre si, não existe superioridade.

Estes resultados mostram que os algoritmos propostos P-ILS e MO-GVNS1 são estratégias competitivas em problemas de otimização multiobjetivo, mesmo em problemas com um número maior do que três objetivos. Também, é possível notar que o desempenho do NSGA-III melhora à medida que o número de objetivos cresce.

Capítulo 7

Conclusões

Este capítulo apresenta as conclusões deste trabalho. Na Seção 7.1 é feita uma apresentação das considerações finais. Na Seção 7.2 são listados os artigos publicados durante a execução deste trabalho. A Seção 7.3 apresenta algumas propostas de trabalhos futuros.

7.1 Considerações finais

Esta tese tratou o Problema *Flow Shop* Híbrido Multiobjetivo com características muito comuns no mundo real. O problema tratado faz parte de uma classe de problemas de sequenciamento de tarefas, os quais são comuns em diversas áreas do conhecimento, sobretudo no contexto da produção industrial.

Neste trabalho foi considerado um problema de sequenciamento de tarefas multiobjetivo em um ambiente *Flow Shop* Híbrido. Nesse tipo de ambiente, todas as tarefas seguem o mesmo fluxo de estágios, e em cada estágio existe um número de máquinas paralelas. Esses ambientes podem possuir diversas características e restrições, aqui foram consideradas as mais próximas da realidade. Algumas dessas características são a existência de máquinas paralelas não-relacionadas em cada estágio, datas de entrega, custos para atraso e antecipação, elegibilidade de máquinas e salto de estágios. Nesse problema o desafio é buscar o conjunto ótimo de soluções não-dominadas, as chamadas fronteiras de Pareto.

No problema abordado considerou-se, inicialmente, três objetivos conflitantes: as minimizações do *makespan*, da soma ponderada dos atrasos e da soma ponderada das antecipações. Posteriormente, foram adicionados dois novos objetivos: as minimizações da soma dos tempos de ociosidade e do número de tarefas atrasadas, assim os objetivos *makespan* e tempos de ociosidade foram combinados em um, o mesmo foi feito com os objetivos soma ponderada dos atrasos e soma ponderada das

antecipações. Por fim, os cinco objetivos foram considerados simultaneamente.

Para resolver o problema com três e cinco objetivos foram propostas duas adaptações de algoritmos multiobjetivo. Os algoritmos propostos foram baseados nos métodos MO-GVNS e P-ILS. Nesses algoritmos, as soluções iniciais são geradas por um método de construção GRASP e a exploração do espaço de soluções é feita por meio de perturbações nas soluções geradas. O algoritmo MO-GVNS ainda aplica duas buscas locais, sendo uma feita com o método MO-VND usando os movimentos inserção, troca e inversão, e a outra baseada no trabalho de Arroyo et al. (2011). Uma versão desse algoritmo sem nenhuma busca local, denominada versão MO-RVNS, também foi analisada.

Quatro métricas foram utilizadas para comparar os algoritmos propostos com outros algoritmos da literatura. Duas métricas de convergência: *Hypervolume* e *Epsilon*, e outras duas métricas de diversidade: *Spacing* e HCC.

Para o problema com três objetivos, os resultados mostraram uma superioridade dos algoritmos propostos na maior parte dos casos, especialmente em instâncias de grande porte. Nas métricas de convergência a superioridade dos algoritmos propostos é bem notável em todas as classes de instâncias, que se diferenciam pelo número de tarefas, número de estágios e número de máquinas por estágio. Já nas métricas de diversidade a diferença notável só aparece nas instâncias com um número de tarefas maior que nove. Isso ocorre porque em instâncias menores a quantidade de possíveis soluções é consideravelmente menor.

Os resultados da aplicação dos algoritmos propostos ao problema com cinco objetivos mostraram que eles são competitivos quando comparados ao NSGA-III, que é o algoritmo atualmente estado da arte para problemas com mais de dois objetivos. Para cinco objetivos, o NSGA-III apresenta uma melhora nos seus resultados; o que já era esperado pois esse método foi criado para resolução de problemas multiobjetivo, especialmente aqueles os quais o número de objetivos sejam maiores que três. Apesar disso, os algoritmos propostos mantiveram bons resultados mesmo com aumento do número de objetivos.

7.2 Publicações Realizadas

Nesta Seção são apresentadas as publicações decorrentes da execução desta Tese. Os trabalhos listados a seguir, resultantes dos métodos desenvolvidos para tratar o problema MOHFS, foram publicados nos seguintes eventos e periódicos:

- DE SIQUEIRA, E.C.; SOUZA, M. J. F.; SOUZA, S. R. *A Multi-objective Variable Neighborhood Search algorithm for solving the Hybrid Flow Shop Problem.*

ELECTRONIC NOTES IN DISCRETE MATHEMATICS, v. 66, p. 87-94, 2018.

- DE SIQUEIRA, E. C.; SOUZA, M. J. F.; SOUZA, S. R. *An MO-GVNS algorithm for solving a multiobjective hybrid flow shop scheduling problem. International Transactions in Operational Research*, v. 27, p. 614-650, 2019. doi:10.1111/itor.12662

- DE SIQUEIRA, E. C.; SOUZA, M. J. F. ; SOUZA, S. R.; DIANA, R. O. M. *A Study concerning the application of Genetic Algorithms for solving the Multi-Objective Hybrid Flowshop Scheduling Problem.* In: XIII Encontro Nacional de Inteligência Artificial e Computacional, 2016, Recife. Anais do XIII ENIAC, 2016.

7.3 Propostas de trabalhos futuros

Esta Seção apresenta propostas de trabalhos futuros a esta Tese, são elas:

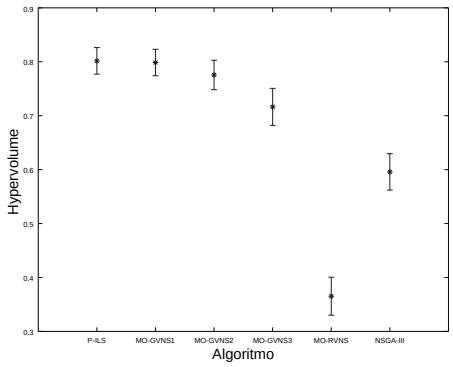
- a implementação de outras regras para alocação das tarefas nas máquinas no problema *Flow Shop* Híbrido multiobjetivo;
- a implementação de outros tipos de movimentos para exploração do espaço de busca aplicados ao problema *Flow Shop* Híbrido multiobjetivo;
- a implementação de outras metaheurísticas de busca local, bem como de métodos evolutivos aplicados ao problema *Flow Shop* Híbrido multiobjetivo;
- a implementação de algoritmos híbridos, combinando o poderio dos métodos de programação matemática com a flexibilidade dos métodos heurísticos, aplicados ao problema *Flow Shop* Híbrido multiobjetivo;
- o tratamento de outras funções objetivos ao problema *Flow Shop* Híbrido multiobjetivo, como por exemplo, a minimização do afastamento máximo ($L_{\max}$);
- o tratamento do problema *Flow Shop* Híbrido multiobjetivo com outras características comuns no mundo real, como por exemplo, tempos de *setup* dependentes da sequência e tempos de *release*;
- a implementação de outros tipos de relação de dominância para problemas com múltiplos objetivos, como por exemplo, a dominância de Lorenz (Kostreva e Ogryczak, 1999; Kostreva et al., 2004);

- a aplicação dos métodos aqui propostos em problemas de sequenciamento de tarefas com outros tipos de ambiente, como por exemplo, o ambiente *Job Shop*;

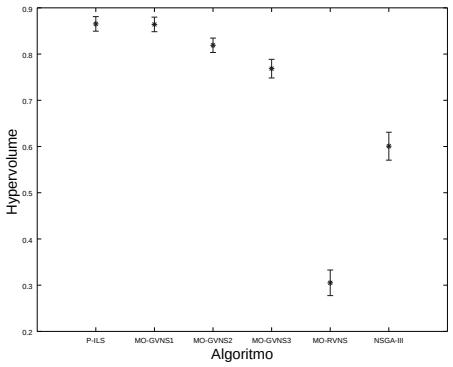
Anexo I

Análise gráfica da solução do problema multiobjetivo com três objetivos

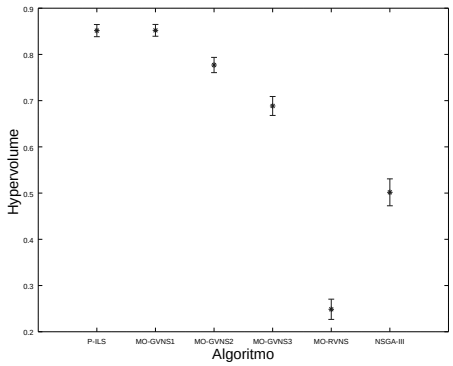
I.1 Análise da métrica *Hypervolume*



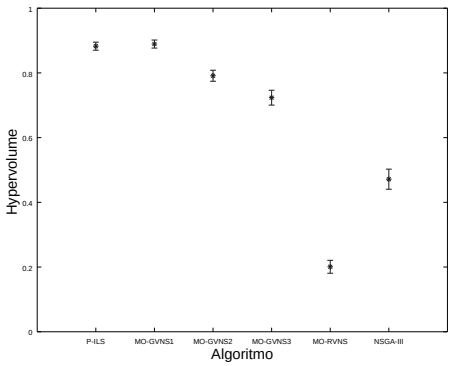
(a) $n = 5$



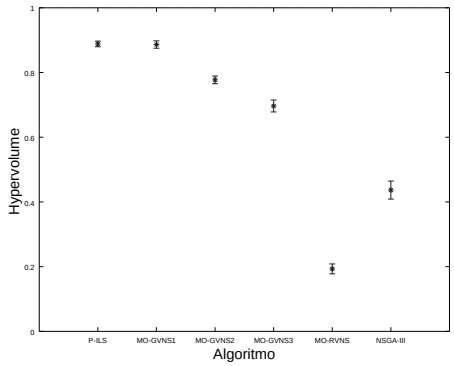
(b) $n = 7$



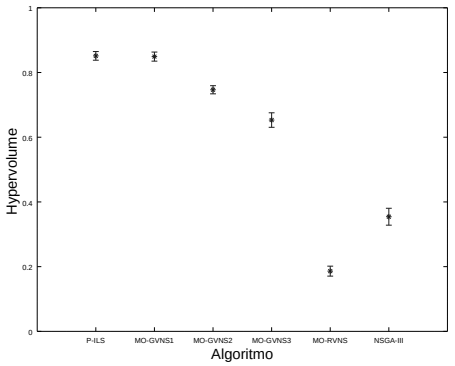
(c) $n = 9$



(d) $n = 11$

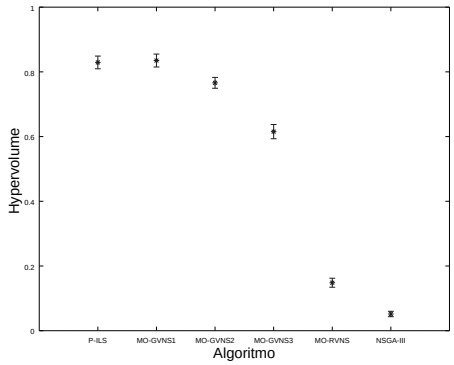


(e) $n = 13$

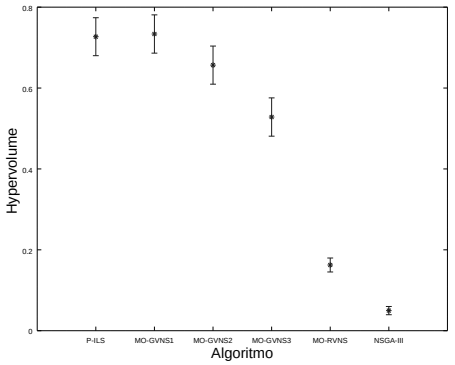


(f) $n = 15$

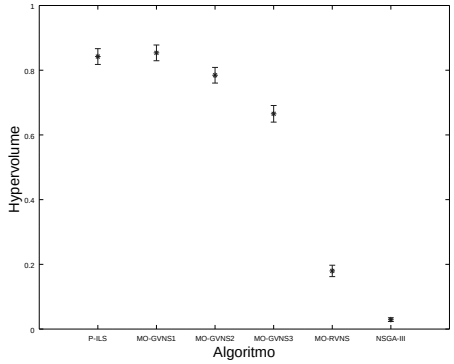
Figura I.1: Intervalo de Confiança na métrica *Hypervolume*: instâncias pequenas.



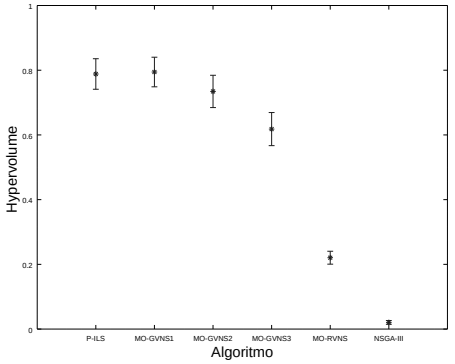
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

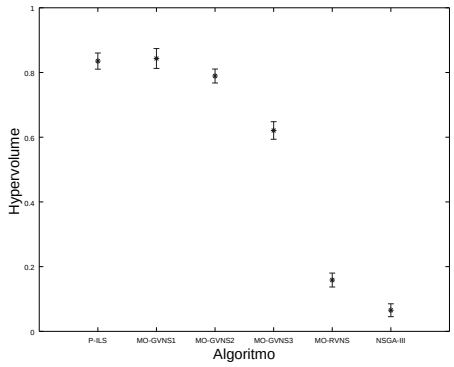


(c) $n = 50, m = 8, m_i = 2$

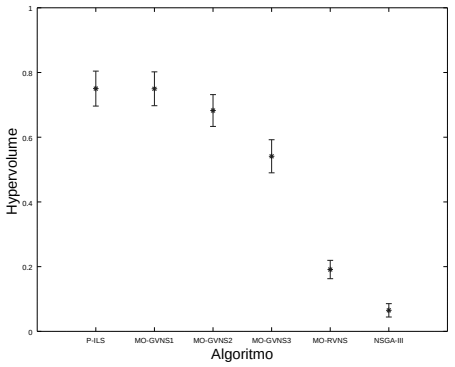


(d) $n = 50, m = 8, m_i = 4$

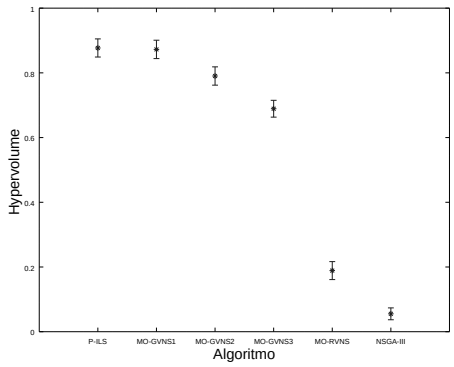
Figura I.2: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 50$)



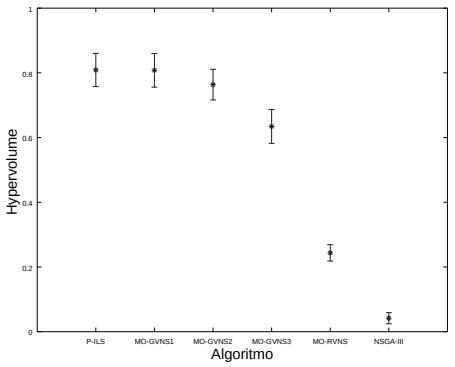
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

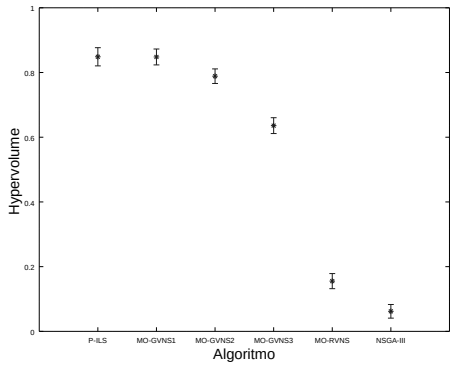


(c) $n = 100, m = 8, m_i = 2$

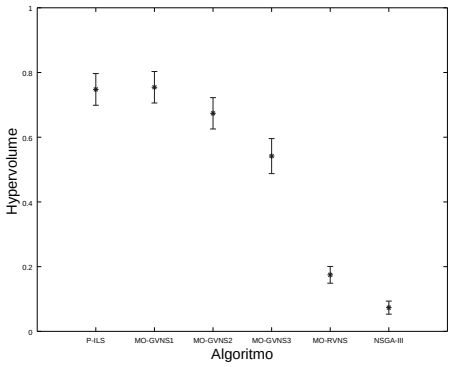


(d) $n = 100, m = 8, m_i = 4$

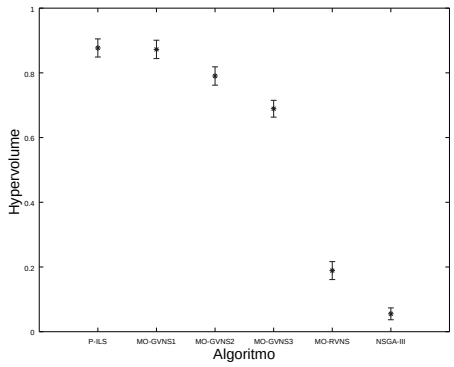
Figura I.3: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 100$)



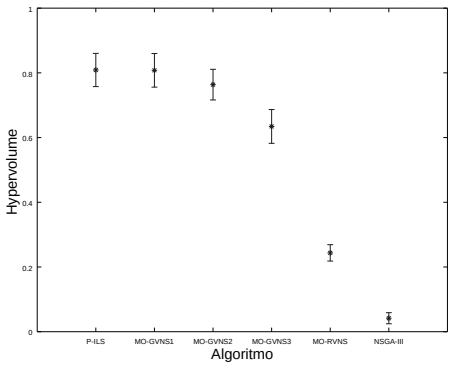
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

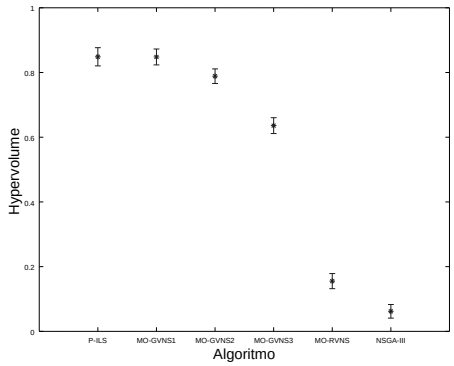


(c) $n = 150, m = 8, m_i = 2$

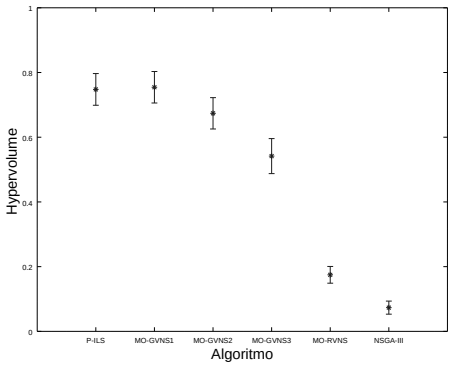


(d) $n = 150, m = 8, m_i = 4$

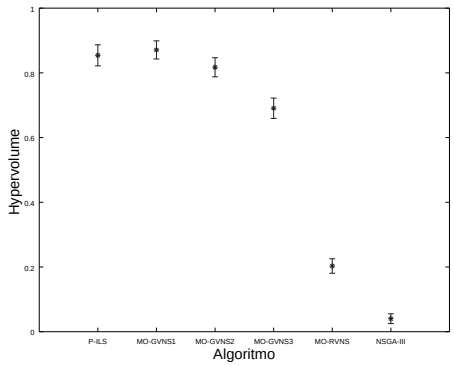
Figura I.4: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 150$)



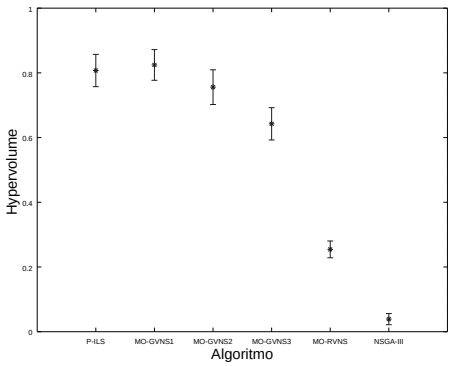
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



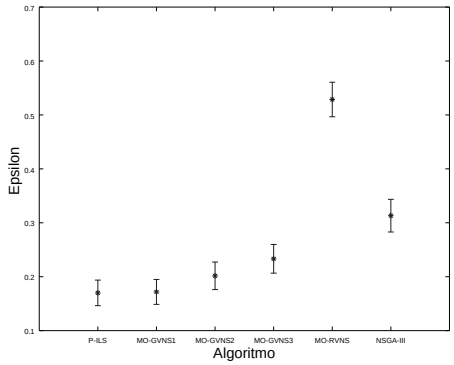
(c) $n = 200, m = 8, m_i = 2$



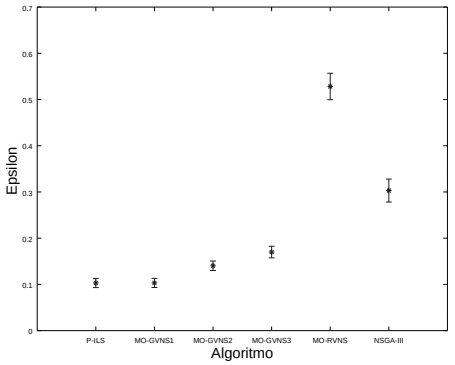
(d) $n = 200, m = 8, m_i = 4$

Figura I.5: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 200$)

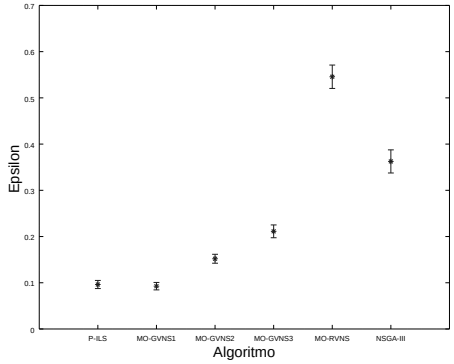
I.2 Análise da métrica *Epsilon*



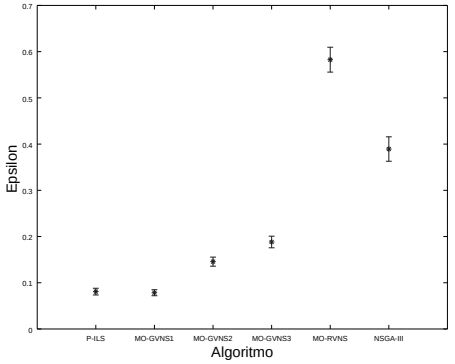
(a) $n = 5$



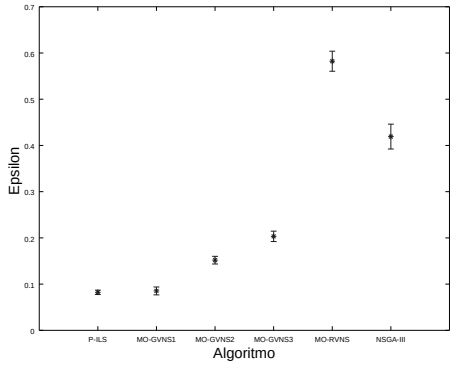
(b) $n = 7$



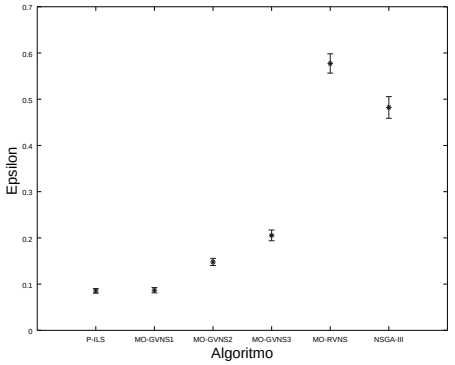
(c) $n = 9$



(d) $n = 11$

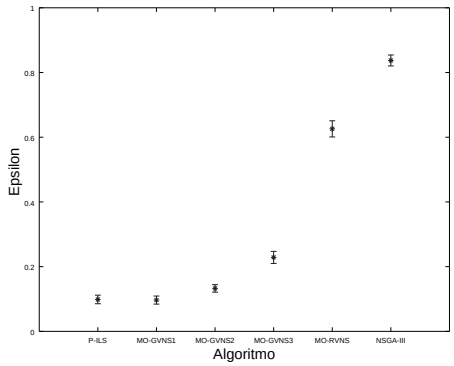


(e) $n = 13$

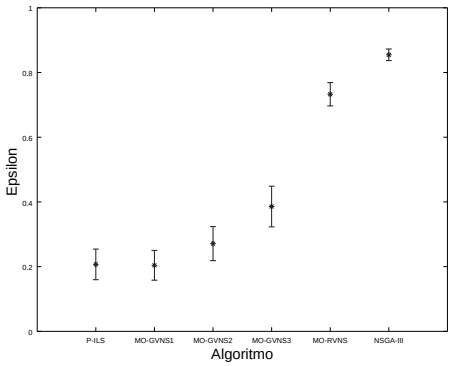


(f) $n = 15$

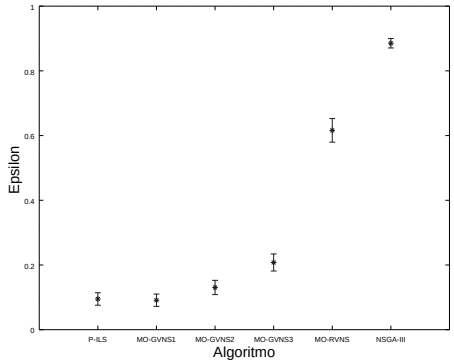
Figura I.6: Intervalo de Confiança na métrica *Epsilon*: instâncias pequenas



