Abordagens exatas e heurísticas para o problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência

Luciano Perdigão Cota
Universidade Federal de Minas Gerais

Orientador: Frederico Gadelha Guimarães
Coorientador: Marcone Jamilson Freitas Souza

Tese submetida ao
Programa de Pós-Graduação em Engenharia Elétrica
da Universidade Federal de Minas Gerais
como requisito para obtenção do título de Doutor em Engenharia Elétrica.

Abordagens exatas e heurísticas para o problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência

Luciano Perdigão Cota
Universidade Federal de Minas Gerais

Área de Concentração: Sistemas de Computação e Telecomunicações

Linha de Pesquisa: Inteligência Computacional

Orientador: Frederico Gadelha Guimarães

Coorientador: Marcone Jamilson Freitas Souza

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

MINDS
MACHINE INTELLIGENCE AND DATA SCIENCE

PPGEE

Cota, Luciano Perdigão.
C843a Abordagens exatas e heurísticas para o problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência [manuscrito] / Luciano Perdigão Cota. - 2018.
xxviii, 134 f., enc.: il.

Orientador: Frederico Gadelha Guimarães.
Coorientador: Marcone Jamilson Freitas Souza.

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

Apêndices: f. 121-122.

Bibliografia: f. 123-134.

1. Engenharia elétrica - Teses. 2. Otimização combinatória - Teses. 3. Programação heurística - Teses. 4. Programação (Matemática) - Teses. I. Guimarães, Frederico Gadelha. II. Souza, Marcone Jamilson Freitas. III. Universidade Federal de Minas Gerais. Escola de Engenharia. IV. Título.

CDU: 621.3(043)

Dedico este trabalho aos meus pais José Perdigão (in memoriam) e Maria Aparecida, aos meus irmãos Renato e Ana Paula, e a minha noiva Paula Amora.

Resumo

O sequenciamento de máquinas desempenha um papel importante na indústria, permitindo que gerentes e engenheiros de produção maximizem a produtividade ao alocar as tarefas aos recursos disponíveis de maneira eficiente. Neste trabalho são propostas abordagens exatas e heurísticas para resolver um importante problema de sequenciamento de máquinas. O problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência é tratado especificamente. Duas versões deste problema são abordadas, uma mono-objetivo e a outra multiobjetivo. A versão mono-objetivo tem como objetivo minimizar o *makespan*, esta pode ser considerada uma versão clássica do problema. Já a multiobjetivo é uma versão precursora, no contexto de *green scheduling*, que tem como objetivos minimizar o *makespan* e o consumo total de energia elétrica.

Para resolver o problema na versão clássica, mono-objetivo, é proposto um novo algoritmo chamado LA-ALNS que combina o *Adaptive Large Neighborhood Search* com aprendizagem em autômatos para ajustar as probabilidades das heurísticas de inserção e remoção. Este algoritmo encontrou melhores resultados que outros cinco importantes algoritmos da literatura, mostrando que possui grande capacidade de se adaptar a diferentes variações do problema, devido ao processo de aprendizagem com autômatos.

Para resolver o problema na versão multiobjetivo é construído e implementado um modelo de programação linear inteira mista. Para implementar este modelo matemático é usado o método clássico ϵ-restrito e o método *Smart Pool*, recentemente proposto. Nos experimentos é exposto o conflito existente entre os dois objetivos, destacando a relevância desta versão do problema. O método *Smart Pool* alcançou boa convergência em relação à fronteira Pareto verdadeira e obteve melhores resultados que o método ϵ-restrito, mostrando assim, ser um método eficiente para resolver problemas de pequeno a médio porte em relação aos métodos tradicionais. Para resolver versões de grande

porte deste mesmo problema são propostos dois algoritmos multiobjetivo. O primeiro algoritmo é chamado MO-ALNS e trata-se de uma versão multiobjetivo do LA-ALNS. O segundo algoritmo é chamado MOEA/D + LA-ALNS e combina o algoritmo multiobjetivo MOEA/D e o algoritmo mono-objetivo LA-ALNS. Neste algoritmo é usado o LA-ALNS para explorar os subproblemas escalares do MOEA/D. Nos experimentos computacionais o algoritmo MOEA/D + LA-ALNS obteve os melhores resultados para três métricas de qualidade avaliadas. Uma das grandes qualidades deste algoritmo é o controle da diversificação na aproximação da fronteira Pareto.

Palavras-chave: sequenciamento de máquinas, sequenciamento verde, *adaptive large neighborhood search*, aprendizagem com autômatos, programação linear inteira mista, MOEA/D.