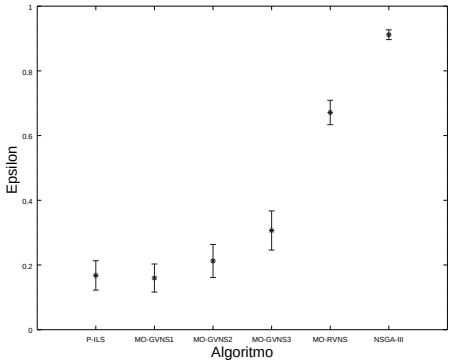
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

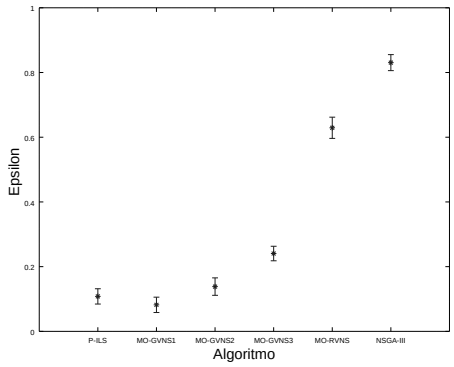


(c) $n = 50, m = 8, m_i = 2$

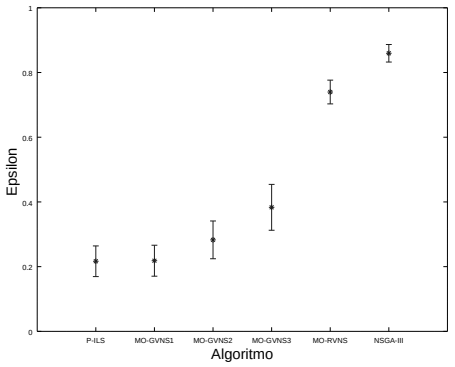


(d) $n = 50, m = 8, m_i = 4$

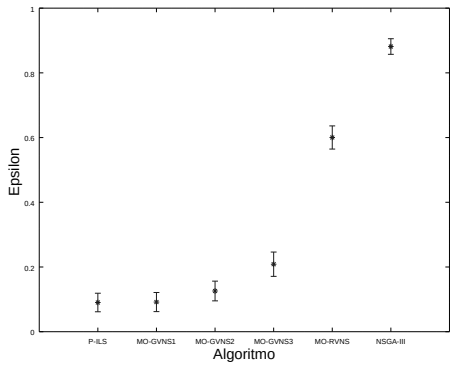
Figura I.7: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 50$)



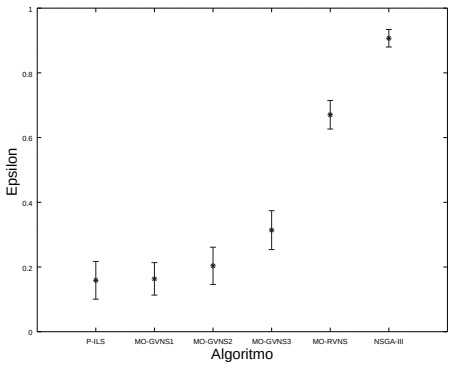
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

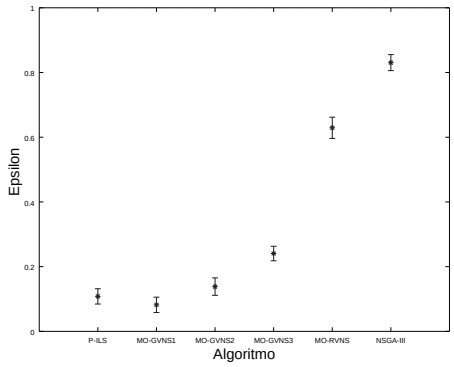


(c) $n = 100, m = 8, m_i = 2$

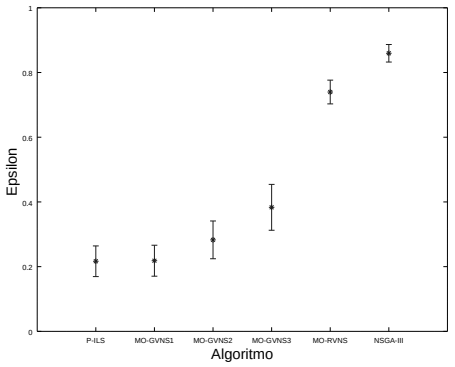


(d) $n = 100, m = 8, m_i = 4$

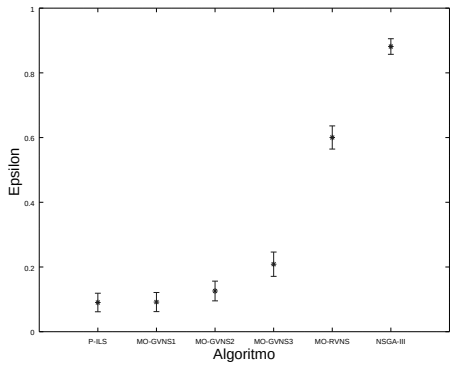
Figura I.8: Intervalo de Confiança métrica *Epsilon*: instâncias grandes ($n = 100$)



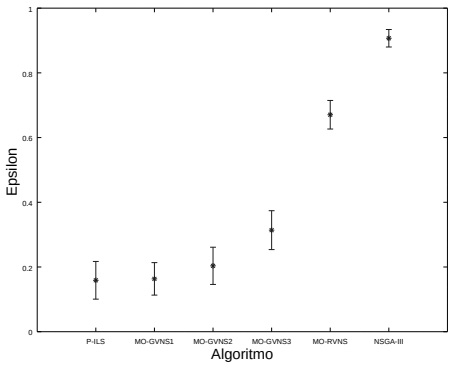
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

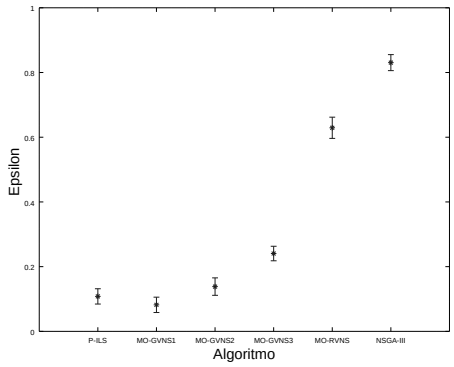


(c) $n = 150, m = 8, m_i = 2$

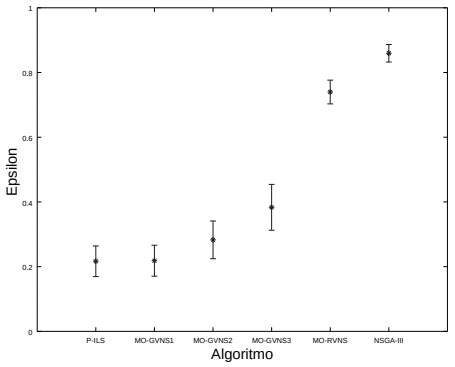


(d) $n = 150, m = 8, m_i = 4$

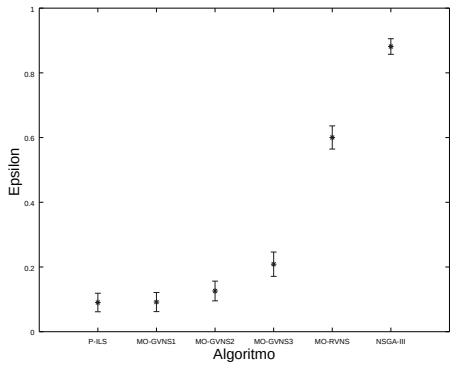
Figura I.9: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 150$)



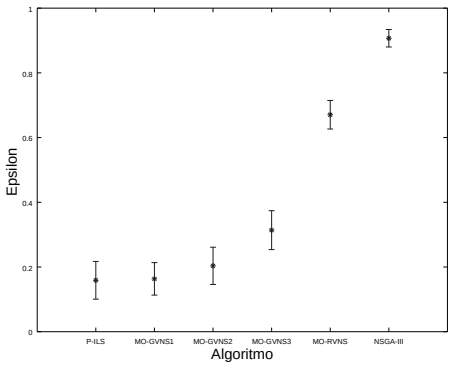
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



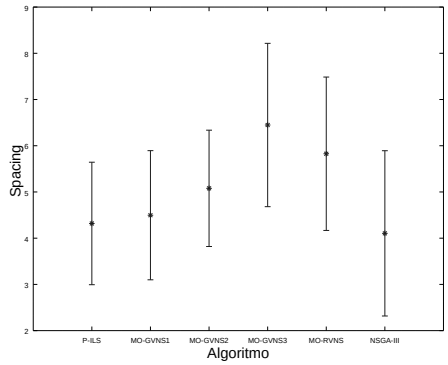
(c) $n = 200, m = 8, m_i = 2$



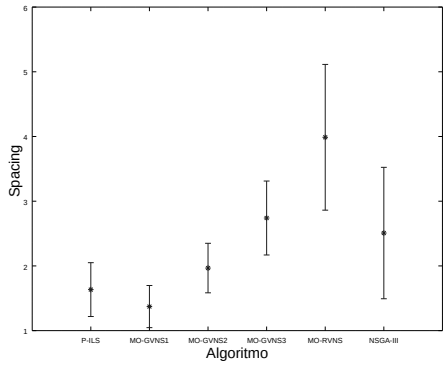
(d) $n = 200, m = 8, m_i = 4$

Figura I.10: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 200$)

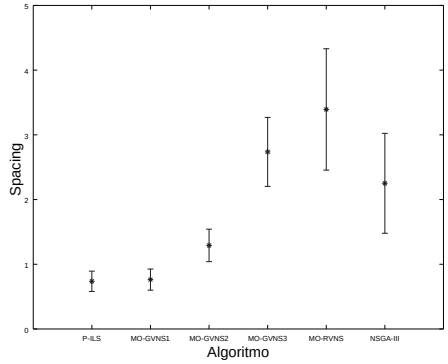
I.3 Análise da métrica *Spacing*



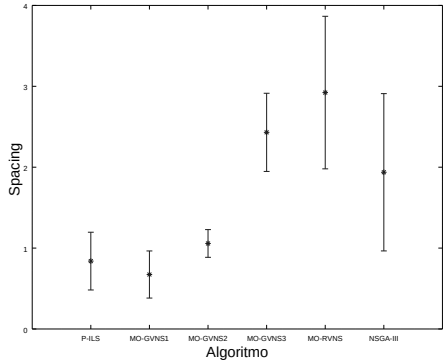
(a) $n = 5$



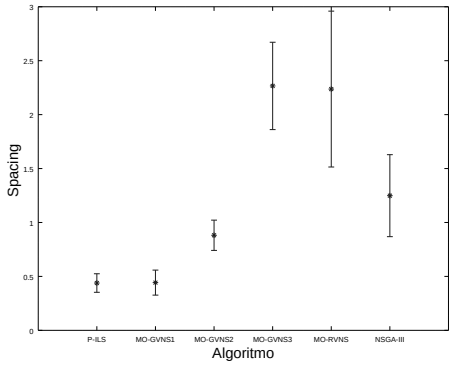
(b) $n = 7$



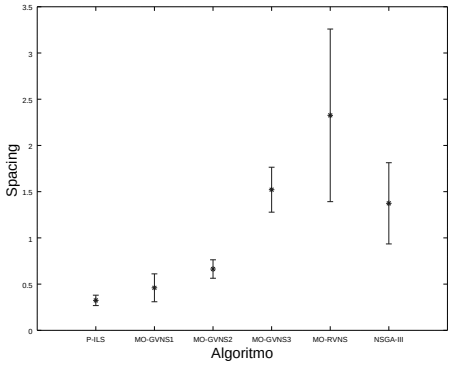
(c) $n = 9$



(d) $n = 11$

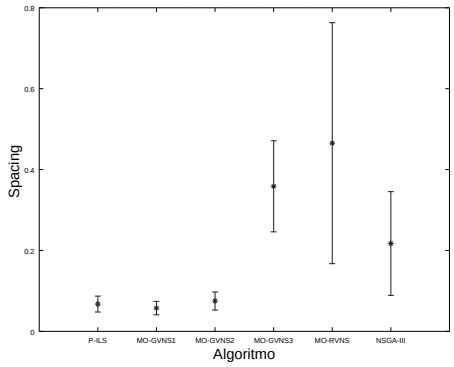


(e) $n = 13$

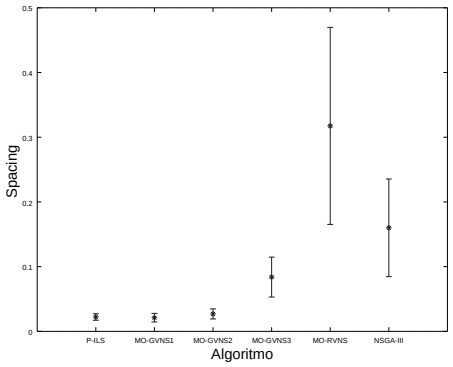


(f) $n = 15$

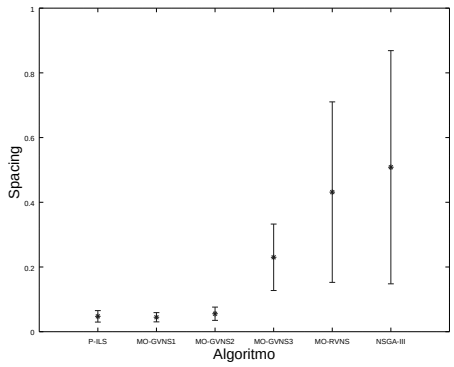
Figura I.11: Intervalo de Confiança na métrica *Spacing*: instâncias pequenas



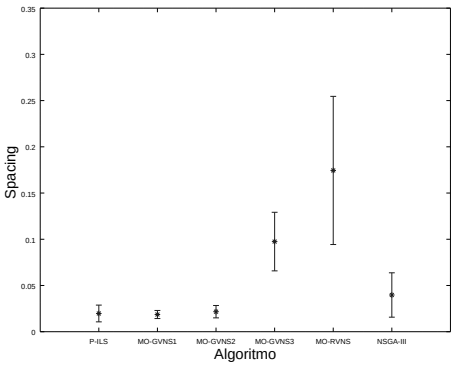
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

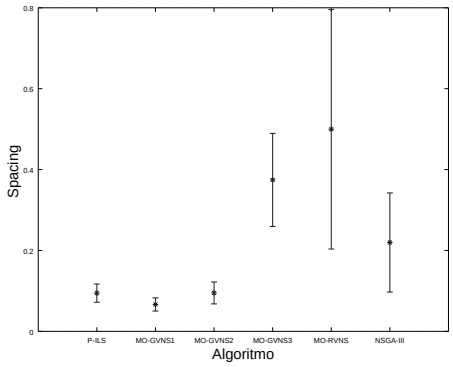


(c) $n = 50, m = 8, m_i = 2$

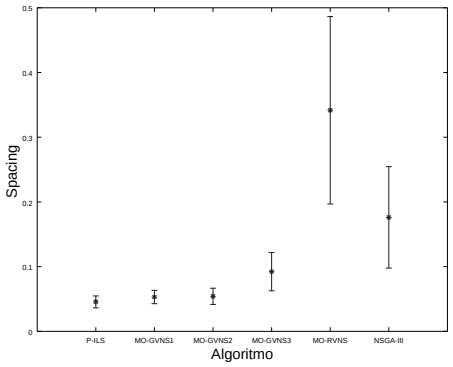


(d) $n = 50, m = 8, m_i = 4$

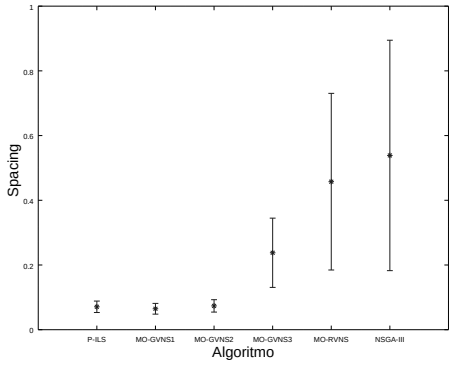
Figura I.12: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 50$)



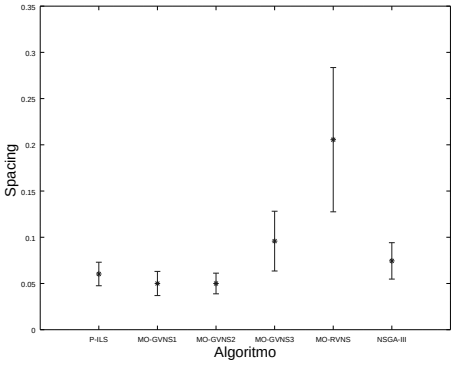
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

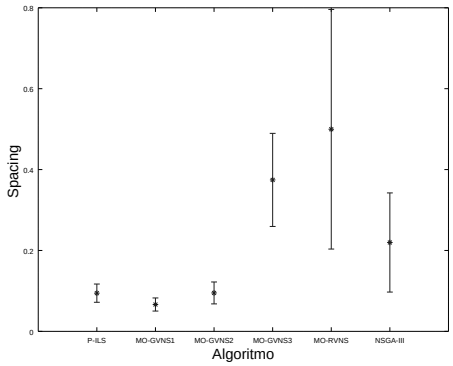


(c) $n = 100, m = 8, m_i = 2$

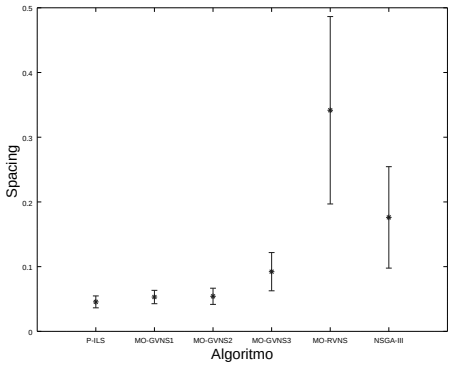


(d) $n = 100, m = 8, m_i = 4$

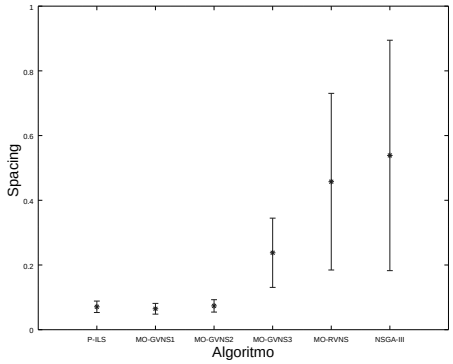
Figura I.13: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 100$)



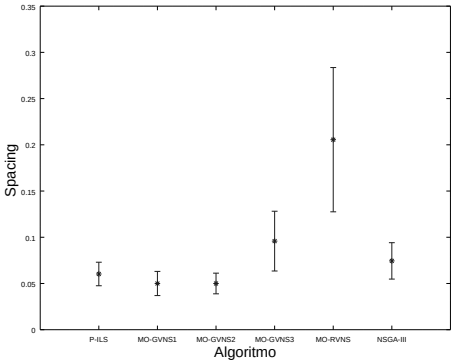
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

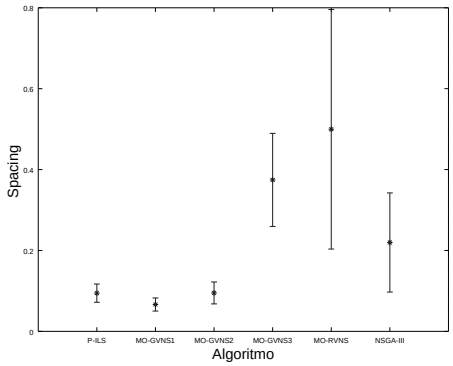


(c) $n = 150, m = 8, m_i = 2$

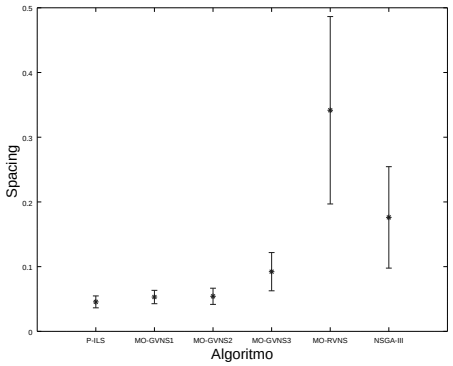


(d) $n = 150, m = 8, m_i = 4$

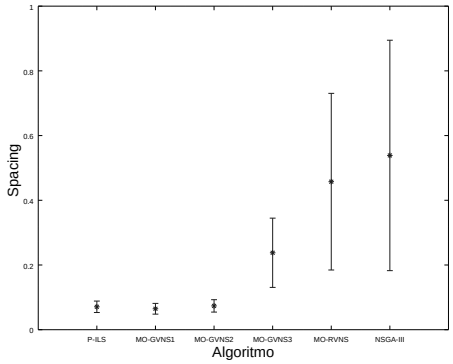
Figura I.14: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 150$)



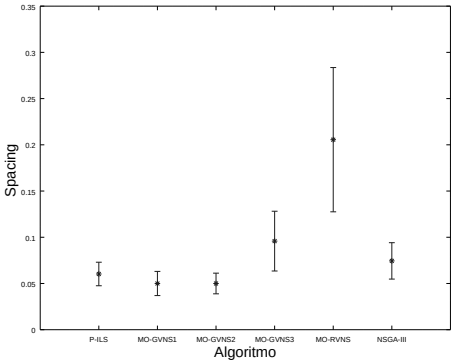
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



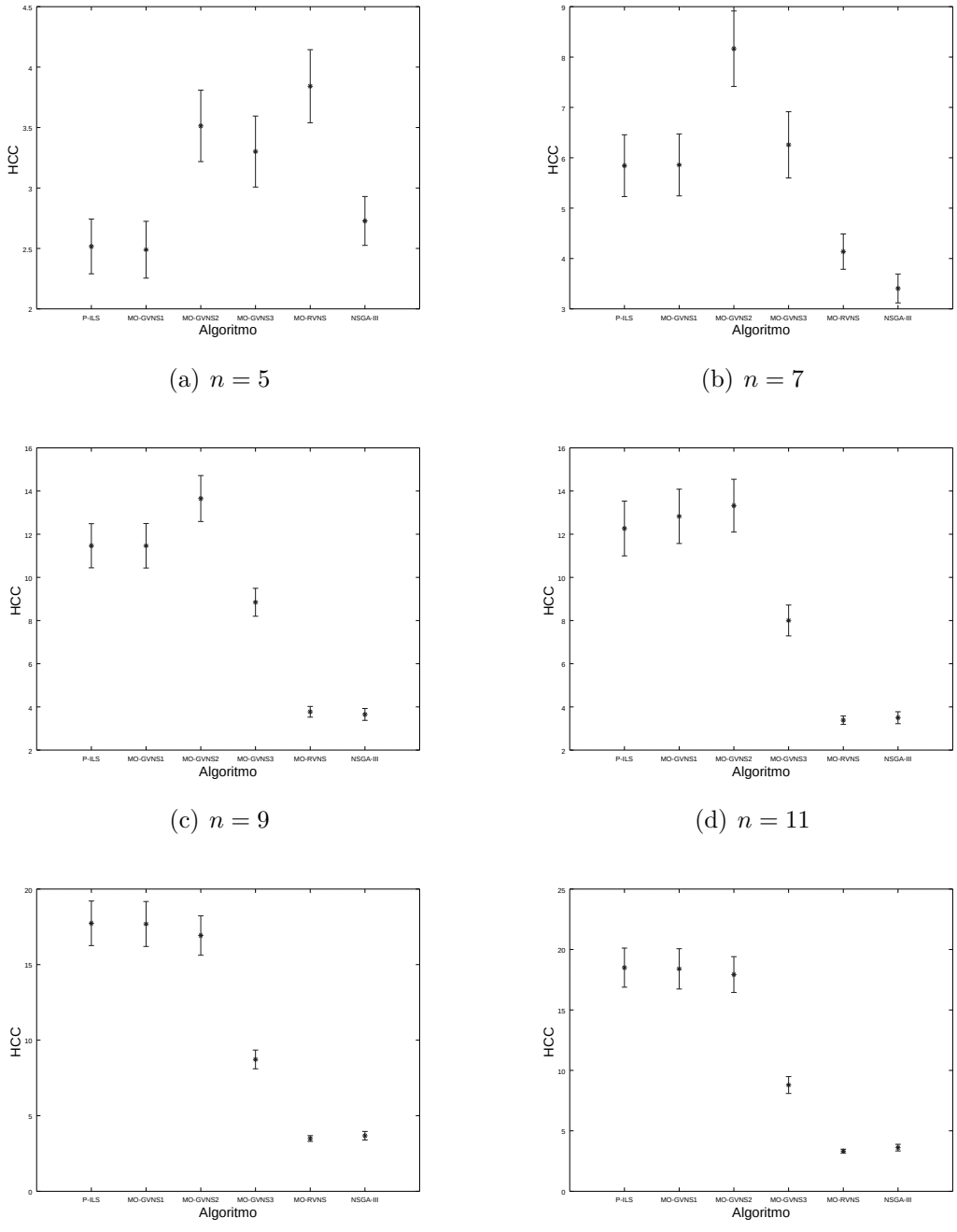
(c) $n = 200, m = 8, m_i = 2$



(d) $n = 200, m = 8, m_i = 4$

Figura I.15: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 200$)

I.4 Análise da métrica HCC

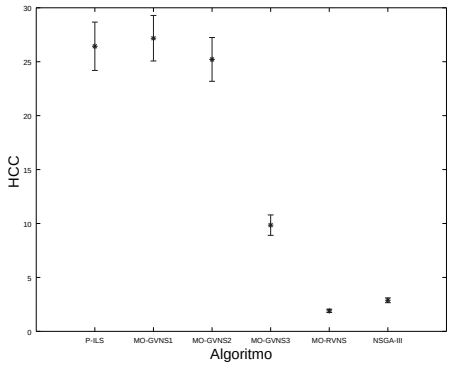


(a) $n = 5$ (b) $n = 7$

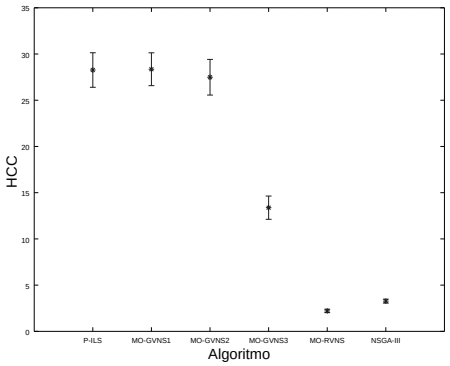
(c) $n = 9$ (d) $n = 11$

(e) $n = 13$ (f) $n = 15$

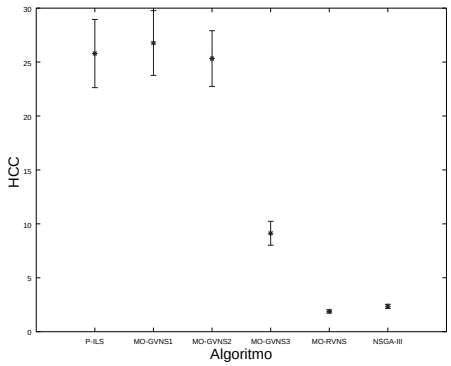
Figura I.16: Intervalo de Confiança na métrica *HCC*: instâncias pequenas



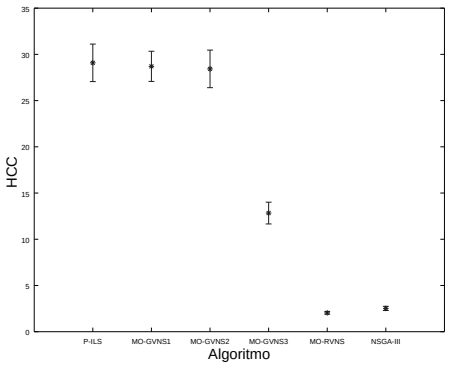
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

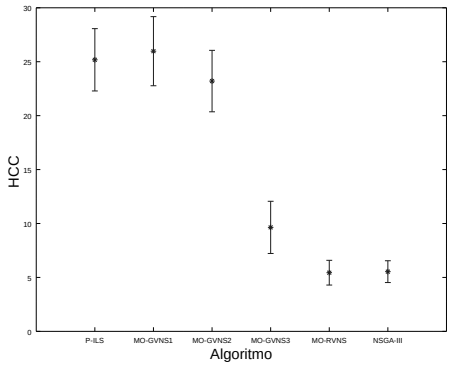


(c) $n = 50, m = 8, m_i = 2$

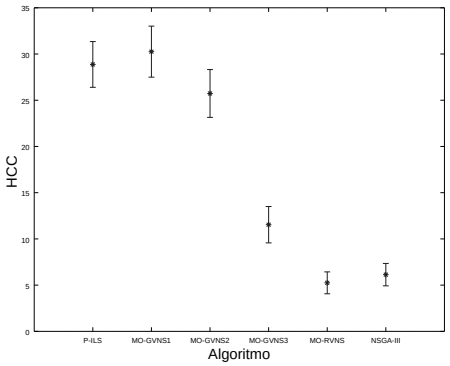


(d) $n = 50, m = 8, m_i = 4$

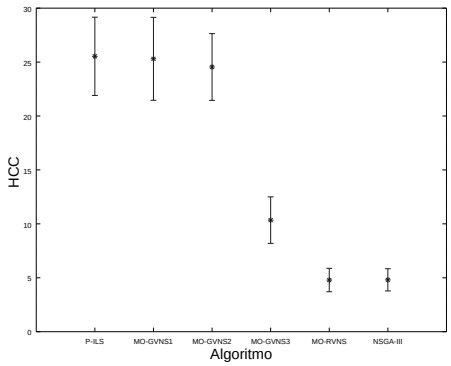
Figura I.17: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 50$)



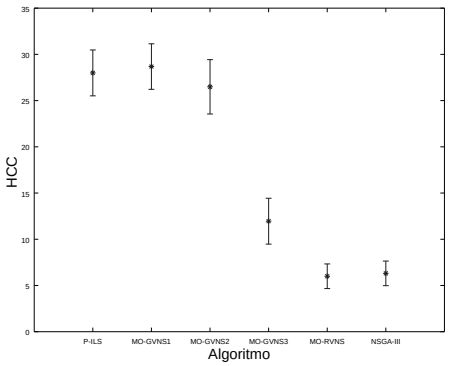
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

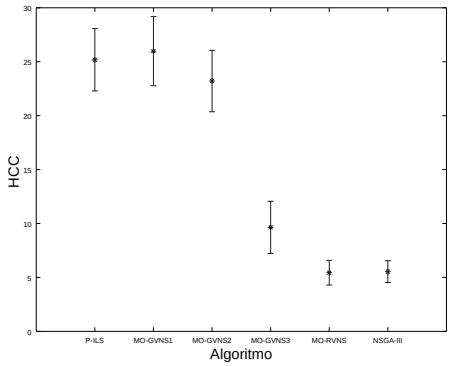


(c) $n = 100, m = 8, m_i = 2$

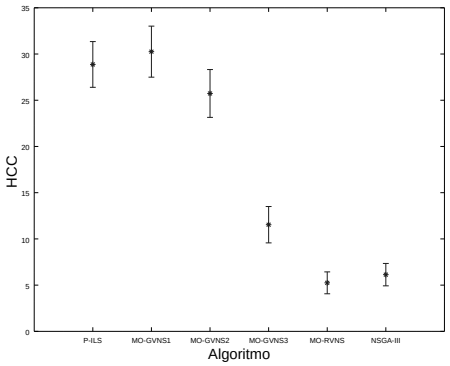


(d) $n = 100, m = 8, m_i = 4$

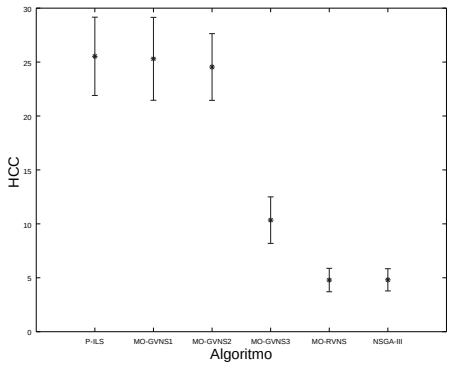
Figura I.18: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 100$)



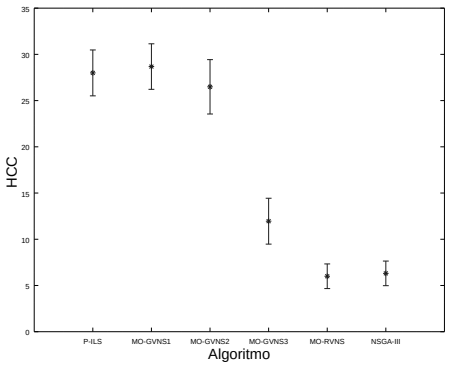
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

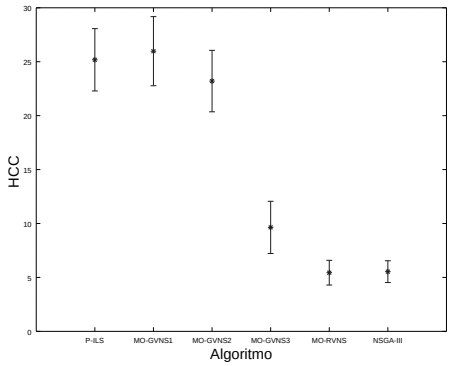


(c) $n = 150, m = 8, m_i = 2$

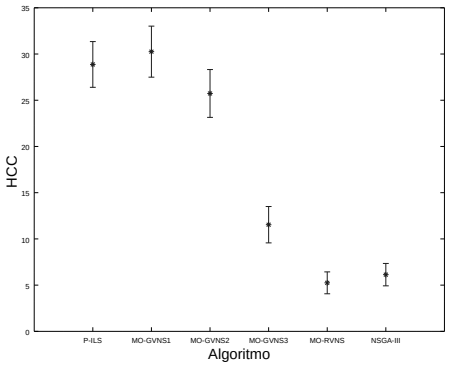


(d) $n = 150, m = 8, m_i = 4$

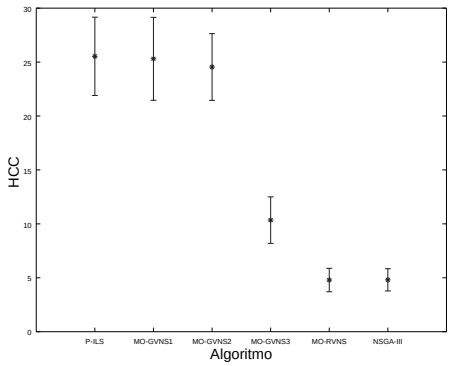
Figura I.19: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 150$)



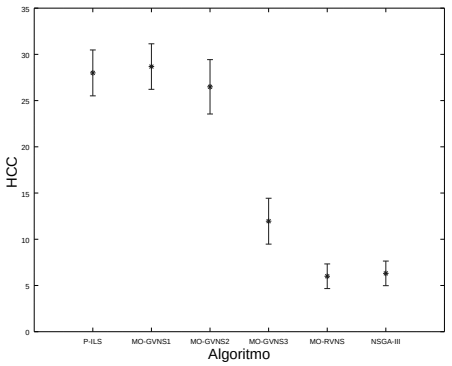
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



(c) $n = 200, m = 8, m_i = 2$



(d) $n = 200, m = 8, m_i = 4$

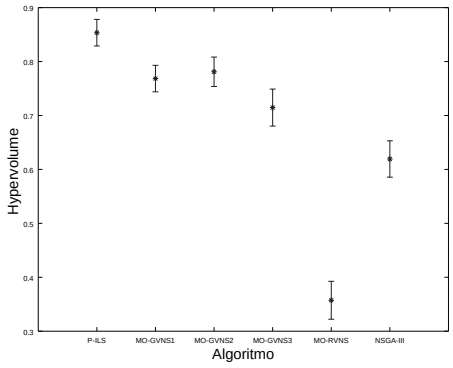
Figura I.20: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 200$)

Anexo II

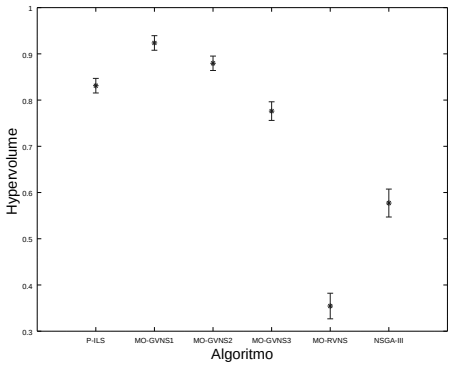
Análise gráfica da solução do problema multiobjetivo com cinco objetivos

II.1 Análise do problema com agregação e redução para três objetivos

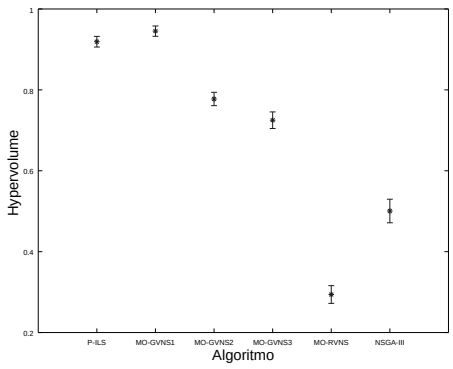
II.1.1 Análise da métrica *Hypervolume*



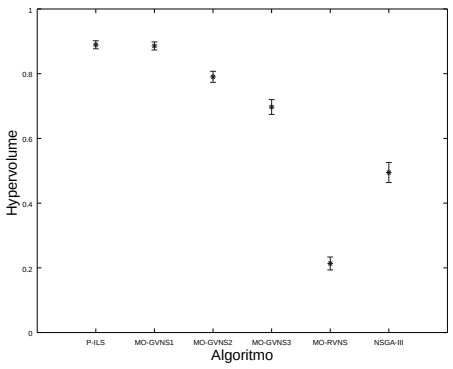
(a) $n = 5$



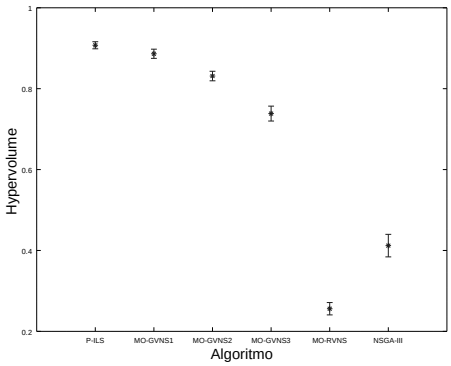
(b) $n = 7$



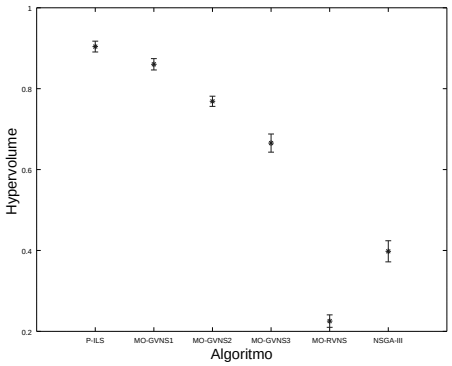
(c) $n = 9$



(d) $n = 11$

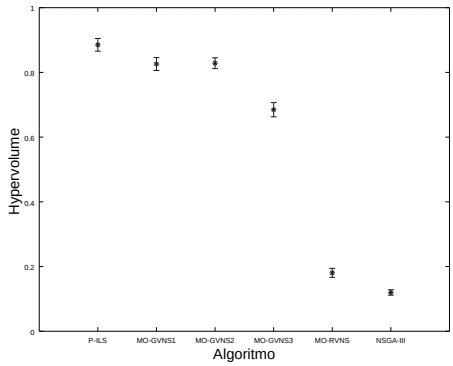


(e) $n = 13$

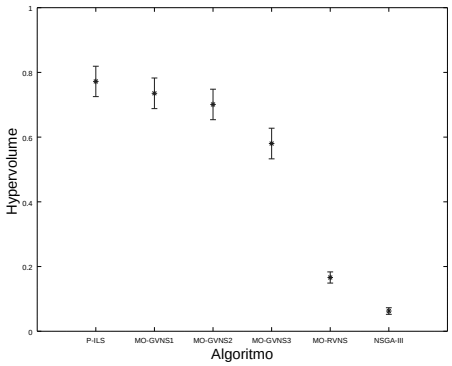


(f) $n = 15$

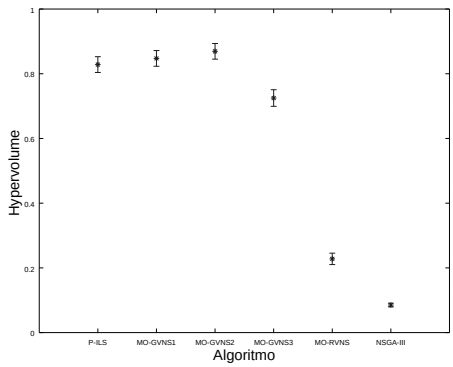
Figura II.1: Intervalo de Confiança na métrica *Hypervolume* instâncias pequenas. Três objetivos agregados.



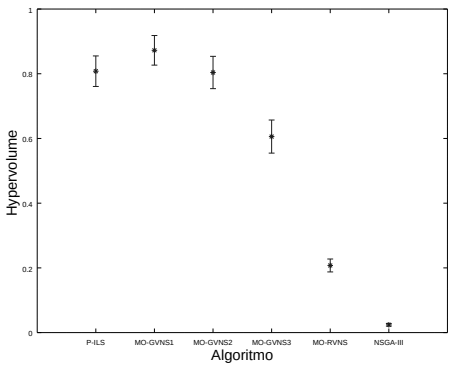
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

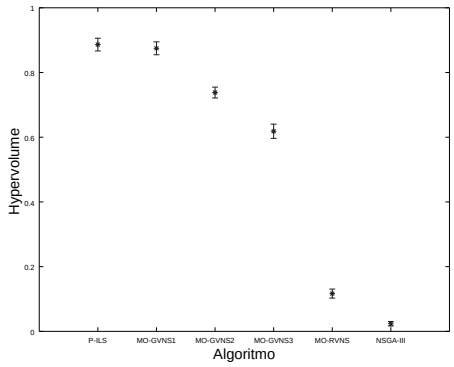


(c) $n = 50, m = 8, m_i = 2$

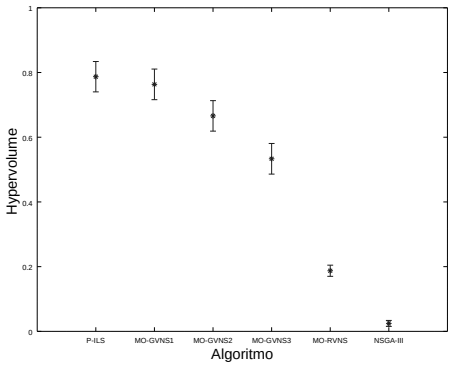


(d) $n = 50, m = 8, m_i = 4$

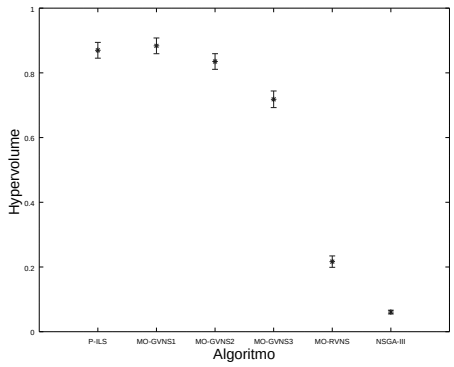
Figura II.2: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 50$). Três objetivos agregados.



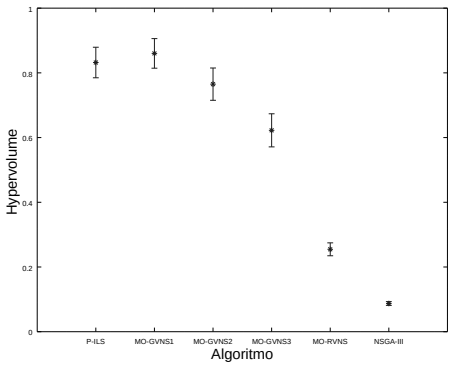
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

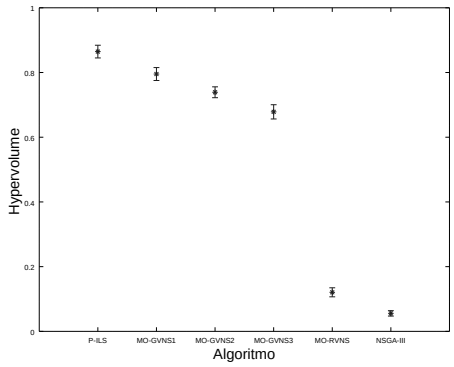


(c) $n = 100, m = 8, m_i = 2$

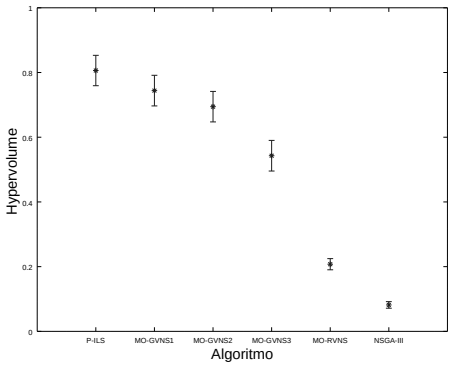


(d) $n = 100, m = 8, m_i = 4$

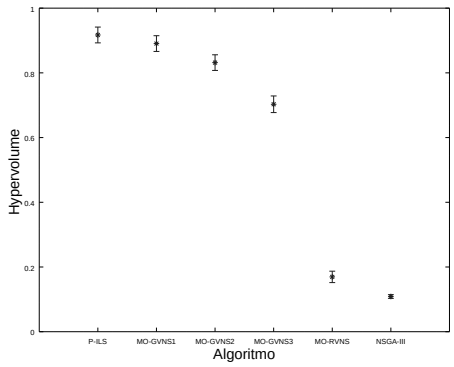
Figura II.3: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 100$). Três objetivos agregados.



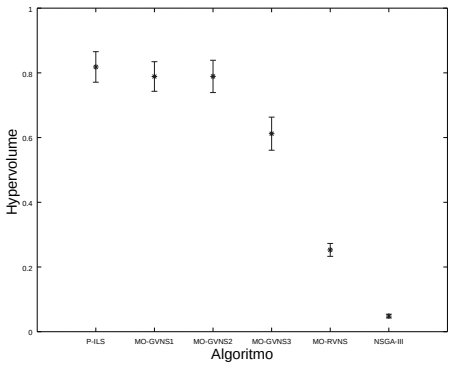
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

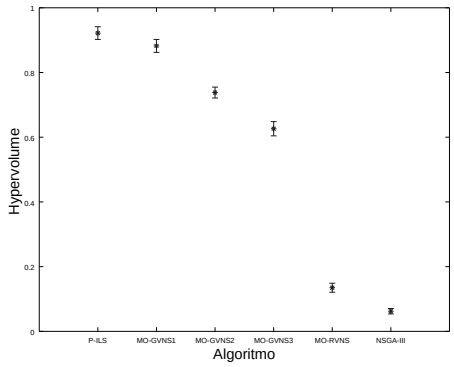


(c) $n = 150, m = 8, m_i = 2$

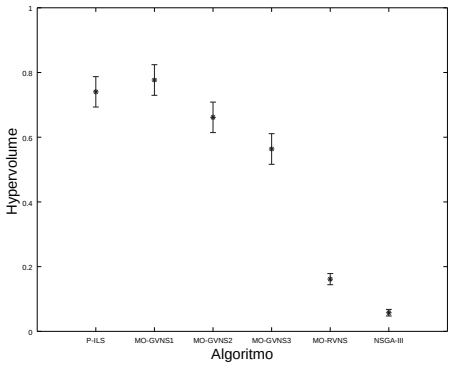


(d) $n = 150, m = 8, m_i = 4$

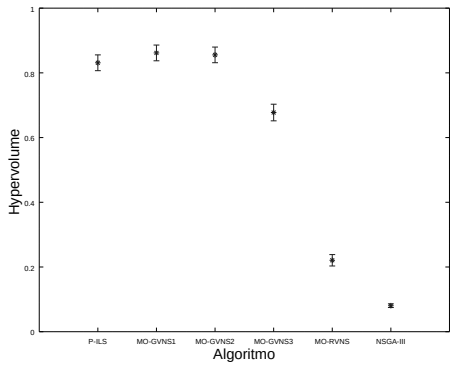
Figura II.4: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 150$). Três objetivos agregados.