Abstract

Machine scheduling plays an important role in manufacturing industry, allowing production managers and production engineers to maximize productivity by allocating the jobs to the resources available in an optimal way. In this work exact and heuristic approaches are proposed to solve an important machine scheduling problem, namely the unrelated parallel machine scheduling problem with setup times. Two versions of this problem are considered, the mono-objective one and the multiobjective one. In the mono-objective version the minimization of the makespan is considered, as a classic version of the problem. The multiobjective problem is an original version within the context of green scheduling. This version has the objectives of minimizing the makespan and the total consumption of electricity.

To solve the problem in the single objective version, a new algorithm called LA-ALNS is proposed, which combines the Adaptive Large Neighborhood Search with learning automata to adjust the probabilities of the insertion and removal heuristics. This algorithm found better results than other five important algorithms in the literature, showing that it has great capacity of adapting to different variations of the problem, thanks to the process of learning automata.

To solve the problem in the multiobjective version, a mixed integer linear programming model is constructed and implemented. To solve this mathematical model, the classic ϵ-constrained method and the recently proposed Smart Pool method are used. In the experiments the conflict between the two objectives is shown, proving the relevance of this version of the problem. The Smart Pool method achieved good convergence in relation to the true Pareto front and obtained better results than the ϵ-constrained method. The Smart Pool method showed to be an efficient method to solve small to medium problems in relation to the traditional methods. In order to solve large versions of this same problem, two multiobjective algorithms are proposed. The first algorithm

is called MO-ALNS and it is a multiobjective version of LA-ALNS. The second algorithm is called MOEA/D + LA-ALNS and combines the MOEA/D and the LA-ALNS algorithm. In this algorithm the LA-ALNS is used to explore the scalar subproblems of the MOEA/D. In the computational experiments the MOEA/D + LA-ALNS algorithm obtained the best results for two quality metrics evaluated. One of the great qualities of this algorithm is the control of diversity in the approximation of the Pareto front.

Keywords: *machine scheduling*, *green scheduling*, *adaptive large neighborhood search*, *learning automata*, *mixed integer linear programming*, MOEA/D.

Declaração

Esta tese é resultado de meu próprio trabalho, exceto onde referência explícita é feita ao trabalho de outros, e não foi submetida para obtenção de título nesta nem em outra universidade.

Luciano Perdigão Cota

Agradecimentos

Ao meu pai José Perdigão *(in memoriam)*, que mesmo tendo vivido apenas 13 anos ao seu lado, sempre esteve presente em meus sentimentos. Sua bondade, caráter e humildade sempre serão fonte de grande inspiração.

À minha mãe Maria Aparecida símbolo para mim de fé, perseverança e determinação.

Ao meu irmãos Renato e Ana Paula, e seus cônjuges, pela união e incentivo contínuo, juntamente com minha pequena sobrinha Giovanna.

À minha noiva Paula Amora que esteve ao meu lado em todos os momentos desta caminhada, com grande paciência, reconhecimento e apoio incondicional.

Aos demais familiares pela força extra de incentivo e reconhecimento.

Aos meus amigos que sempre estiveram dispostos a ajudar, em especial aos amigos de infância Thieres e Samuel.

Aos membros do laboratório MINDS, que mesmo estando pouco presente, sempre foram muito prestativos e gentis. Em especial ao professor Fernando que foi grande colaborador nos trabalhos de pesquisa realizados.

Aos colaboradores do Programa de Pós-Graduação em Engenharia Elétrica da UFMG que forneceram toda a estrutura adequada para a minha formação no doutorado. Além disso, agradeço também pelos auxílios para participar do Simpósio Brasileiro de Pesquisa Operacional 2017 (SBPO) em Blumenau/SC e IEEE Congress on Evolutionary Computation 2017 (CEC) em San Sebastián/Donostia na Espanha. A viagem para participação do CEC foi a minha primeira para o exterior, e sem dúvida foi uma grande experiência cultural e acadêmica.

Aos meus orientadores Frederico e Marcone, que são pessoas de uma ética admirável

e que são fonte de grande inspiração para seguir o caminho no ensino e na pesquisa científica. Ao Marcone que acreditou no meu potencial quando entrei no mestrado em Ciência da Computação da UFOP, mesmo sem me conhecer previamente. Ao Frederico que sempre me apoiou com excelentes ideias para novos trabalhos e com muito incentivo para continuar "batalhando" em busca dos meus objetivos. Mesmo estando estes últimos meses em Paris, no seu pós-doutorado, sempre esteve muito presente em todas as etapas do meu doutorado, onde mantivemos conversas quase que diariamente.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Nomenclatura