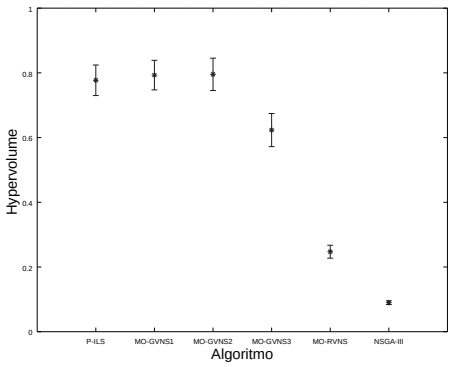
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



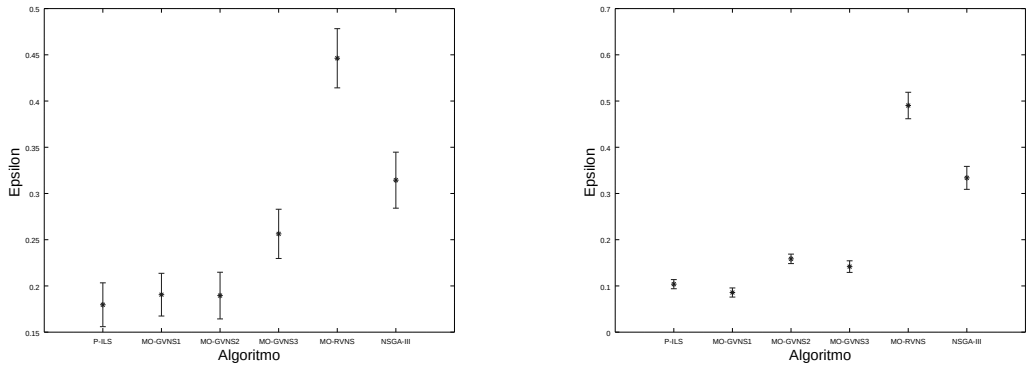
(c) $n = 200, m = 8, m_i = 2$



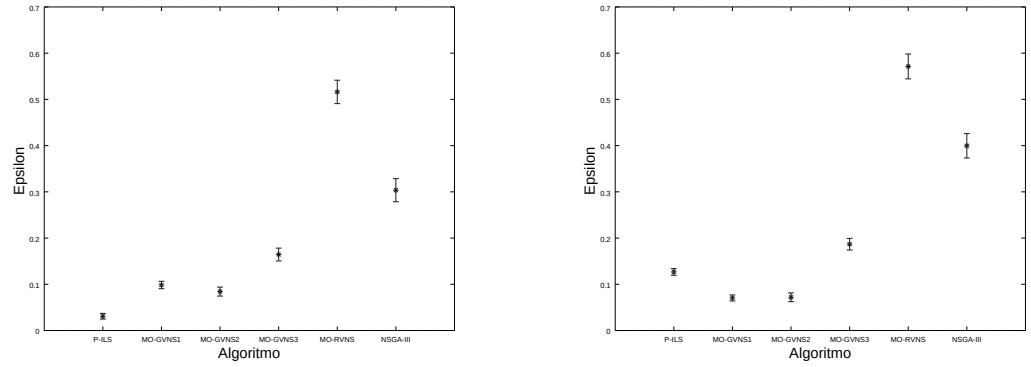
(d) $n = 200, m = 8, m_i = 4$

Figura II.5: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 200$). Três objetivos agregados.

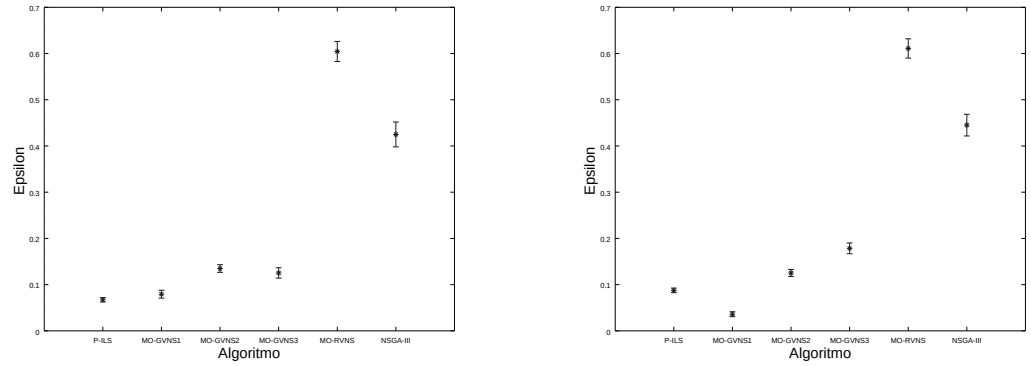
II.1.2 Análise da métrica *Epsilon*



(a) $n = 5$ (b) $n = 7$

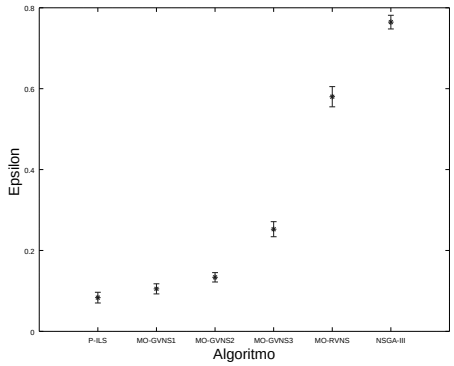


(c) $n = 9$ (d) $n = 11$

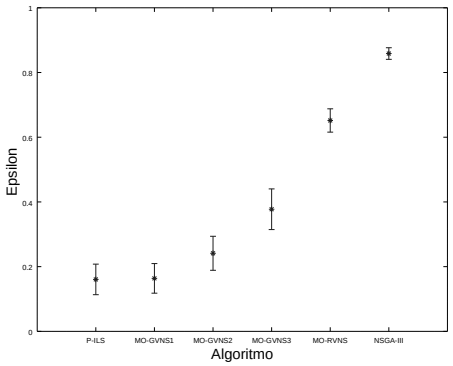


(e) $n = 13$ (f) $n = 15$

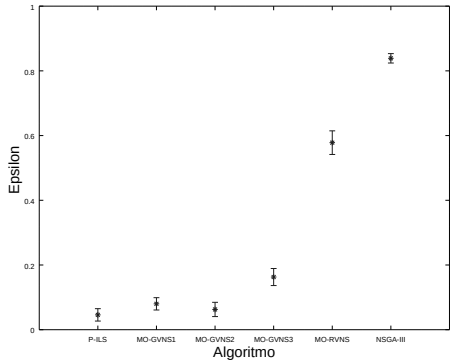
Figura II.6: Intervalo de Confiança na métrica *Epsilon*: instâncias pequenas. Três objetivos agregados.



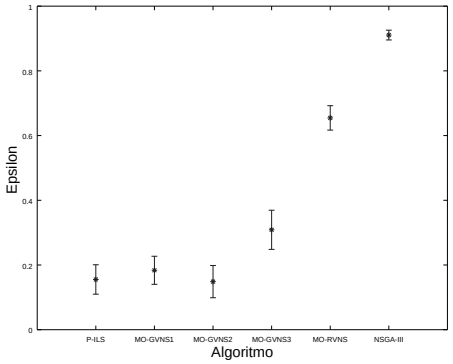
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

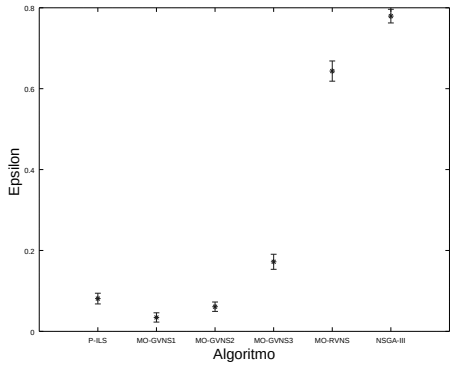


(c) $n = 50, m = 8, m_i = 2$

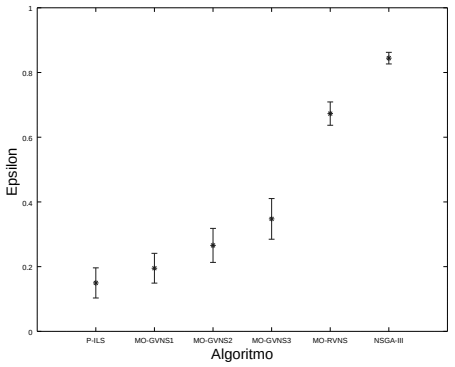


(d) $n = 50, m = 8, m_i = 4$

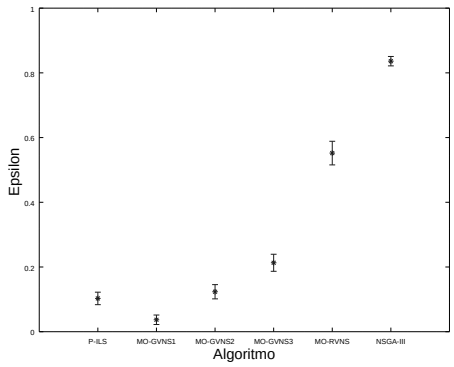
Figura II.7: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 50$). Três objetivos agregados.



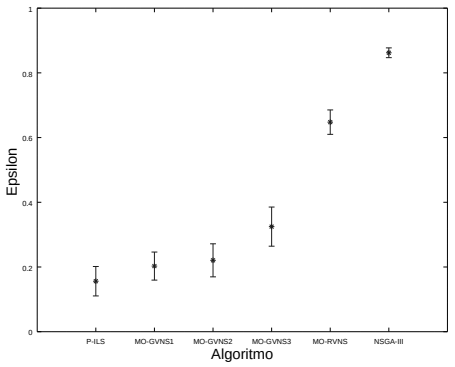
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

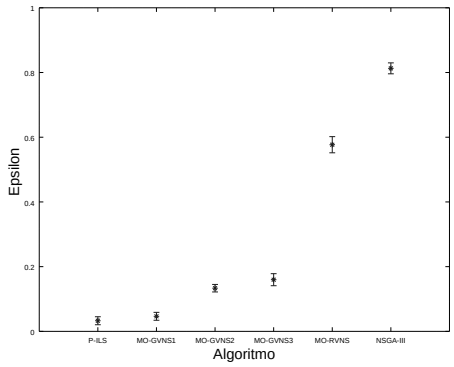


(c) $n = 100, m = 8, m_i = 2$

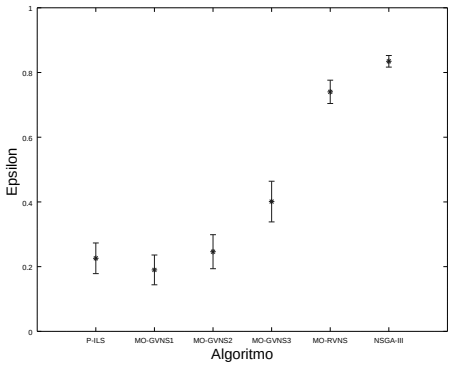


(d) $n = 100, m = 8, m_i = 4$

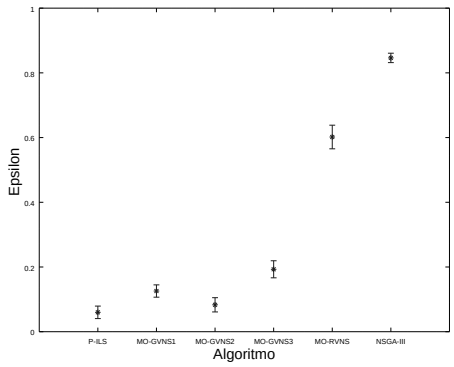
Figura II.8: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 100$). Três objetivos agregados.



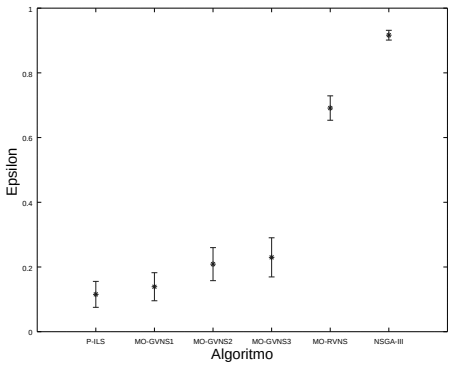
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

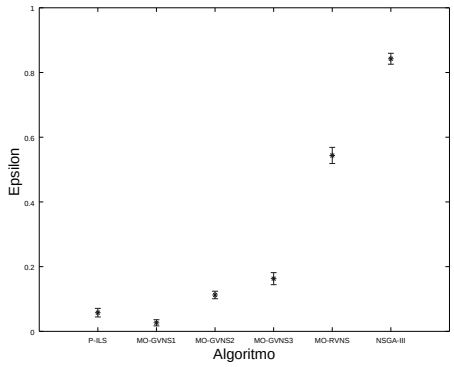


(c) $n = 150, m = 8, m_i = 2$

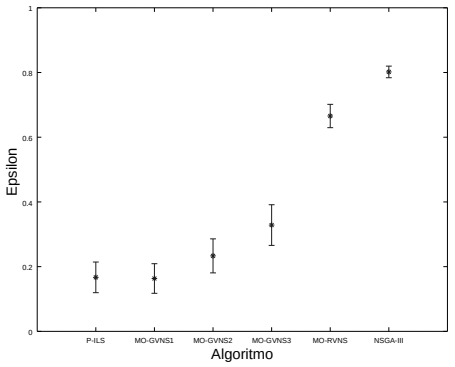


(d) $n = 150, m = 8, m_i = 4$

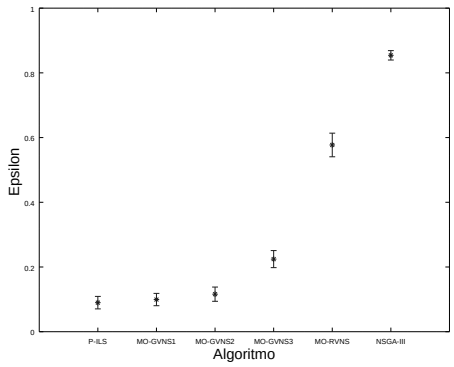
Figura II.9: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 150$). Três objetivos agregados.



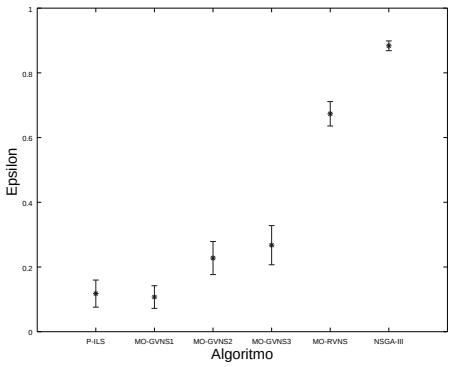
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



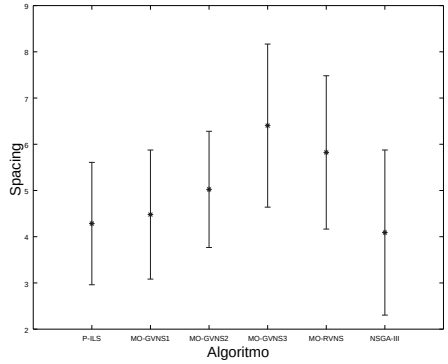
(c) $n = 200, m = 8, m_i = 2$



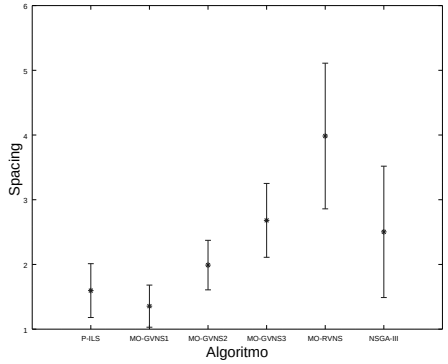
(d) $n = 200, m = 8, m_i = 4$

Figura II.10: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 200$). Três objetivos agregados.

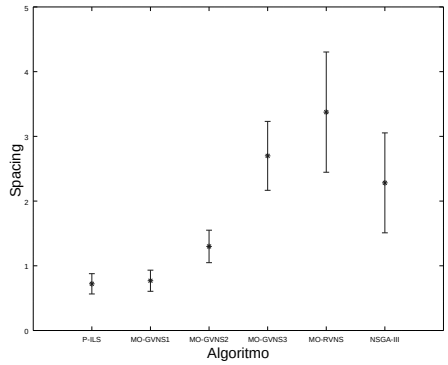
II.1.3 Análise da métrica *Spacing*



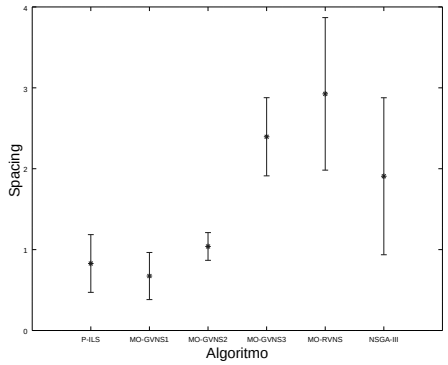
(a) $n = 5$



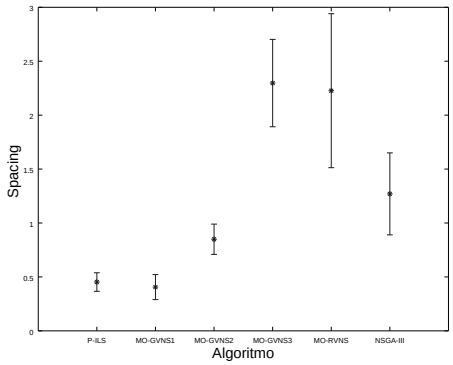
(b) $n = 7$



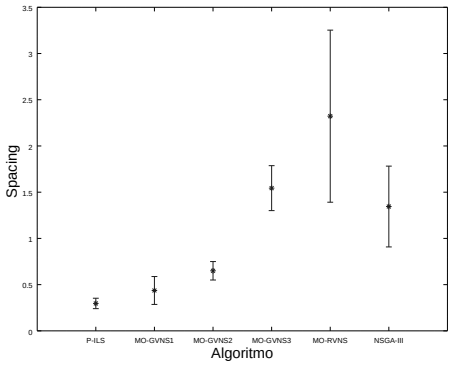
(c) $n = 9$



(d) $n = 11$

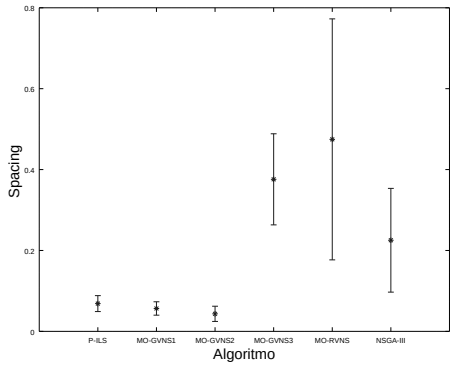


(e) $n = 13$

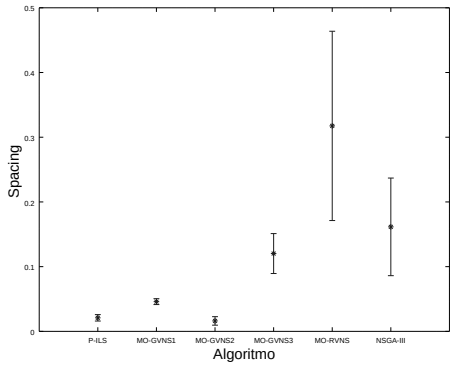


(f) $n = 15$

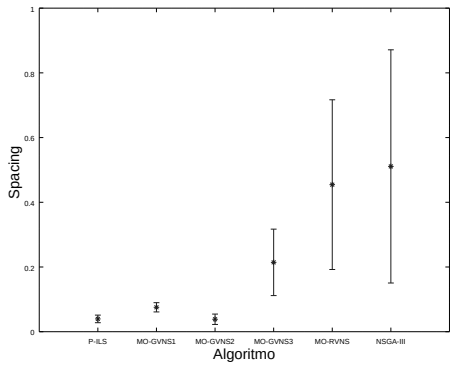
Figura II.11: Intervalo de Confiança na métrica *Spacing*: instâncias pequenas. Três objetivos agregados.



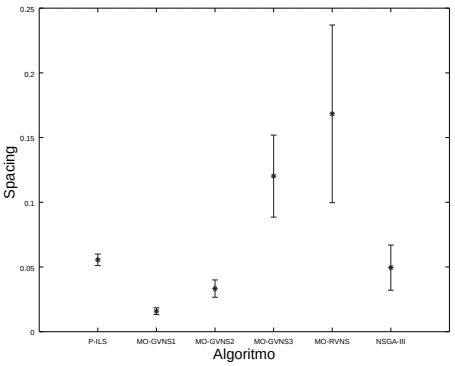
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

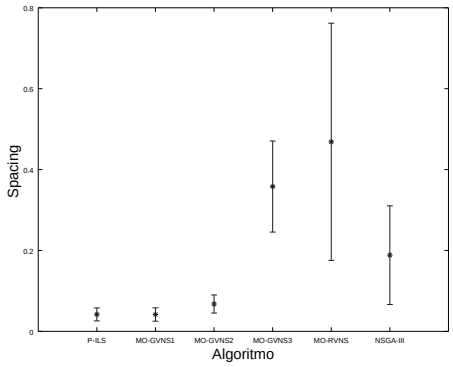


(c) $n = 50, m = 8, m_i = 2$

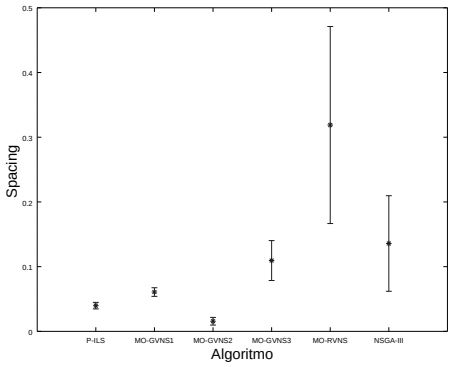


(d) $n = 50, m = 8, m_i = 4$

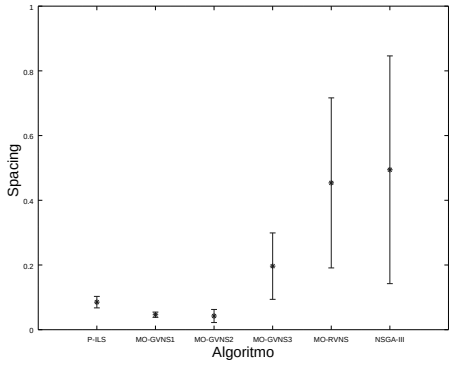
Figura II.12: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 50$). Três objetivos agregados.



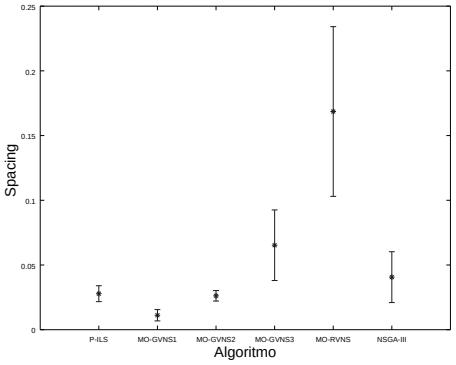
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

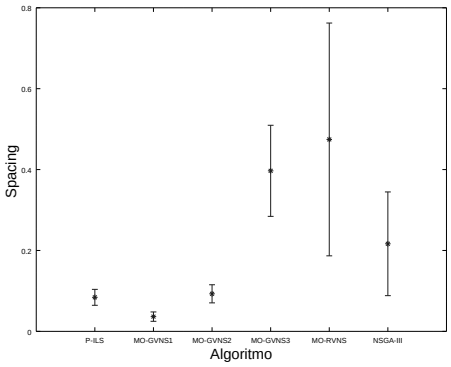


(c) $n = 100, m = 8, m_i = 2$

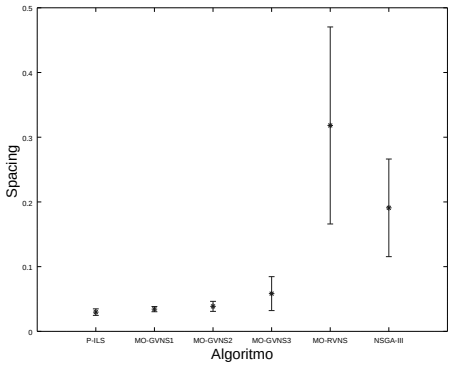


(d) $n = 100, m = 8, m_i = 4$

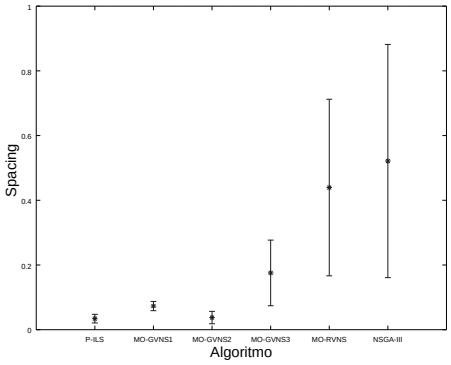
Figura II.13: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 100$). Três objetivos agregados.



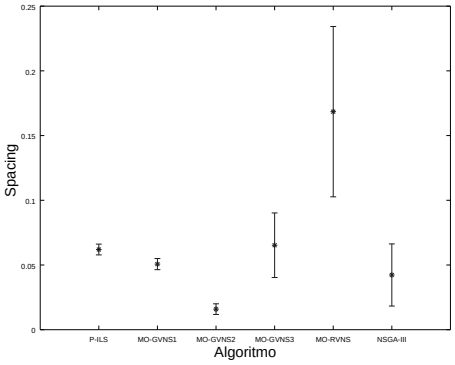
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

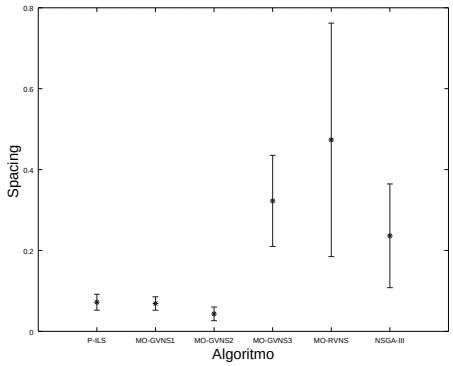


(c) $n = 150, m = 8, m_i = 2$

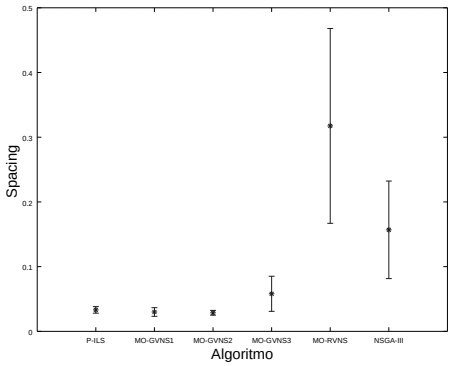


(d) $n = 150, m = 8, m_i = 4$

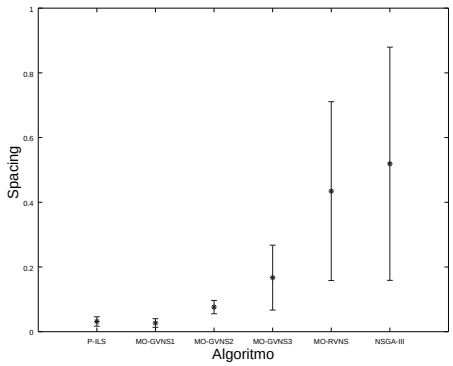
Figura II.14: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 150$). Três objetivos agregados.



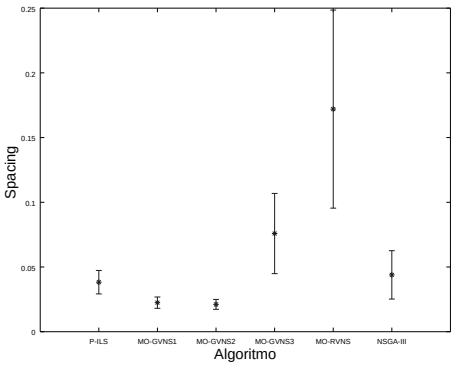
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



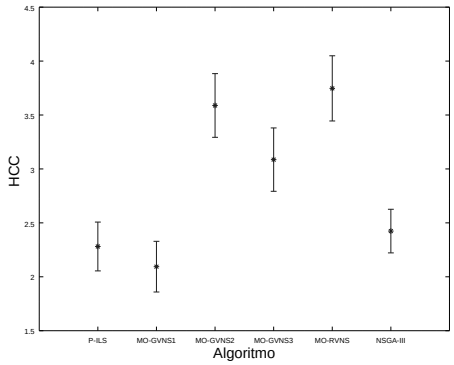
(c) $n = 200, m = 8, m_i = 2$



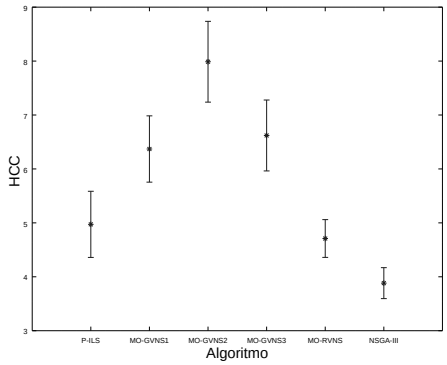
(d) $n = 200, m = 8, m_i = 4$

Figura II.15: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 200$). Três objetivos agregados.

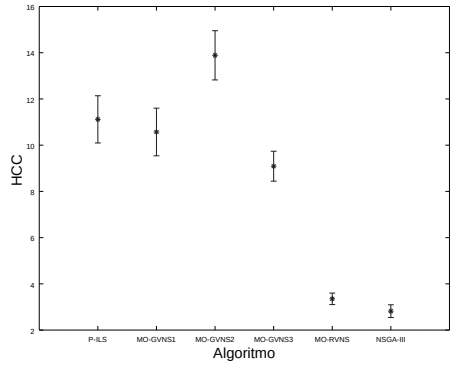
II.1.4 Análise da métrica HCC



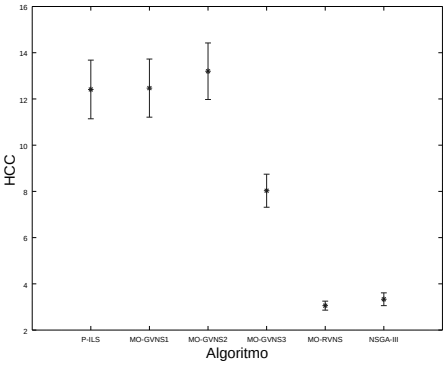
(a) $n = 5$



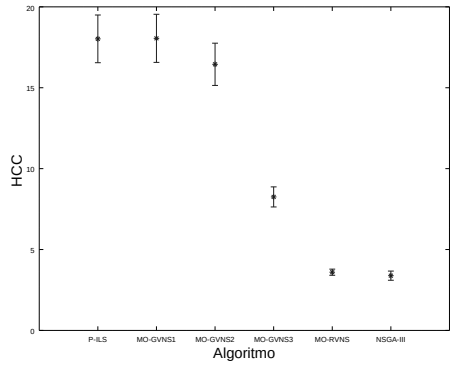
(b) $n = 7$



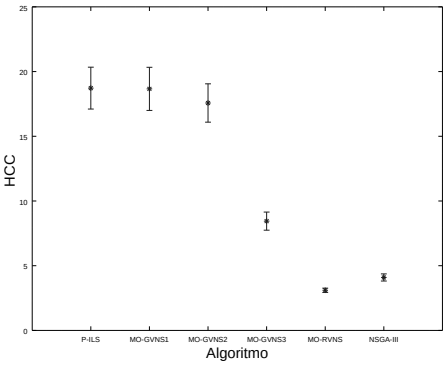
(c) $n = 9$



(d) $n = 11$

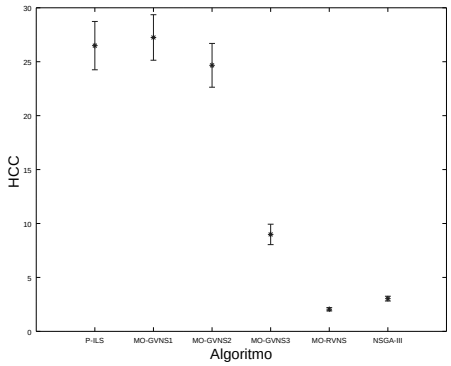


(e) $n = 13$

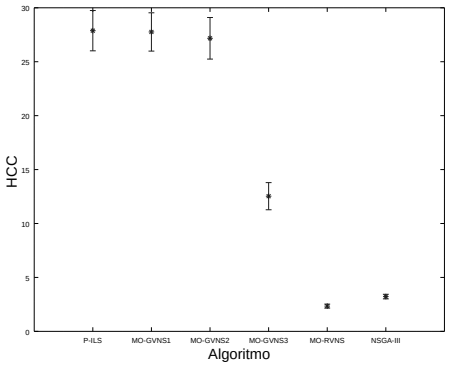


(f) $n = 15$

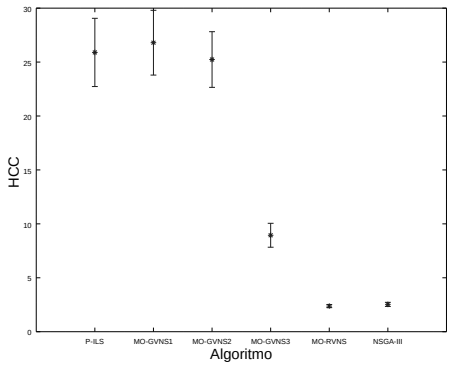
Figura II.16: Intervalo de Confiança na métrica *HCC*: instâncias pequenas. Três objetivos agregados.



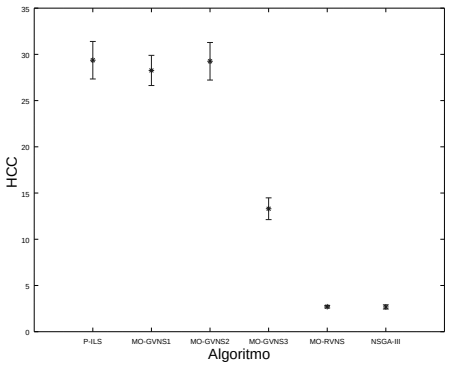
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

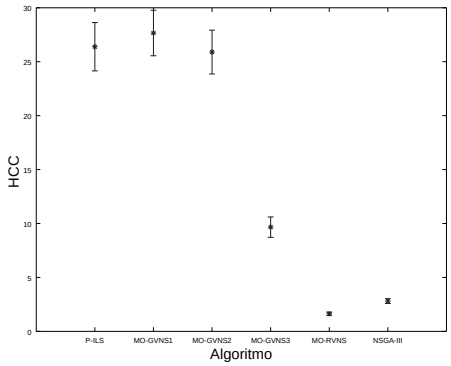


(c) $n = 50, m = 8, m_i = 2$

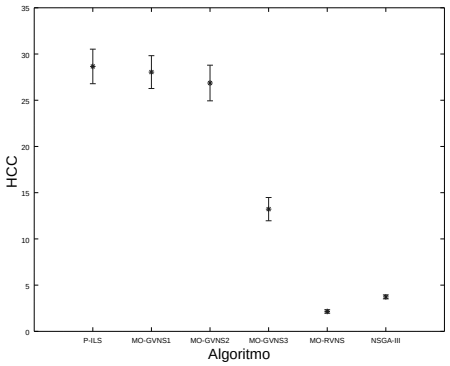


(d) $n = 50, m = 8, m_i = 4$

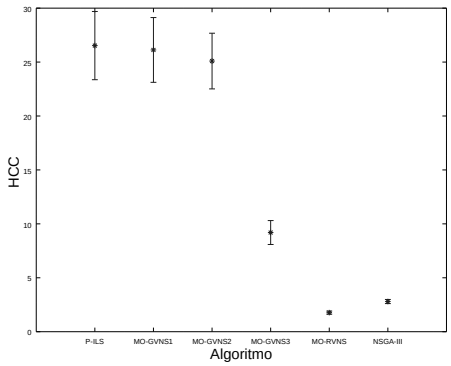
Figura II.17: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 50$). Três objetivos agregados.



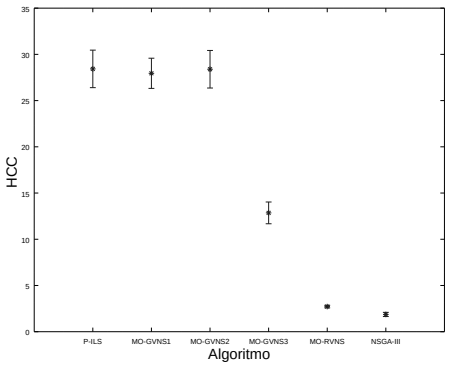
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

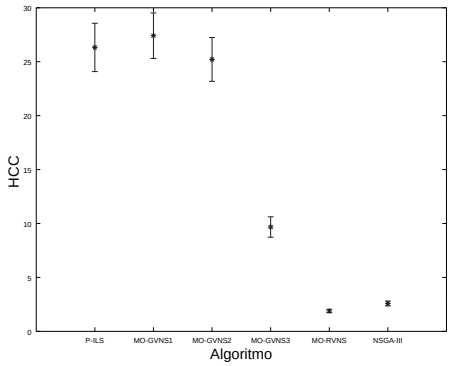


(c) $n = 100, m = 8, m_i = 2$

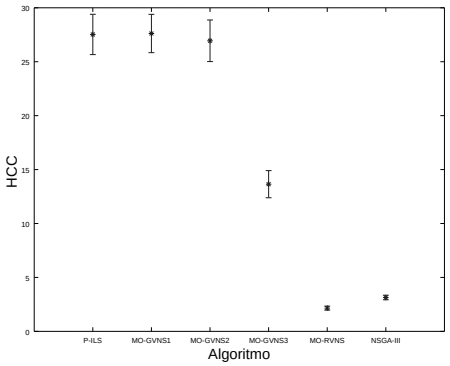


(d) $n = 100, m = 8, m_i = 4$

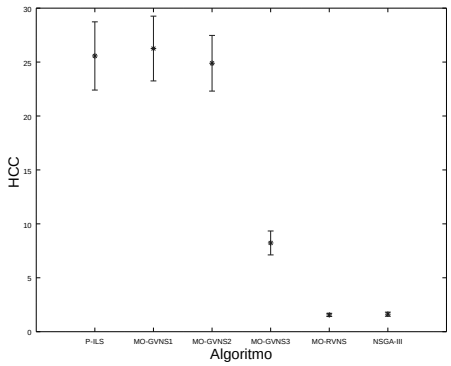
Figura II.18: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 100$). Três objetivos agregados.



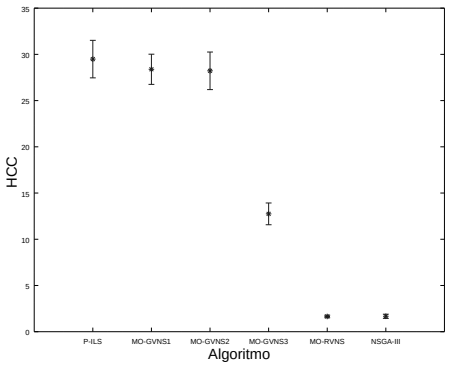
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

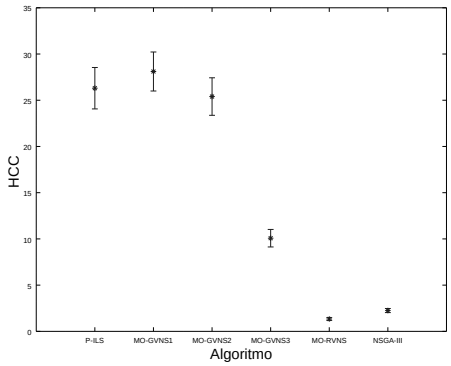


(c) $n = 150, m = 8, m_i = 2$

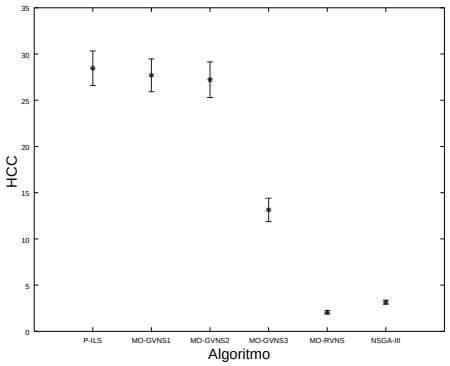


(d) $n = 150, m = 8, m_i = 4$

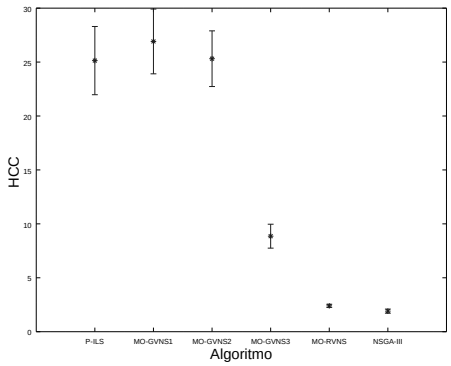
Figura II.19: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 150$). Três objetivos agregados.



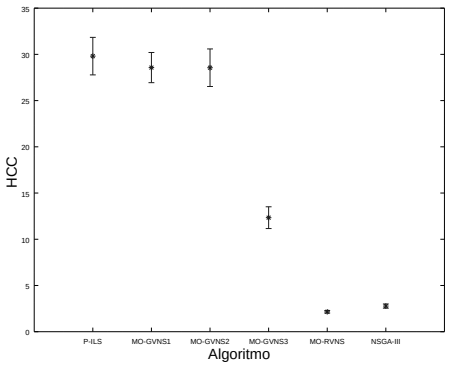
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



(c) $n = 200, m = 8, m_i = 2$

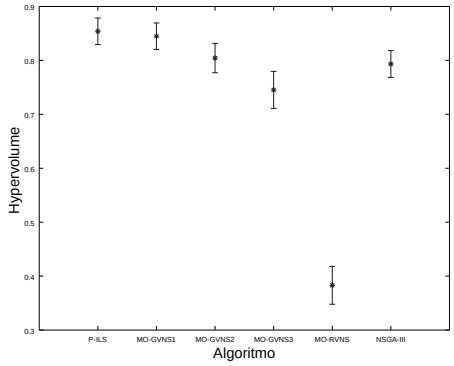


(d) $n = 200, m = 8, m_i = 4$

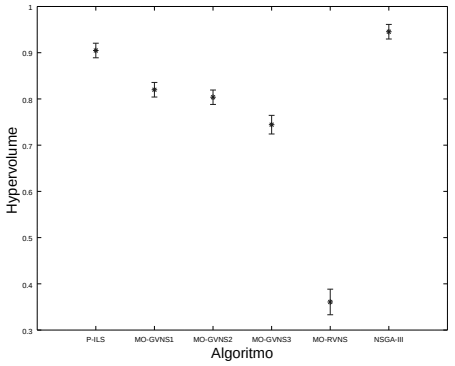
Figura II.20: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 200$). Três objetivos agregados.

II.2 Análise do problema para cinco objetivos simultâneos

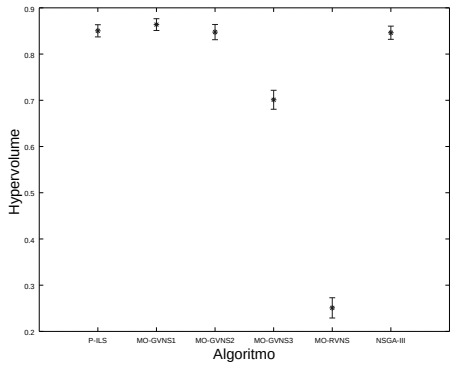
II.2.1 Análise da métrica *Hypervolume*



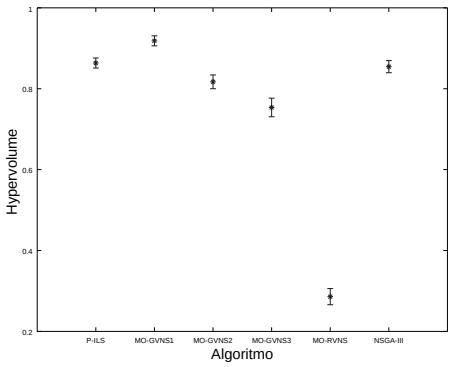
(a) $n = 5$



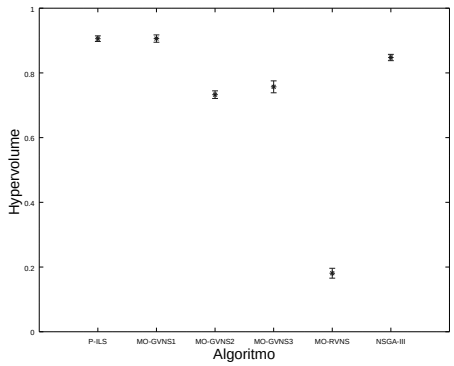
(b) $n = 7$



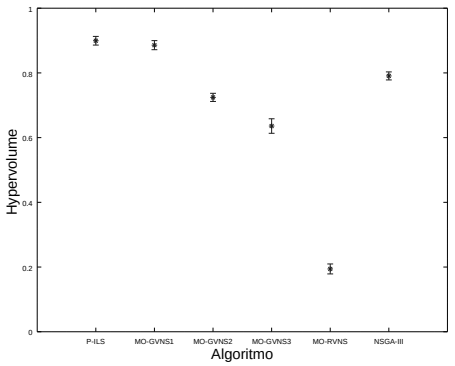
(c) $n = 9$



(d) $n = 11$

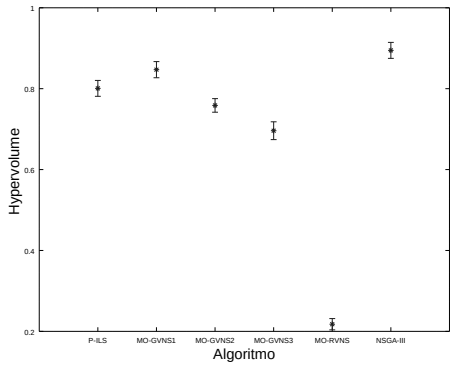


(e) $n = 13$

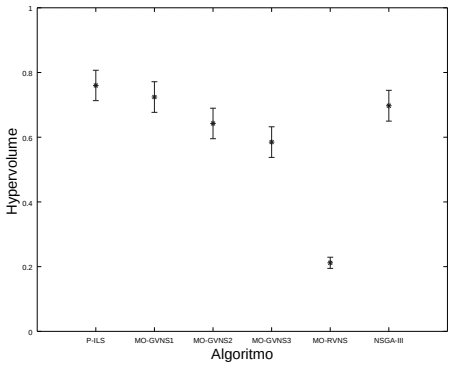


(f) $n = 15$

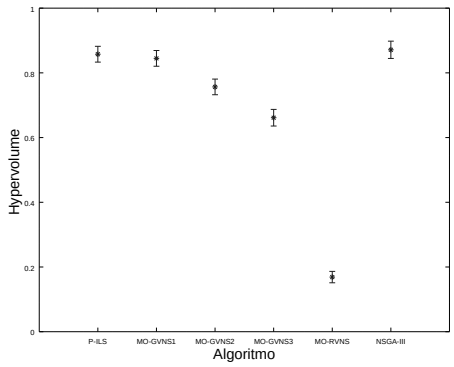
Figura II.21: Intervalo de Confiança na métrica *Hypervolume* instâncias pequenas. Cinco objetivos simultâneos.



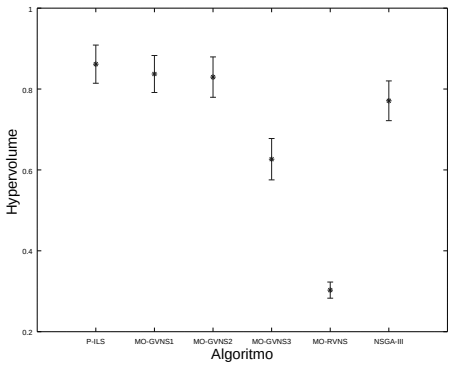
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

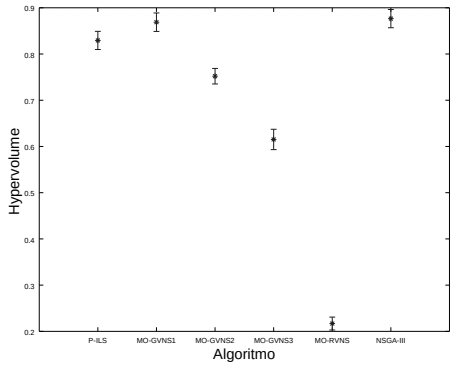


(c) $n = 50, m = 8, m_i = 2$

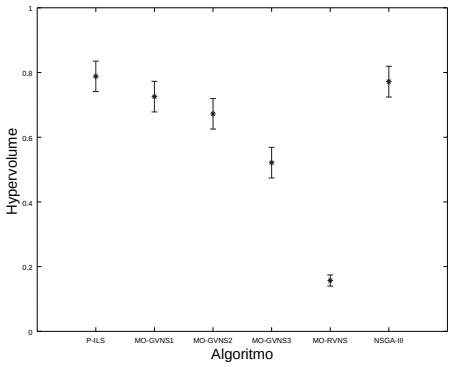


(d) $n = 50, m = 8, m_i = 4$

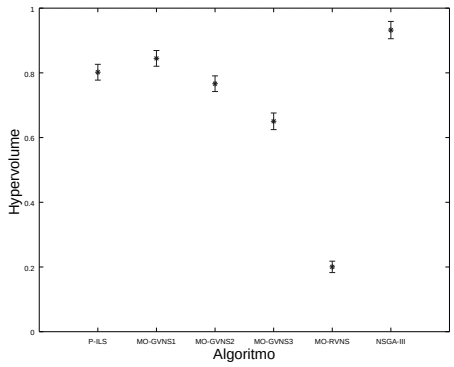
Figura II.22: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 50$). Cinco objetivos simultâneos.