ACO	*Ant Colony Optimization*
ALNS	*Adaptive Local Neighborhood Search*
ANOVA	Análise de Variância
C_{max}	*Makespan*
CS	*Coverage Between Two Sets*
GRASP	*Greedy Randomized Adaptive Search Procedures*
LA	*Learning Automata*
LC	Lista de Candidatos
HV	Hipervolume
ILS	*Iterated Local Search*
LNS	*Local Neighborhood Search*
MO-RVND	*Multi-Objective Random Variable Neighborhood Descent*
PLIM	Programação Linear Inteira Mista
RPD	Desvio Percentual Relativo
RVND	*Random Variable Neighborhood Descent*
TEC	*Total Energy Consumption*
UPMSP-ST	*Unrelated Parallel Machine Scheduling Problem with Setup Times*
VND	*Variable Neighborhood Descent*
VRP	*Vehicle Routing Problem*

Capítulo 1

Introdução

Os problemas de sequenciamento (ou *scheduling* no idioma inglês) têm atraído a atenção de muitos pesquisadores, por sua importância na indústria e relevância teórica na área de otimização (Zhu e Wilhelm, 2006). Este tipo de problema trata da alocação de recursos, que podem ser tarefas ou serviços, buscando otimizar um ou mais objetivos.

Uma classe importante dos problemas de *scheduling* são os problemas de sequenciamento em máquinas, que tratam da alocação de tarefas às máquinas.

Este trabalho trata especificamente o problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência (ou UPMSP-ST, do termo *Unrelated Parallel Machine Scheduling Problem with Setup Times* no idioma inglês). O UPMSP-ST tem grande importância prática e teórica dada a sua ampla aplicabilidade nas indústrias e a dificuldade existente em sua resolução. Ele aparece em processos produtivos de indústrias de diferentes ramos, como as têxteis, químicas, tintas, semicondutores e de papel (Pereira Lopes e de Carvalho, 2007; Rabadi et al., 2006).

Neste trabalho são tratadas duas versões do problema, uma mono-objetivo e outra multiobjetivo. A versão mono-objetivo é uma versão clássica do problema com o objetivo de minimizar o *makespan*. Na literatura são encontradas várias pesquisas abordando a resolução desta versão do problema. Dois trabalhos muito citados são Rabadi et al. (2006) e Vallada e Ruiz (2011). Em cada um deles são propostos um modelo matemático, algoritmos heurísticos e um grande conjunto de instâncias. Desde então, esses conjuntos de instâncias têm sido amplamente usados na literatura.

A versão multiobjetivo é uma nova abordagem do problema proposta neste trabalho. Neste problema tem-se os objetivos de minimizar o *makespan* e o consumo total de energia elétrica. A motivação para esta versão do problema vem de um tema crescente nos últimos anos na área de *scheduling*, o chamado *green scheduling* (ou sequenciamento verde), tema que envolve o desenvolvimento sustentável em problemas de sequenciamento (Bampis et al., 2015; Mansouri et al., 2016a,b). Segundo Sauer et al. (2015), o setor industrial é responsável por 44% do consumo de eletricidade no Brasil. A geração de energia no Brasil é composta principalmente de hidrelétricas, e termelétricas para fornecer energia adicional quando necessário. Apesar do crescente uso da geração distribuída nos últimos anos, a participação de fontes de energia renováveis ainda é tímida, abaixo de 5% da produção nacional de energia. Assim, reduzir o consumo total de energia na indústria é fundamental para diminuir as emissões de gases do efeito estufa e reduzir indiretamente os custos de produção na indústria.

A seguir, uma breve apresentação das características dos problemas é dada, e posteriormente, no Capítulo 2, os problemas são descritos de maneira detalhada.

Basicamente, na versão mono-objetivo tem-se um conjunto de tarefas e um conjunto de máquinas com as seguintes características: *i*) cada tarefa deve ser alocada a uma única máquina; *ii*) Existe um tempo de processamento para processar cada tarefa em uma máquina; *iii*) Existe um tempo de preparação que depende da ordem de alocação das tarefas nas máquinas; *iv*) O objetivo é alocar todas as tarefas nas máquinas, minimizando o tempo máximo do sequenciamento, o chamado *makespan*. Este problema pode ser referenciado por $R_M|S_{ijk}|C_{\max}$, usando a notação clássica de Graham et al. (1979). Nesta notação, R_M representa os tempos de processamento, S_{ijk} os tempos de preparação e $C_{\max}$ o *makespan*. O $R_M|S_{ijk}|C_{\max}$ pode ser considerado uma versão clássica do problema e já é tratada na literatura há algum tempo.