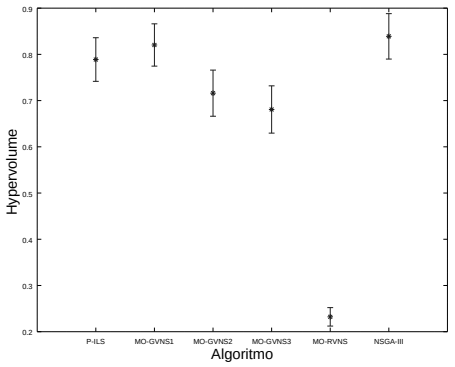
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

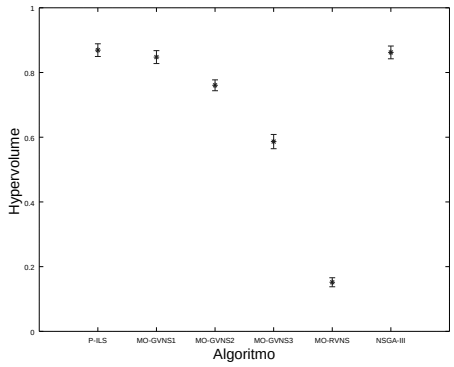


(c) $n = 100, m = 8, m_i = 2$

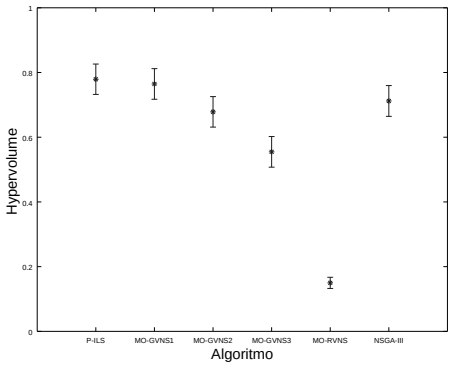


(d) $n = 100, m = 8, m_i = 4$

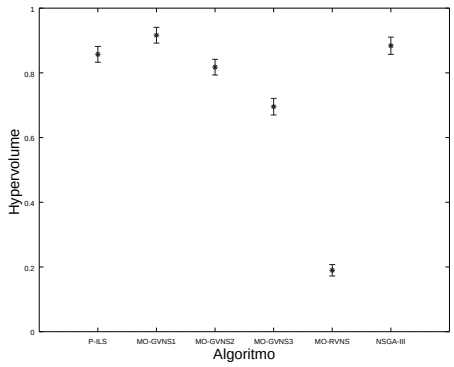
Figura II.23: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 100$). Cinco objetivos simultâneos.



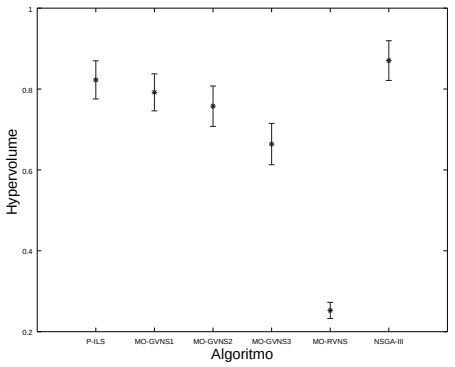
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

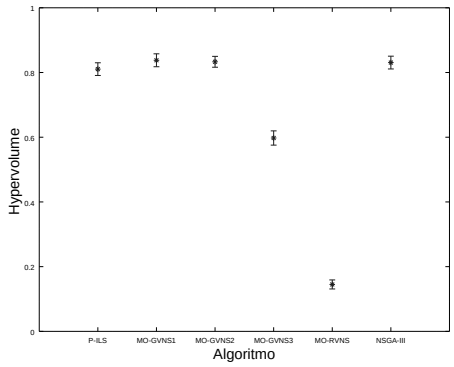


(c) $n = 150, m = 8, m_i = 2$

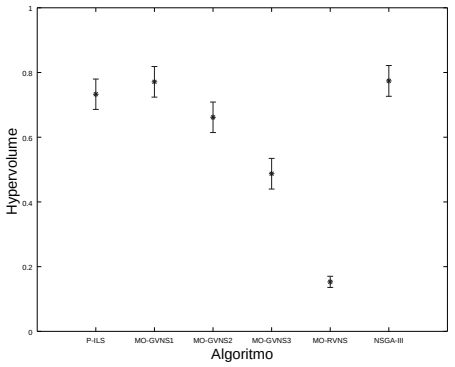


(d) $n = 150, m = 8, m_i = 4$

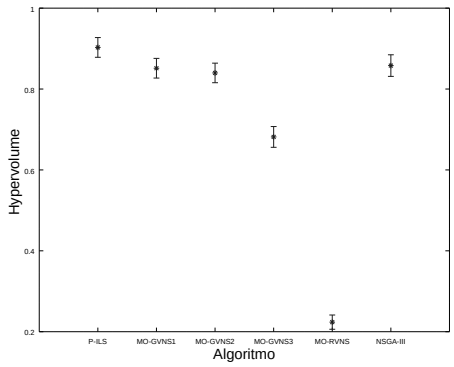
Figura II.24: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 150$). Cinco objetivos simultâneos.



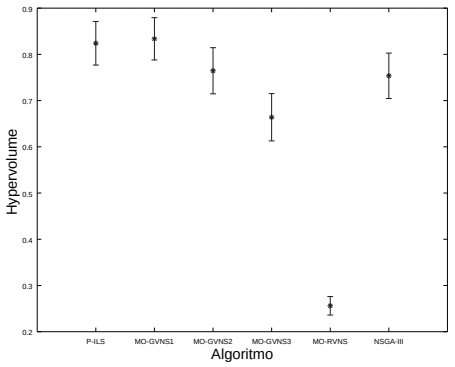
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



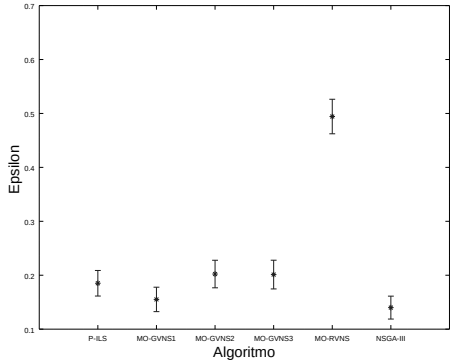
(c) $n = 200, m = 8, m_i = 2$



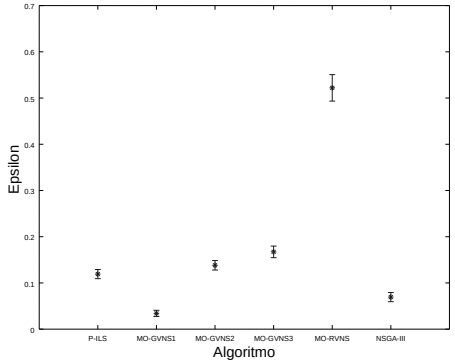
(d) $n = 200, m = 8, m_i = 4$

Figura II.25: Intervalo de Confiança na métrica *Hypervolume*: instâncias grandes ($n = 200$). Cinco objetivos simultâneos.

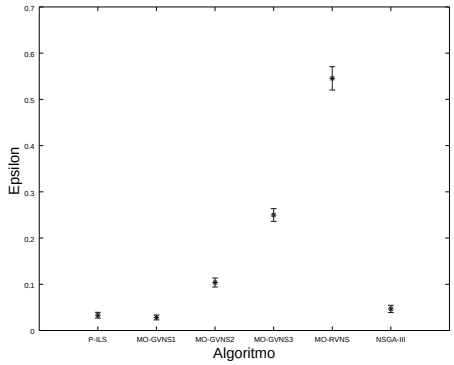
II.2.2 Análise da métrica *Epsilon*



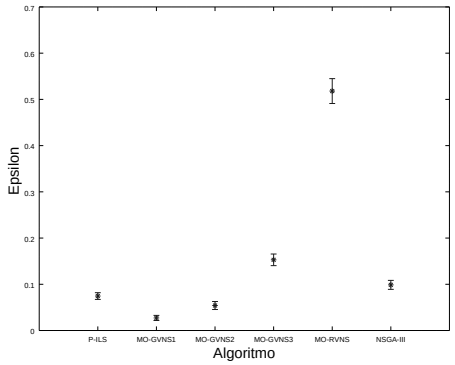
(a) $n = 5$



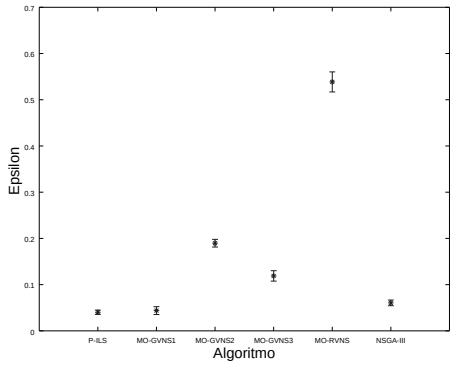
(b) $n = 7$



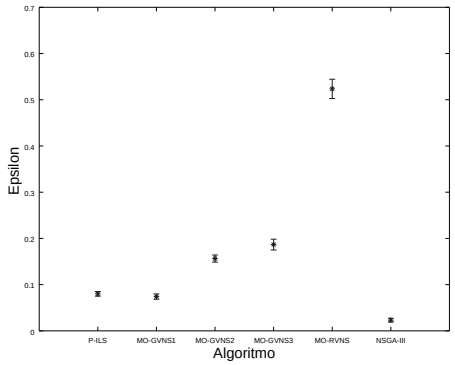
(c) $n = 9$



(d) $n = 11$

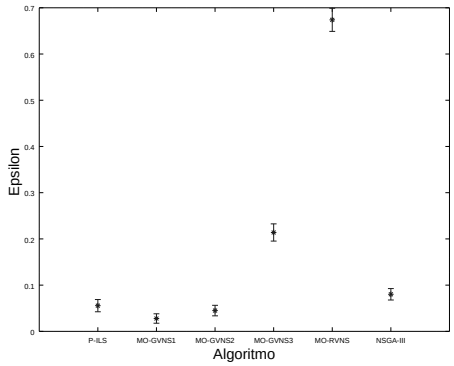


(e) $n = 13$

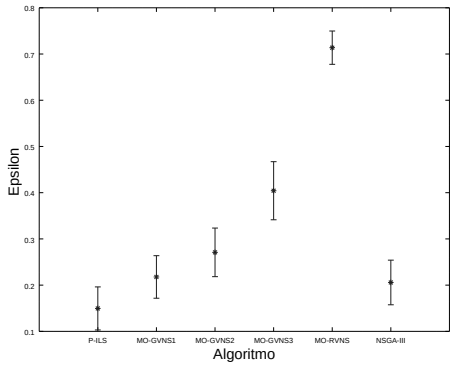


(f) $n = 15$

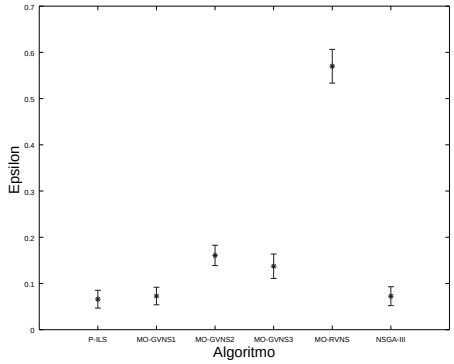
Figura II.26: Intervalo de Confiança na métrica *Epsilon*: instâncias pequenas. Cinco objetivos simultâneos.



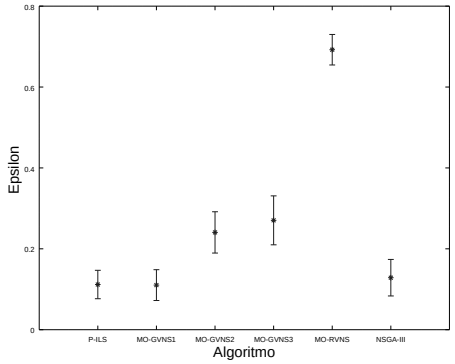
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

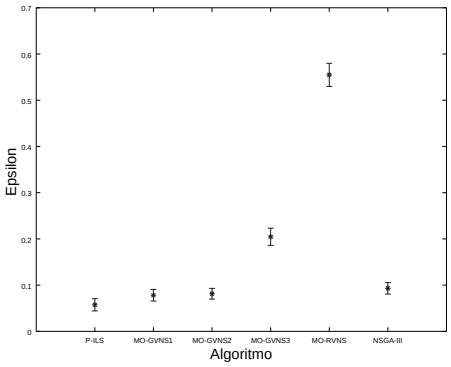


(c) $n = 50, m = 8, m_i = 2$

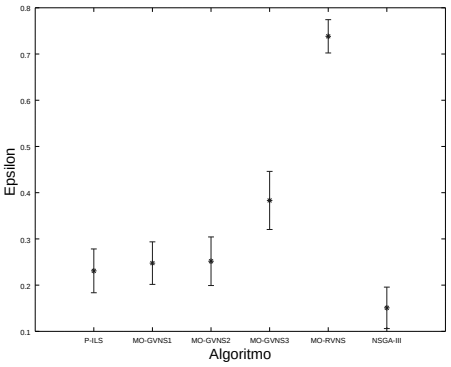


(d) $n = 50, m = 8, m_i = 4$

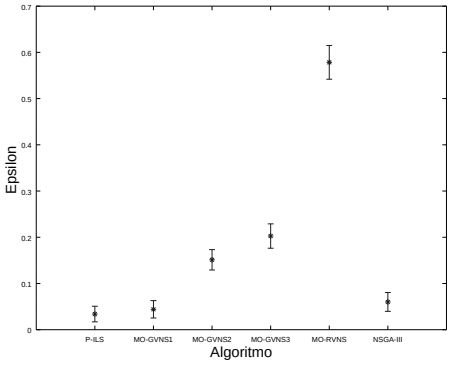
Figura II.27: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 50$). Cinco objetivos simultâneos.



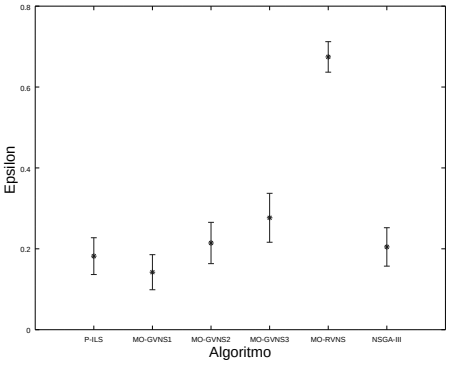
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

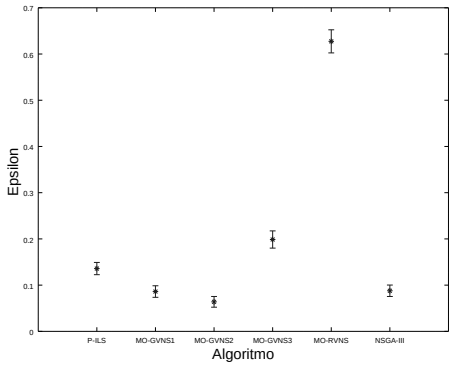


(c) $n = 100, m = 8, m_i = 2$

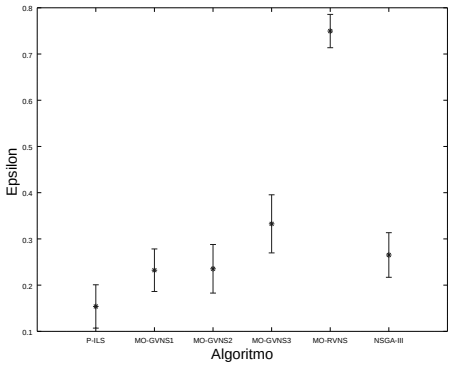


(d) $n = 100, m = 8, m_i = 4$

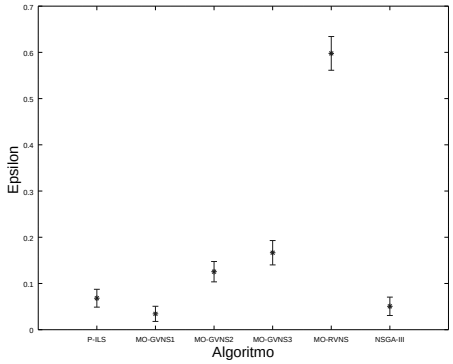
Figura II.28: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 100$). Cinco objetivos simultâneos.



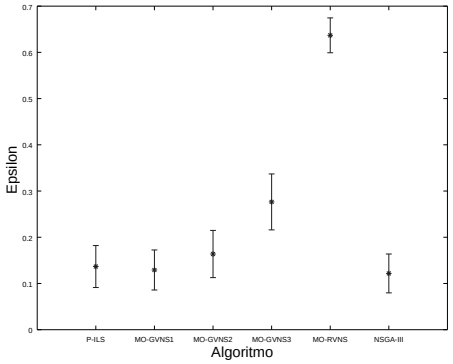
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

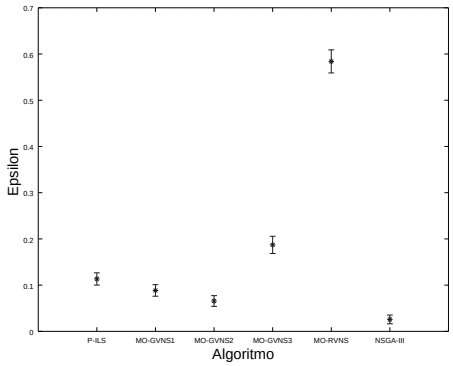


(c) $n = 150, m = 8, m_i = 2$

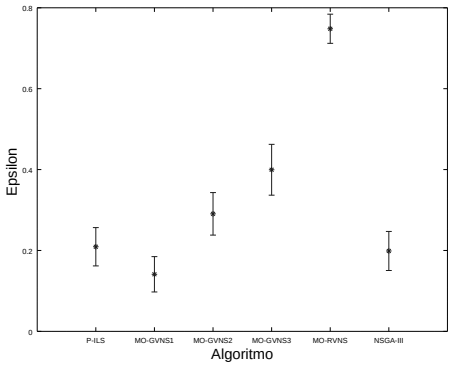


(d) $n = 150, m = 8, m_i = 4$

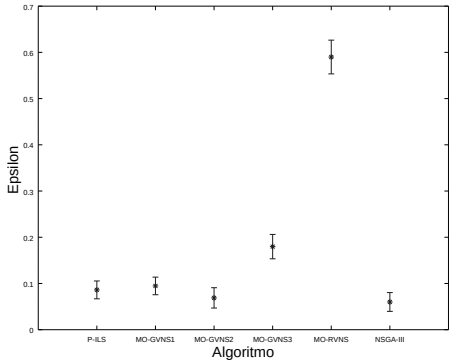
Figura II.29: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 150$). Cinco objetivos simultâneos.



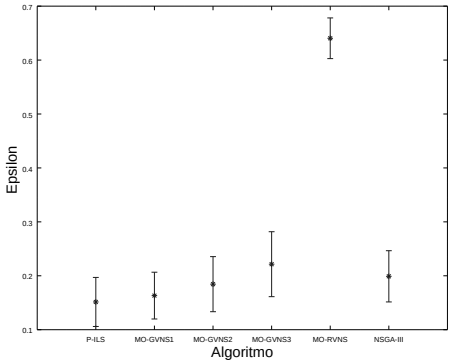
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



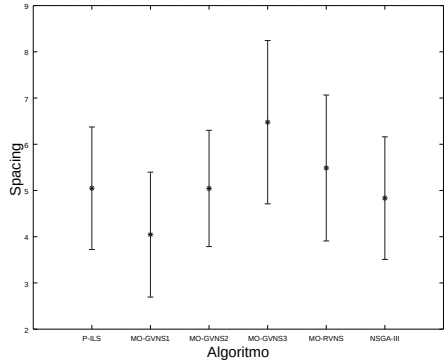
(c) $n = 200, m = 8, m_i = 2$



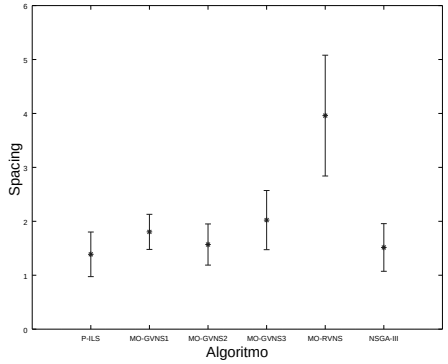
(d) $n = 200, m = 8, m_i = 4$

Figura II.30: Intervalo de Confiança na métrica *Epsilon*: instâncias grandes ($n = 200$). Cinco objetivos simultâneos.

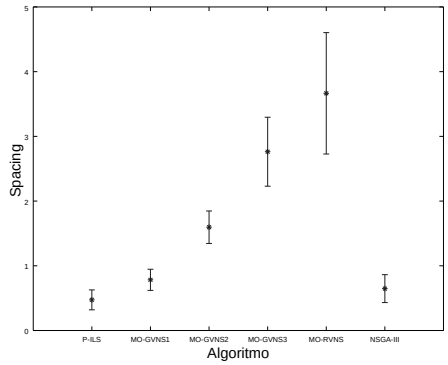
II.2.3 Análise da métrica *Spacing*



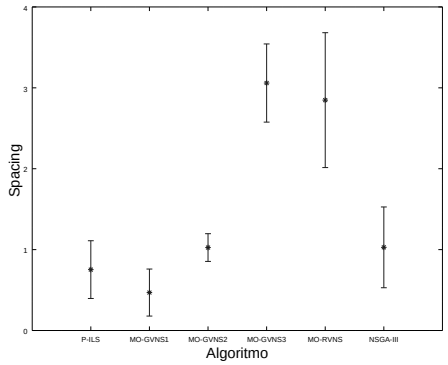
(a) $n = 5$



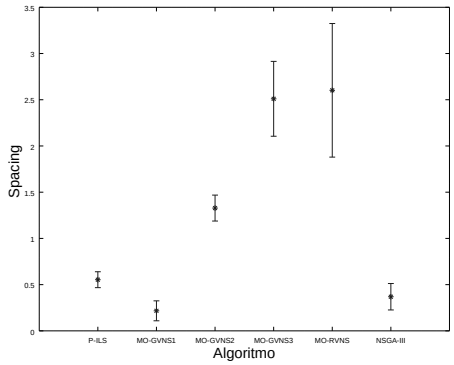
(b) $n = 7$



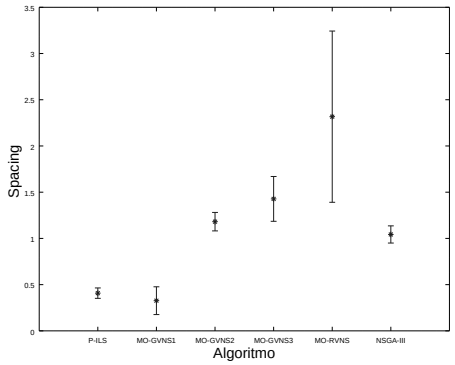
(c) $n = 9$



(d) $n = 11$

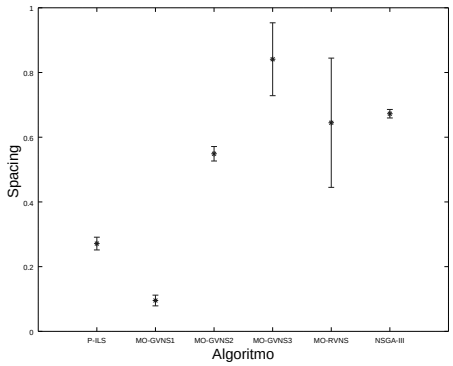


(e) $n = 13$

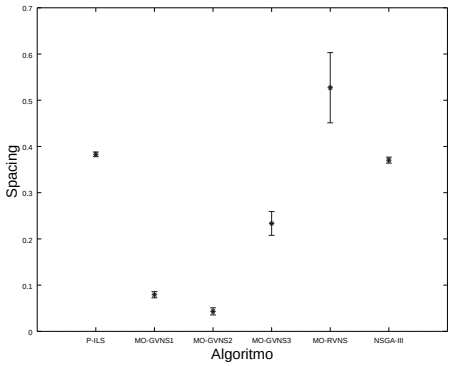


(f) $n = 15$

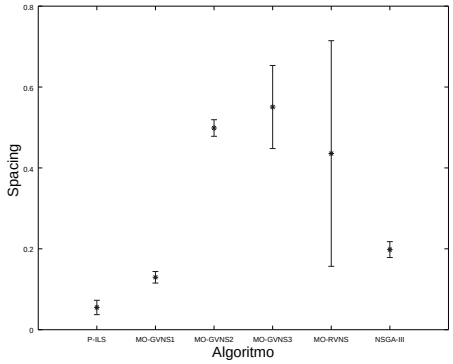
Figura II.31: Intervalo de Confiança na métrica *Spacing*: instâncias pequenas. Cinco objetivos simultâneos.



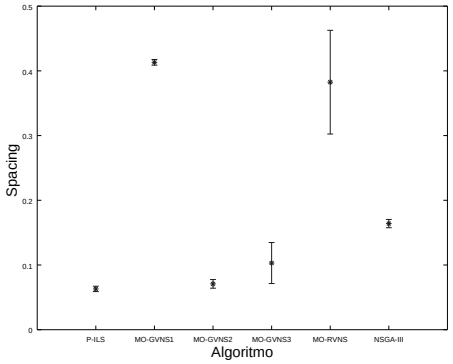
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

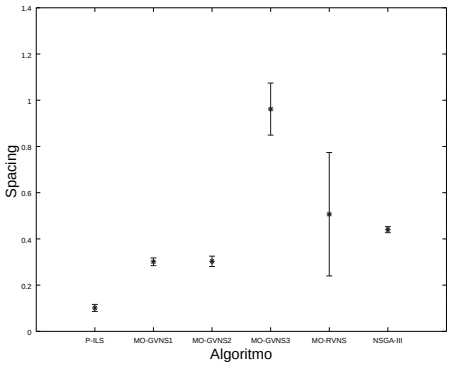


(c) $n = 50, m = 8, m_i = 2$

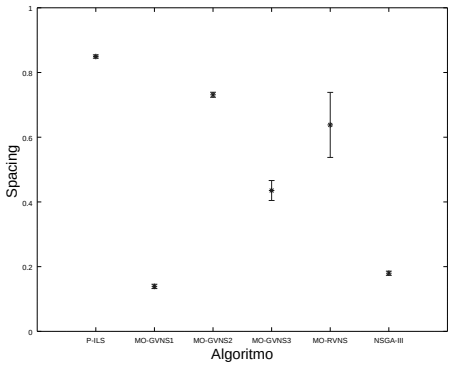


(d) $n = 50, m = 8, m_i = 4$

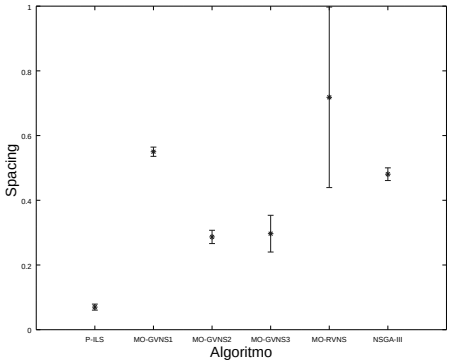
Figura II.32: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 50$). Cinco objetivos simultâneos.



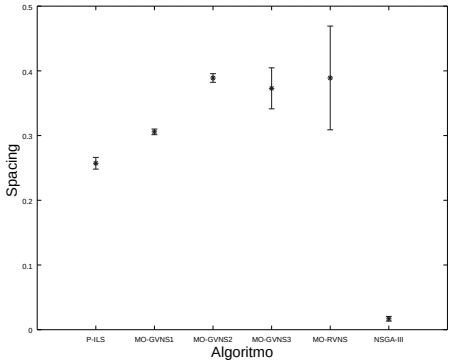
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

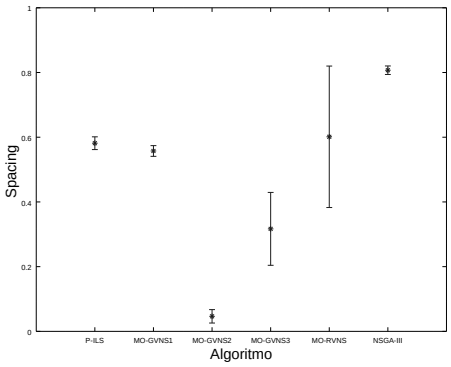


(c) $n = 100, m = 8, m_i = 2$

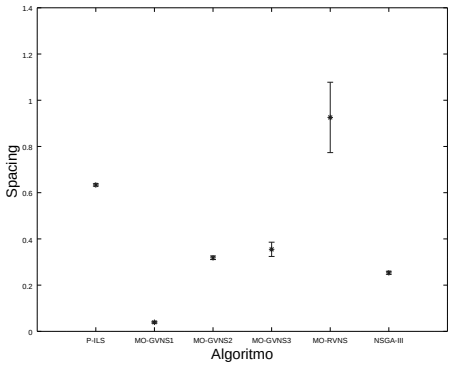


(d) $n = 100, m = 8, m_i = 4$

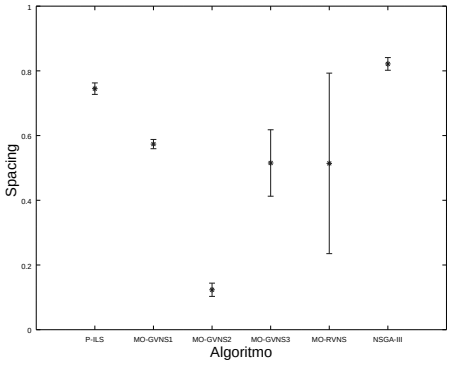
Figura II.33: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 100$). Cinco objetivos simultâneos.



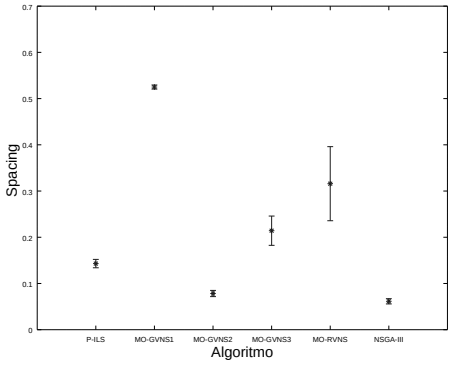
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

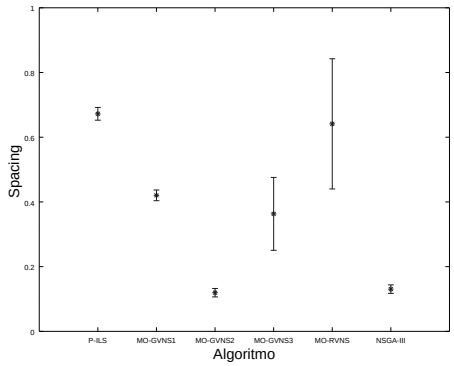


(c) $n = 150, m = 8, m_i = 2$

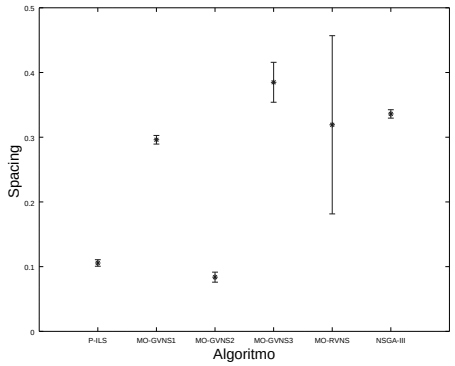


(d) $n = 150, m = 8, m_i = 4$

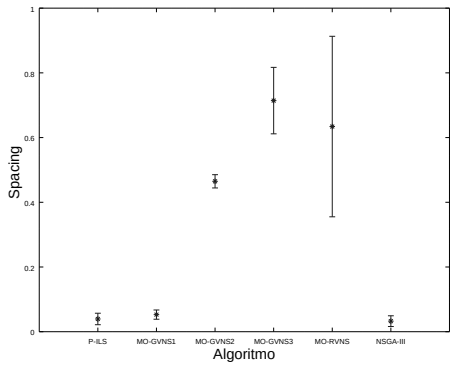
Figura II.34: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 150$). Cinco objetivos simultâneos.



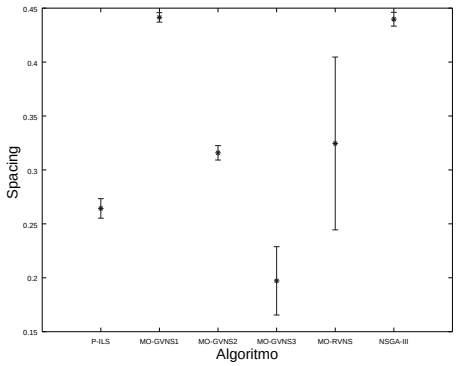
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



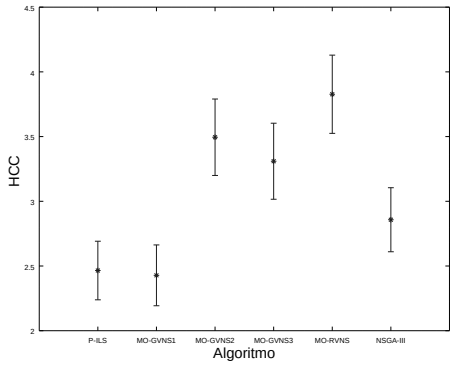
(c) $n = 200, m = 8, m_i = 2$



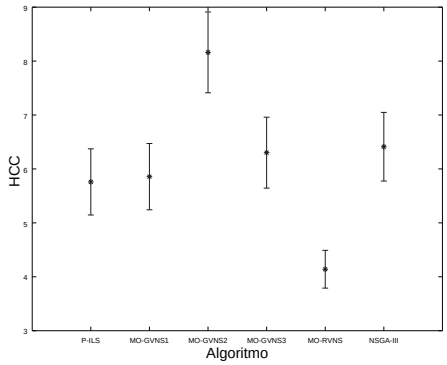
(d) $n = 200, m = 8, m_i = 4$

Figura II.35: Intervalo de Confiança na métrica *Spacing*: instâncias grandes ($n = 200$). Cinco objetivos simultâneos.

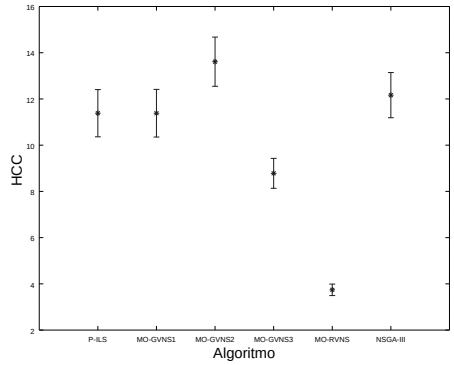
II.2.4 Análise da métrica HCC



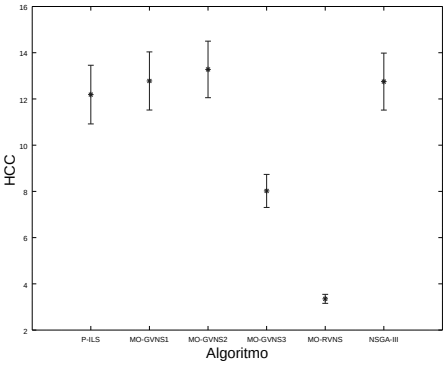
(a) $n = 5$



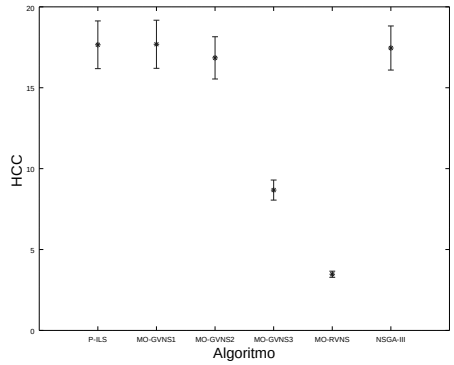
(b) $n = 7$



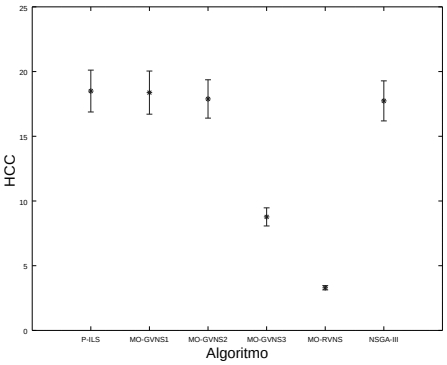
(c) $n = 9$



(d) $n = 11$

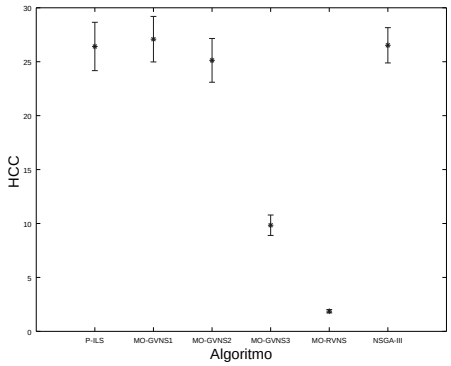


(e) $n = 13$

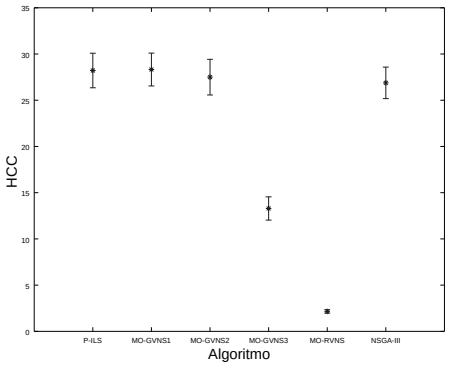


(f) $n = 15$

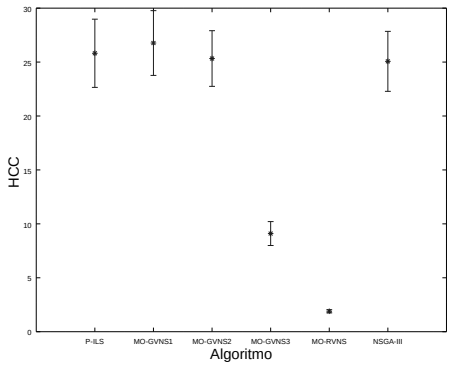
Figura II.36: Intervalo de Confiança na métrica *HCC*: instâncias pequenas. Cinco objetivos simultâneos.



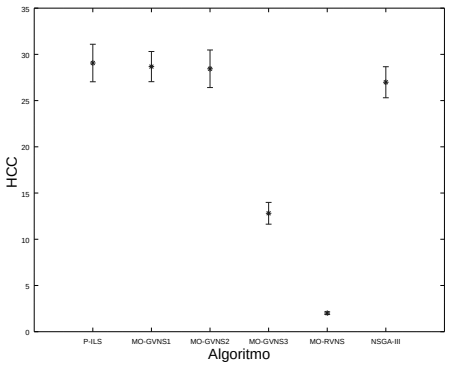
(a) $n = 50, m = 4, m_i = 2$



(b) $n = 50, m = 4, m_i = 4$

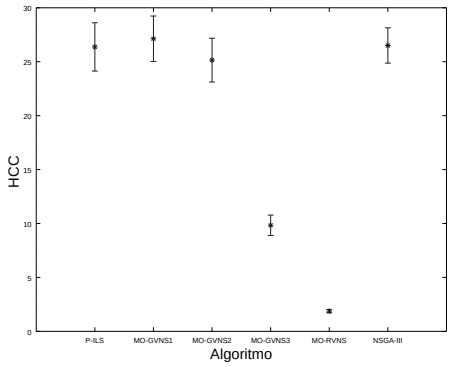


(c) $n = 50, m = 8, m_i = 2$

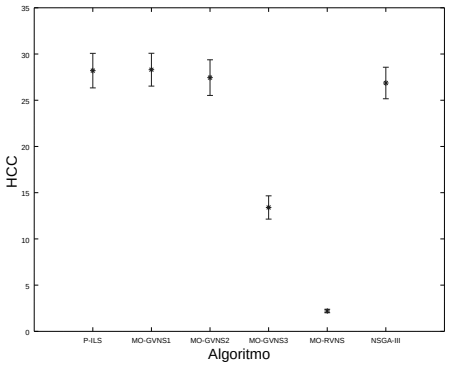


(d) $n = 50, m = 8, m_i = 4$

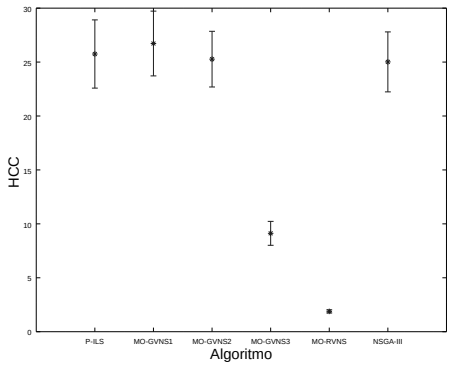
Figura II.37: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 50$). Cinco objetivos simultâneos.



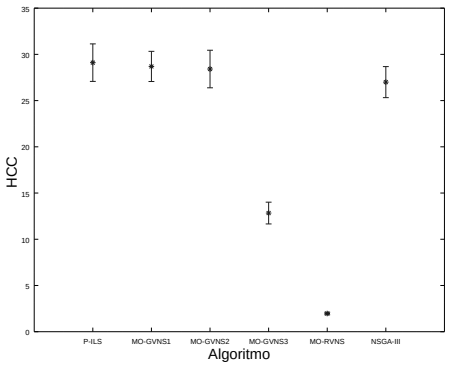
(a) $n = 100, m = 4, m_i = 2$



(b) $n = 100, m = 4, m_i = 4$

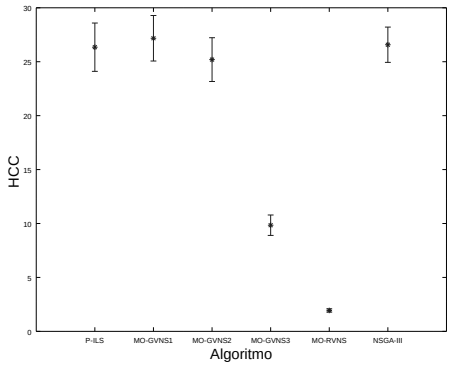


(c) $n = 100, m = 8, m_i = 2$

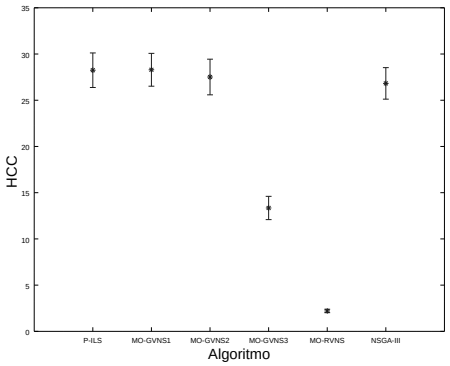


(d) $n = 100, m = 8, m_i = 4$

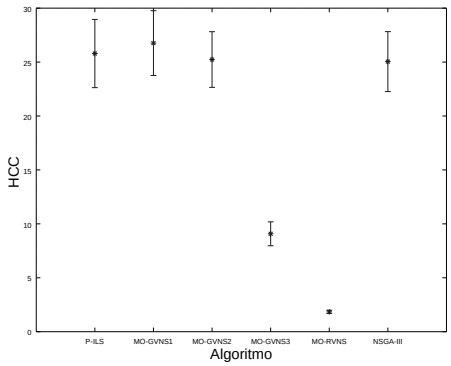
Figura II.38: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 100$). Cinco objetivos simultâneos.



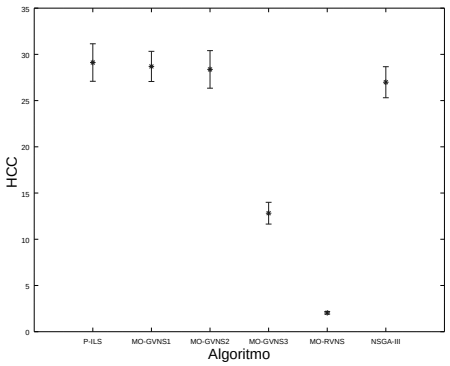
(a) $n = 150, m = 4, m_i = 2$



(b) $n = 150, m = 4, m_i = 4$

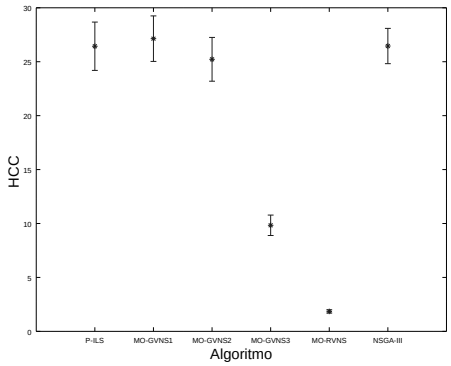


(c) $n = 150, m = 8, m_i = 2$

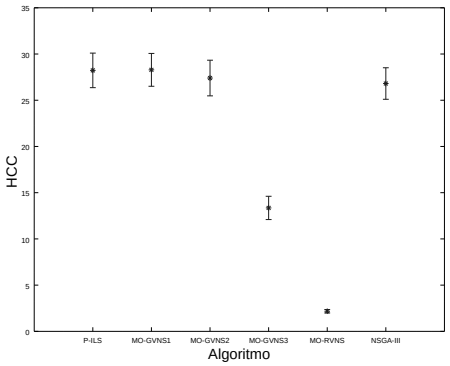


(d) $n = 150, m = 8, m_i = 4$

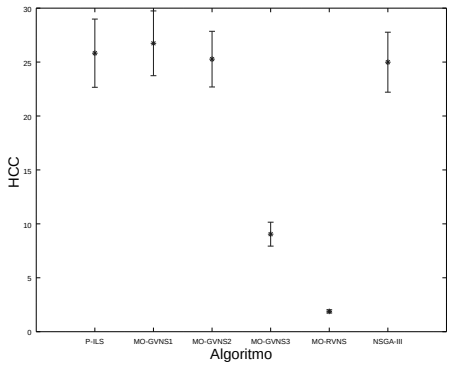
Figura II.39: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 150$). Cinco objetivos simultâneos.



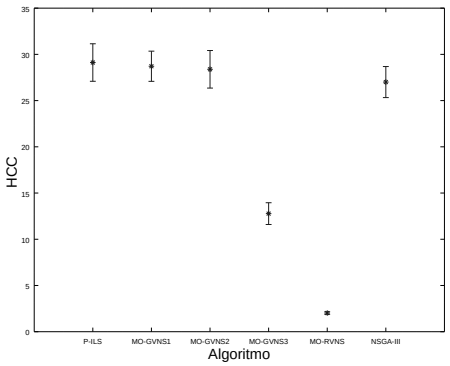
(a) $n = 200, m = 4, m_i = 2$



(b) $n = 200, m = 4, m_i = 4$



(c) $n = 200, m = 8, m_i = 2$



(d) $n = 200, m = 8, m_i = 4$

Figura II.40: Intervalo de Confiança na métrica *HCC*: instâncias grandes ($n = 200$). Cinco objetivos simultâneos.

Referências Bibliográficas

Arroyo, José Elias Claudio. *Heurísticas e metaheurísticas para otimização combinatória multiobjetivo*. Tese de doutorado, UNICAMP, Campinas – SP, (2002).

Arroyo, José Elias Claudio e Armentano, Vinícius Amaral. (2005). Genetic local search for multi-objective flowshop scheduling problems. *European Journal of Operational Research*, v. 167, n. 3, p. 717 – 738. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2004.07.017. URL `http://www.sciencedirect.com/science/article/pii/S0377221704004746`. Multicriteria Scheduling.

Arroyo, José Elias Claudio; dosSantos Ottoni, Rafael e de Paiva Oliveira, Alcione. (2011). Multi-objective variable neighborhood search algorithms for a single machine scheduling problem with distinct due windows. *Electronic Notes in Theoretical Computer Science*, v. 281, p. 5–19. doi: https://doi.org/10.1016/j.entcs.2011.11.022. URL `http://www.sciencedirect.com/science/article/pii/S157106611100171X`.

Asefi, H.; Jolai, F.; Rabiee, M. e Araghi, M. E. Tayebi. (2014). A hybrid NSGA-II and VNS for solving a bi-objective no-wait flexible flowshop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, v. 75, n. 5, p. 1017–1033. doi: 10.1007/s00170-014-6177-9. URL `https://doi.org/10.1007/s00170-014-6177-9`.

Behnamian, J.; Ghomi, S. M. T. Fatemi e Zandieh, M. (2009). A multi-phase covering Pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic. *Expert Systems with Applications*, v. 36, n. 8, p. 11057–11069.

Campos, Saulo Cunha e Arroyo, José Elias Claudio. (2014). NSGA-II with iterated greedy for a bi-objective three-stage assembly flowshop scheduling problem. *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO'14)*, p. 429–436, Vancouver, BC, Canada. ACM. doi: 10.1145/2576768.2598324. URL `http://doi.acm.org/10.1145/2576768.2598324`.

Chamnanlor, Chettha; Sethanan, Kanchana; Gen, Mitsuo e Chien, Chen-Fu. (2017). Embedding ant system in genetic algorithm for re-entrant hybrid flow shop scheduling problems with time window constraints. *Journal of Intelligent Manufacturing*, v. 28, n. 8, p. 1915–1931. doi: 10.1007/s10845-015-1078-9. URL `https://doi.org/10.1007/s10845-015-1078-9`.

Ciavotta, Michele; Minella, Gerardo e Ruiz, Rubén. (2013). Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study. *European Journal of Operational Research*, v. 227, n. 2, p. 301–313.

Coello, Carlos A. C.; Dhaenens, Clarisse e Jourdan, Laetitia. (2010). *Multi-Objective Combinatorial Optimization: Problematic and Context*, p. 1–21. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-11218-8. doi: 10.1007/978-3-642-11218-8_1. URL https://doi.org/10.1007/978-3-642-11218-8_1.

Czyzak, P. (1998). Pareto simulated annealing, a meta-heuristic technique for multiple objective combinatorial problems. *Journal of Multi-Criteria Decision Analysis, Journal of Multi-Criteria Decision Analysis*, v. 7, n. 1, p. 34–47.

de Siqueira, Eduardo Camargo; Souza, Marcone Jamilson Freitas e de Souza, Sérgio Ricardo. (2018). A multi-objective variable neighborhood search algorithm for solving the hybrid flow shop problem. *Electronic Notes in Discrete Mathematics*, v. 66, p. 87 – 94. ISSN 1571-0653. doi: https://doi.org/10.1016/j.endm.2018.03.012. URL http://www.sciencedirect.com/science/article/pii/S1571065318300581. 5th International Conference on Variable Neighborhood Search.

de Siqueira, Eduardo Camargo; Souza, Marcone Jamilson Freitas; de Souza, Sérgio Ricardo; de França Filho, Moacir Felizardo e Marcelino, Carolina Gil. (2013). An algorithm based on evolution strategies for makespan minimization in hybrid flexible flowshop scheduling problems. *Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC 2013)*, p. 989–996, Cancún, Mexico.

Deb, K.; Pratap, A.; Agarwal, S. e Meyarivan, T. (2002). A fast elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, v. 6, p. 182–197.

Deb, Kalyanmoy e Jain, Himanshu. (2014). 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*, v. 18, n. 4, p. 577–601.

Defersha, Fantahun Melaku e Chen, Mingyuan. (2012). Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem. *International Journal of Advanced Manufacturing Technology*, v. 62, n. 1, p. 249–265. doi: 10.1007/s00170-011-3798-0. URL https://doi.org/10.1007/s00170-011-3798-0.

Deng, Jin e Wang, Ling. (2017). A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm and Evolutionary Computation*, v. 32, p. 121–131.

Dios, Manuel; Fernandez-Viagas, Victor e Framinan, Jose M. (2018). Efficient heuristics for the hybrid flow shop scheduling problem with missing operations. *Computers & Industrial Engineering*, v. 115, p. 88–99. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2017.10.034. URL http://www.sciencedirect.com/science/article/pii/S0360835217305211.

Dorigo, M. e Stützle, T. (2004). *Ant colony optimization*. MIT Press, Cambridge, USA.

Duarte, Abraham; Pantrigo, Juan J.; Pardo, Eduardo G. e Mladenovic, Nenad. (2015). Multi-objective variable neighborhood search: an application to combinatorial optimization problems. *Journal of Global Optimization*, v. 63, n. 3, p. 515–536.

Dubois-Lacoste, Jérémie; López-Ibáñez, Manuel e Stützle, Thomas. (2009). Effective hybrid stochastic local search algorithms for biobjective permutation flowshop scheduling. Blesa, María J.; Blum, Christian; Di Gaspero, Luca; Roli, Andrea; Sampels, Michael e Schaerf, Andrea, editors, *Hybrid Metaheuristics*, p. 100–114, Berlin, Heidelberg. Springer Berlin Heidelberg. ISBN 978-3-642-04918-7.

Dugardin, F.; Yalaoui, F. e Amadeo, L. (2010). New multi-objective method to solve reentrant hybrid flow shop scheduling problem. *European Journal of Operational Research*, v. 203, n. 1, p. 22–31.

Ehrgott, Matthias. (2005). *Multicriteria Optimization*. Springer, 2nd edição.

Ehrgott, Matthias e Gandibleux, Xavier. (2008). *Hybrid Metaheuristics for Multi-objective Combinatorial Optimization*, p. 221–259. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-78295-7. doi: 10.1007/978-3-540-78295-7_8. URL `https://doi.org/10.1007/978-3-540-78295-7_8`.

Ehrgott, Matthias; Gandibleux, Xavier e Przybylski, Anthony. (2016). *Exact Methods for Multi-Objective Combinatorial Optimisation*, p. 817–850. Springer New York, New York, NY. ISBN 978-1-4939-3094-4. doi: 10.1007/978-1-4939-3094-4_19. URL `https://doi.org/10.1007/978-1-4939-3094-4_19`.

Feo, Thomas A e Resende, Mauricio GC. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, v. 6, n. 2, p. 109–133.

Fonseca, Carlos M; Fleming, Peter J e others,. (1993). Genetic algorithms for multiobjective optimization: Formulation discussion and generalization. *Icga*, volume 93, p. 416–423. Citeseer, (1993).

Geiger, Martin Josef. November 4–5(2004). Randomised variable neighbourhood search for multi objective optimisation. *Proceedings of EU/ME Workshop: Design and Evaluation of Advanced Hybrid Meta-heuristics*, p. 34–42, Nottingham, United Kingdom. Available at `https://arxiv.org/pdf/0809.0271.pdf`.

Geiger, Martin Josef. (2007). On operators and search space topology in multiobjective flow shop scheduling. *European Journal of Operational Research*, v. 181, n. 1, p. 195–206.

Geiger, Martin Josef. (2008). Randomised variable neighbourhood search for multi objective optimisation. *CoRR*, v. abs/0809.0271. URL `http://arxiv.org/abs/0809.0271`.

Goldbarg, M.C. e Luna, H.P.L. (2000). *Otimização combinatória e programação linear: modelos e algoritmos*. Campus.

Gomes, Helton Cristiano; dasNeves, Francisco de Assis e Souza, Marcone Jamilson Freitas. (2014). Multi-objective metaheuristic algorithms for the resource-constrained project scheduling problem with precedence relations. *Computers & Operations Research*, v. 44, p. 92–104. doi: https://doi.org/10.1016/j.cor.2013.11.002.

Gong, Dunwei; Han, Yuyan e Sun, Jianyong. (2018). A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems. *Knowledge-Based Systems*, v. 148, p. 115 – 130. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2018.02.029. URL `http://www.sciencedirect.com/science/article/pii/S0950705118300923`.

Guimaraes, F. G.; Wanner, E. F. e Takahashi, R. H. C. May(2009). A quality metric for multi-objective optimization based on hierarchical clustering techniques. *2009 IEEE Congress on Evolutionary Computation*, p. 3292–3299, May(2009). doi: 10.1109/CEC.2009.4983362.

Gupta, Jatinder N. D. e Tunc, Enar A. (1991). Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. *International Journal of Production Research*, v. 29, n. 7, p. 1489–1502. doi: 10.1080/00207549108948025. URL `https://doi.org/10.1080/00207549108948025`.

Hansen, Michael Pilegaard. (1997). Tabu search for multiobjective optimization: Mots. *Proceedings of the 13th international conference on multiple criteria decision making*, p. 574–586, (1997).

Hansen, Pierre; Mladenović, Nenad e Pérez, José A. Moreno. (2008). Variable neighborhood search: methods and applications. *4OR – A Quarterly Journal of Operations Research*, v. 6, n. 4, p. 319–360.

Hayrinen, Timo; Johnsson, Mika; Johtela, Tommi; Smed, Jouni e Nevalainen, Olli. (2000). Scheduling algorithms for computer-aided line balancing in printed circuit board assembly. *Production Planning & Control*, v. 11, n. 5, p. 497–510. doi: 10.1080/09537280050051997. URL `https://doi.org/10.1080/09537280050051997`.

Holland, John H. (1975). *Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence*. University of Michigan Press, Ann Arbor, MI, USA.

Hoogeveen, J.A.; Lenstra, J.K. e Veltman, B. (1996). Preemptive scheduling in a two-stage multiprocessor flow shop is np-hard. *European Journal of Operational Research*, v. 89, n. 1, p. 172 – 175. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(96)90070-3. URL `http://www.sciencedirect.com/science/article/pii/S0377221796900703`.

Horn, Jeffrey; Nafpliotis, Nicholas e Goldberg, David. (1994). A niched Pareto genetic algorithm for multiobjective optmization. *First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Volume 1*, (1994).

Ishibuchi, H. e Murata, T. May(1996). Multi-objective genetic local search algorithm. *Proceedings of IEEE International Conference on Evolutionary Computation*, p. 119–124, May(1996). doi: 10.1109/ICEC.1996.542345.

Ishibuchi, H. e Murata, T. Aug(1998). A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 28, n. 3, p. 392–403. ISSN 1094-6977. doi: 10.1109/5326.704576.

Ishibuchi, Hisao; Tsukamoto, Noritaka e Nojima, Yusuke. June(2008). Evolutionary many-objective optimization: A short review. *Proceedings of the 2008 IEEE Congress on Evolutionary Computation*, p. 2419–2426, Hong Kong, China.

Jain, Himanshu e Deb, Kalyanmoy. (2014). An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part II: Handling constraints and extending to an adaptive approach. *IEEE Transactions on Evolutionary Computation*, v. 18, n. 4, p. 602–622.

Jaszkiewicz, Andrzej. (2002). Genetic local search for multi-objective combinatorial optimization. *European Journal of Operational Research*, v. 137, n. 1, p. 50 – 71. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(01)00104-7. URL `http://www.sciencedirect.com/science/article/pii/S0377221701001047`.

Jenabi, M.; Ghomi, S.M.T. Fatemi; Torabi, S.A. e Karimi, B. (2007). Two hybrid meta-heuristics for the finite horizon elsp in flexible flow lines with unrelated parallel machines. *Applied Mathematics and Computation*, v. 186, n. 1, p. 230 – 245. ISSN 0096-3003. doi: https://doi.org/10.1016/j.amc.2006.06.121. URL `http://www.sciencedirect.com/science/article/pii/S0096300306009337`.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, v. 1, p. 61–68.

Jungwattanakit, Jitti; Reodecha, Manop; Chaovalitwongse, Paveena e Werner, Frank. (2005). An evaluation of sequencing heuristics for flexible flowshop scheduling problems with unrelated parallel machines and dual criteria. *Otto-von-Guericke-Universitat Magdeburg*, v. 28, n. 05, p. 1–23.

Knowles, Joshua D. e Corne, David W. (2000). Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, v. 8, n. 2, p. 149–172. doi: 10.1162/106365600568167. URL `https://doi.org/10.1162/106365600568167`.

Kostreva, Michael M.; Ogryczak, Wlodzimierz e Wierzbicki, Adam. (2004). Equitable aggregations and multiple criteria analysis. *European Journal of Operational Research*, v. 158, n. 2, p. 362 – 377. doi: https://doi.org/10.1016/j.ejor.2003.06.010. URL `http://www.sciencedirect.com/science/article/pii/S0377221703004892`.

Kostreva, Michael M. e Ogryczak, Wodzimierz. (1999). Linear optimization with multiple equitable criteria. *RAIRO - Operations Research*, v. 33, n. 3, p. 275–297. doi: 10.1051/ro:1999112. URL `https://doi.org/10.1051/ro:1999112`.

Lei, Deming e Zheng, Youlian. (2017). Hybrid flow shop scheduling with assembly operations and key objectives: A novel neighborhood search. *Applied Soft Computing*, v. 61, p. 122–128.

Levene, Howard. (1960). Robust tests for equality of variances. Olkin, I.; Ghurye, S. G.; Hoeffding, W.; Madow, W. G. e Mann, H. B., editors, *Contributions to probability and statistics: Essays in honor of Harold Hotelling*, p. 278–292. Stanford University Press, Palo Alto, CA, USA.

Li, Bingdong; Li, Jinlong; Tang, Ke e Yao, Xin. (2015). Many-objective evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, v. 48, n. 1, p. 13.

Li, Jun-Qing; Han, Yu-Yan e Wang, Cun-Gang. June(2017). A hybrid artificial bee colony algorithm to solve multi-objective hybrid flowshop in cloud computing systems. Sun, Xingming; Chao, Han-Chieh; You, Xingang e Bertino, Elisa, editors, *Proceedings of the Third International Conference on Cloud Computing and Security (ICCCS 2017)*, volume 1, p. 201–213, Nanjing, China. Springer International.

Li, Jun-Qing; Pan, Quan-Ke e Gao, Kai-Zhou. (2011). Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, v. 55, n. 9, p. 1159–1169.

Li, Jun-Qing; Sang, Hong-Yan; Han, Yu-Yan; Wang, Cun-Gang e Gao, Kai-Zhou. (2018). Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions. *Journal of Cleaner Production*, v. 181, p. 584 – 598. ISSN 0959-6526. doi: https://doi.org/10.1016/j.jclepro.2018.02.004. URL `http://www.sciencedirect.com/science/article/pii/S0959652618303081`.

Li, Xiangtao e Li, Mingjie. (2015). Multiobjective local search algorithm-based decomposition for multiobjective permutation flow shop scheduling problem. *IEEE Transactions on Engineering Management*, v. 62, n. 4, p. 544–557.

Li, Xiangtao e Ma, Shijing. (2016). Multi-objective memetic search algorithm for multi-objective permutation flow shop scheduling problem. *IEEE Access*, v. 4, p. 2154–2165. doi: 10.1109/ACCESS.2016.2565622.

Liang, Yun-Chia; Chen, Angela Hsiang-Ling e Tien, Chia-Yun. (2009). Variable neighborhood search for multi-objective parallel machine scheduling problems. *Proceedings of the 8th International Conference on Information and Management Sciences (IMS 2009)*, p. 519–522, Kunming, China.

Liang, Yun-Chia e Chuang, Chia-Yin. (2013). Variable neighborhood search for multi-objective resource allocation problems. *Robotics and Computer-Integrated Manufacturing*, v. 29, n. 3, p. 73–78. doi: https://doi.org/10.1016/j.rcim.2012.04.015.

López-Ibáñez, Manuel; Dubois-Lacoste, Jérémie; Stützle, Thomas e Birattari, Mauro. February(2011). The IRACE package: Iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de

Bruxelles, Belgium. Available at `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`.

López-Ibáñez, Manuel; Dubois-Lacoste, Jérémie; Pérez Cáceres, Leslie; Stützle, Thomas e Birattari, Mauro. (2016). The IRACE package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43–58. doi: 10.1016/j.orp.2016.09.002.

López-Sánchez, A.D.; Hernández-Díaz, A.G.; Gortázar, F. e Hinojosa, M.A. (2018). A multiobjective GRASP–VND algorithm to solve the waste collection problem. *International Transactions in Operational Research*, v. 25, n. 2, p. 545–567. ISSN 1475-3995. doi: 10.1111/itor.12452. URL `http://dx.doi.org/10.1111/itor.12452`.

Lourenço, H. R.; Martin, O. C. e Stützle, T. (2003). Iterated local search. Glover, F. e Kochenberger, G., editors, *Handbook of Metaheuristics*, International Series in Operations Research and Management Science, p. 321–353. Kluwer Academic Publishers, Boston.

Low, Chinyao. (2005). Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Computers & Operations Research*, v. 32, n. 8, p. 2013 – 2025. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2004.01.003. URL `http://www.sciencedirect.com/science/article/pii/S0305054804000048`.

Lu, Chao; Gao, Liang; Li, Xinyu; Pan, Quanke e Wang, Qi. (2017). Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. *Journal of Cleaner Production*, v. 144, p. 228 – 238. ISSN 0959-6526. doi: https://doi.org/10.1016/j.jclepro.2017.01.011. URL `http://www.sciencedirect.com/science/article/pii/S0959652617300112`.

Luenberger, David G e Ye, Yinyu. (1984). *Linear and nonlinear programming*, volume 2. Springer, 3 edição.

Marichelvam, M. K. e Geetha, M. (2014). Solving tri-objective multistage hybrid flow shop scheduling problems using a discrete firefly algorithm. *International Journal of Intelligent Engineering Informatics*, v. 2, n. 4, p. 284–303.

Masri, Hela; Krichen, Saoussen e Guitouni, Adel. (2019). Metaheuristics for solving the biobjective single-path multicommodity communication flow problem. *International Transactions in Operational Research*, v. 26, n. 2, p. 589–614.

Mersmann, Olaf. *EMOA: Evolutionary Multiobjective Optimization Algorithms*. Available at `https://CRAN.R-project.org/package=emoa`, September(2012). Version 0.5-0.

Minella, G.; Ruiz, R. e Ciavotta, M. (2008). A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing*, v. 20, n. 3, p. 451–471.

Minella, G.; Ruiz, R. e Ciavotta, M. (2011). Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers & Operations Research*, v. 38, n. 11, p. 1521–1533.

Mladenović, Nenad e Hansen, Pierre. (1997). Variable neighborhood search. *Computers & Operations Research*, v. 24, n. 11, p. 1097–1100.

Mousavi, S. M.; Mahdavi, I.; Rezaeian, J. e Zandieh, M. Apr(2018). An efficient bi-objective algorithm to solve re-entrant hybrid flow shop scheduling with learning effect and setup times. *Operational Research*, v. 18, n. 1, p. 123–158. ISSN 1866-1505. doi: 10.1007/s12351-016-0257-6. URL `https://doi.org/10.1007/s12351-016-0257-6`.

Murata, Tadahiko; Ishibuchi, Hisao e Gen, Mitsuo. (2000). Cellular genetic local search for multi-objective optimization. *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, p. 307–314. Morgan Kaufmann Publishers Inc., (2000).

Naderi, B.; Ruiz, R. e Zandieh, M. (2010). Algorithms for a realistic variant of flowshop scheduling. *Computers & Operations Research*, v. 37, p. 236–246.

Okabe, T.; Jin, Y. e Sendhoff, B. (2003). A critical survey of performance indices for multi-objective optimisation. *Proceedings of the 2003 Congress on Evolutionary Computation (CEC '03)*, volume 2, p. 878–885, Canberra, Australia.

Paquete, Luis; Chiarandini, Marco e Stützle, Thomas. (2004). Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. Gandibleux, Xavier; Sevaux, Marc; Sörensen, Kenneth e T'kindt, Vincent, editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical System*, p. 177–199. Springer.

Pargar, Farzad; Zandieh, Mostafa; Kauppila, Osmo e Kujala, Jaakko. Jun(2018). The effect of worker learning on scheduling jobs in a hybrid flow shop: A bi-objective approach. *Journal of Systems Science and Systems Engineering*, v. 27, n. 3, p. 265–291. ISSN 1861-9576. doi: 10.1007/s11518-018-5361-0. URL `https://doi.org/10.1007/s11518-018-5361-0`.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer, 3 edição.

Przybylski, Anthony e Gandibleux, Xavier. (2017). Multi-objective branch and bound. *European Journal of Operational Research*, v. 260, n. 3, p. 856 – 872. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2017.01.032. URL `http://www.sciencedirect.com/science/article/pii/S037722171730067X`.

Rajendran, Chandrasekharan e Chaudhuri, Dipak. (1992). A multi-stage parallel-processor flowshop problem with minimum flowtime. *European Journal of Operational Research*, v. 57, n. 1, p. 111 – 122. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(92)90310-6. URL `http://www.sciencedirect.com/science/article/pii/0377221792903106`.

Rego, M. F.; Souza, M. J. F.; Coelho, I. M. e Arroyo, J. E. C. (2014). Multi-objective algorithms for the single machine scheduling problem with sequence-dependent family setups. Snášel, Václav; Krömer, Pavel; Köppen, Mario e Schaefer, Gerald, editors, *Soft Computing in Industrial Applications*, volume 223 of *Advances in Intelligent Systems and Computing*, p. 117–127. Springer.

Ruiz, R.; Sivrikaya, F. e Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, v. 35, p. 1151–1175.

Ruiz, R. e Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, v. 177, p. 2033–2049.

Schilde, Michael; Doerner, Karl F; Hartl, Richard F e Kiechle, Guenter. (2009). Metaheuristics for the bi-objective orienteering problem. *Swarm Intelligence*, v. 3, n. 3, p. 179–201.

Schott, Jason R. (1995). Fault tolerant design using single and multicriteria genetic algorithm optimization. M.s. thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

Serafini, Paolo. (1994). Simulated annealing for multi objective optimization problems. Tzeng, G. H.; Wang, H. F.; Wen, U. P. e Yu, P. L., editors, *Multiple Criteria Decision Making*, p. 283–292, New York, NY. Springer New York. ISBN 978-1-4612-2666-6.

Srinivas, N. e Deb, K. (1995). Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation*, v. 2, n. 3, p. 221–248.

Sriskandarajah, C. e Sethi, S.P. (1989). Scheduling algorithms for flexible flowshops: Worst and average case performance. *European Journal of Operational Research*, v. 43, n. 2, p. 143 – 160. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(89)90208-7. URL `http://www.sciencedirect.com/science/article/pii/0377221789902087`.

Ticona, W. G. C. *Algoritmos evolutivos multi-objetivo para a reconstrução de árvores filogenéticas*. Tese de Doutorado, ICMC, USP, São Carlos, (2003).

Torkashvand, M.; Naderi, B. e Hosseini, S.A. (2017). Modelling and scheduling multi-objective flow shop problems with interfering jobs. *Applied Soft Computing*, v. 54, p. 221–228.

Urlings, T. *Heuristics and metaheuristics for heavily constrained hybrid flowshop problems*. PhD thesis, Universidad Politécnica de Valencia, (2010). Available at `https://riunet.upv.es/bitstream/handle/10251/8439/tesisUPV3302.pdf`.

Urlings, T. e Ruiz, R. (2010). Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems. *International Journal of Metaheuristics*, v. 1, p. 30–54.

Vianna, D. S. e Arroyo, J. E. C. Nov(2004). A grasp algorithm for the multi-objective knapsack problem. *XXIV International Conference of the Chilean Computer Science Society*, p. 69–75, Nov(2004). doi: 10.1109/QEST.2004.2.

Vignier, A.; Dardilhac, D.; Dezalay, D. e Proust, C. Nov(1996). A branch and bound approach to minimize the total completion time in a k-stage hybrid flowshop. *Proceedings 1996 IEEE Conference on Emerging Technologies and Factory Automation. ETFA '96*, volume 1, p. 215–220 vol.1, Nov(1996). doi: 10.1109/ETFA.1996.573294.

Wang, Hongfeng; Fu, Yaping; Huang, Min; Huang, George Q. e Wang, Junwei. (2017). A NSGA-II based memetic algorithm for multiobjective parallel flowshop scheduling problem. *Computers & Industrial Engineering*, v. 113, p. 185–194.

Wang, Shijin e Liu, Ming. (2014). Two-stage hybrid flow shop scheduling with preventive maintenance using multi-objective tabu search method. *International Journal of Production Research*, v. 52, n. 5, p. 1495–1508.

Wang, Xianpeng e Tang, Lixin. (2017). A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem. *Computers & Operations Research*, v. 79, p. 60–77.

Wittrock, R. J. July(1985). Scheduling algorithms for flexible flow lines. *IBM Journal of Research and Development*, v. 29, n. 4, p. 401–412. ISSN 0018-8646. doi: 10.1147/rd.294.0401.

Xu, Jianyou; Wu, Chin-Chia; Yin, Yunqiang e Lin, Win-Chin. (2017). An iterated local search for the multi-objective permutation flowshop scheduling problem with sequence-dependent setup times. *Applied Soft Computing*, v. 52, p. 39–47.

Xu, Liang; Yeming, Ji e Ming, Huang. October(2016). Solving hybrid flow-shop scheduling based on improved multi-objective artificial bee colony algorithm. *Proceedings of the 2nd International Conference on Cloud Computing and Internet of Things (CCIOT 2016)*, p. 43–47, Dalian, China. IEEE.

Yang, Xin-She. (2010). *Nature-inspired metaheuristic algorithms*. Luniver Press, UK, 2nd edição. ISBN 978-1905986286. Available at `http://web-ng.info.uvt.ro/~dzaharie/ma2017/projects/techniques/`.

Yen, G. G. e He, Z. (2014). Performance metric ensemble for multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, v. 18, n. 1, p. 131–144. doi: 10.1109/TEVC.2013.2240687.

Ying, Kuo-Ching; Lin, Shih-Wei e Wan, Shu-Yen. (2014). Bi-objective reentrant hybrid flowshop scheduling: an iterated pareto greedy algorithm. *International Journal of Production Research*, v. 52, n. 19, p. 5735–5747.

Yuan, Xiaohui; Tian, Hao; Yuan, Yanbin; Huang, Yuehua e Ikram, Rana M. (2015). An extended NSGA-III for solution multi-objective hydro-thermal-wind scheduling considering wind power cost. *Energy Conversion and Management*, v. 96, p. 568–578.

Yuan, Yuan; Xu, Hua e Wang, Bo. (2014). An improved NSGA-III procedure for evolutionary many-objective optimization. *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO 2014)*, p. 661–668. ACM, (2014).

Zandieh, M.; Mozaffari, E. e Gholami, M. (2010). A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems. *Journal of Intelligent Manufacturing*, v. 21, p. 731–743.

Zitzler, Eckart; Laumanns, Marco e Thiele, Lothar. (2002). SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. Giannakoglou, K. C. e others,, editors, *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)*, p. 95–100. International Center for Numerical Methods in Engineering (CIMNE), (2002).

Zitzler, Eckart e Thiele, Lothar. may(1998). An Evolutionary Approach for Multiobjective Optimization: The Strength Pareto Approach. TIK Report 43, Computer Engineering and Networks Laboratory (TIK), ETH Zurich.

Zitzler, Eckart; Thiele, Lothar; Laumanns, Marco; eDa Fonseca, Carlos M e Grunert, Viviane. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, v. 7, n. 2, p. 117–132.