Na versão multiobjetivo tem-se duas características adicionais em relação a versão mono-objetivo, dadas a seguir. Existe uma potência para cada máquina, e existe uma velocidade de processamento da tarefa na máquina. Neste problema os objetivos são alocar todas as tarefas às máquinas, minimizando o *makespan* e o consumo total de energia. Este problema pode ser definido por $R_M|S_{ijk}|(C_{\max}, TEC)$, em que o TEC (do termo *Total Energy Consumption* no idioma inglês) é o consumo total de energia. Os objetivos $C_{\max}$ e TEC têm grande importância neste problema, porque a minimização do *makespan* normalmente implica em bom uso da máquina (Pinedo, 2008) e a minimização do TEC implica em redução dos custos para as indústrias e o uso consciente de recursos ambientais. A natureza conflitante dos objetivos $C_{\max}$ e TEC deve-se ao *trade-off* entre

maximizar a produção em altas velocidades, o que levaria a um maior consumo de energia do sequenciamento.

No $R_M|S_{ijk}|C_{\max}$ existe grande dificuldade em resolver problemas de grande porte em um tempo restrito. Em Avalos-Rosales et al. (2015) é proposto um modelo matemático capaz de resolver instâncias com até 60 tarefas de maneira ótima em uma média de uma hora de processamento [1]. Como descrito anteriormente, existem dois grandes conjuntos de instâncias na literatura para este problema. Cada um deles possui características bem distintas; por isso, grande parte dos trabalhos da literatura abordam apenas um destes conjuntos de instâncias. O foco deste trabalho na resolução deste problema é propor um algoritmo de propósito geral que seja capaz de resolver diferentes tipos de problemas de forma eficaz e tenha grande aplicabilidade prática. A ideia é desenvolver um algoritmo adaptativo que aprenda durante o processo de busca e que tenha poucos parâmetros. O algoritmo proposto é um *Adaptive Large Neighborhood Search* – ALNS (Ropke e Pisinger, 2006), com a novidade de usar aprendizagem em autômatos (ou *Learning Automata* no idioma inglês) para ajustar as probabilidades de aplicação das heurísticas de remoção e inserção, uma vez que essas heurísticas têm um grande impacto no desempenho do ALNS.

Na literatura vários trabalhos tratam problemas de sequenciamento em máquinas paralelas com tempos de preparação dependentes da sequência, mas poucas abordagens analisam o consumo de energia. Nesse sentido, o presente trabalho contribui com uma visão mais sustentável em relação aos problemas de sequenciamento de máquinas clássicos. Para este propósito, é tratado o problema $R_M|S_{ijk}|(C_{\max}, TEC)$ com os objetivos de minimizar o *makespan* e o consumo total de energia. Para resolver este problema, um modelo matemático de programação linear inteira mista (PLIM) é construído, implementado e analisado. O modelo é resolvido usando o método clássico ϵ-restito e o método *Multi-Objective Smart Pool Search Matheuristic* (ou *Smart Pool*), proposto em Coelho et al. (2016a).

Para resolver versões de grande porte do problema $R_M|S_{ijk}|(C_{\max}, TEC)$, são propostos dois novos algoritmos multiobjetivo. A ideia de implementar estes algoritmos vem da carência dos métodos exatos em resolver problemas de grande porte em um tempo restrito. O primeiro algoritmo proposto é uma versão multiobjetivo do ALNS com aprendizado em autômatos. Já o segundo algoritmo proposto é uma combinação do algoritmo multiobjetivo MOEA/D (Zhang e Li, 2007) e o algoritmo mono-objetivo

[1] Nos experimentos foi utilizado um computador Pentium Dual Core, 2,0 GHz, 3 GB de memória RAM e sistema operacional Ubuntu 11.1.

ALNS com aprendizagem em autômatos. No MOEA/D original um problema multiobjetivo é decomposto em S subproblemas de otimização mono-objetivo por meio de métodos de decomposição; para isso, são usados vetores de peso distribuídos uniformemente. Na versão original os subproblemas são explorados por meio de operadores de reprodução. O algoritmo multiobjetivo MOEA/D proposto neste trabalho, no entanto, se difere do MOEA/D original neste último aspecto; pois, ao invés de usar operadores de reprodução, é utilizado o algoritmo ALNS com aprendizagem em autômatos para explorar os subproblemas escalares do MOEA/D.

A seguir são apresentadas as motivações, os objetivos, as contribuições e a organização deste trabalho.

1.1 Motivação

A motivação para realizar este trabalho se deve a basicamente três fatores, como segue: *i*) o problema tratado tem grande aplicabilidade prática; *ii*) existe grande desafio da comunidade científica em resolvê-lo; *iii*) escassez na literatura de abordagens sustentáveis para o problema em questão. A seguir, cada um destes fatores é detalhado.

A resolução do problema tratado de forma eficaz pode implicar em aperfeiçoamentos dos processos de produção das organizações, tornando-as mais competitivas no mercado. Como citado anteriormente, este problema aparece em organizações que produzem produtos de diversos tipos, como tecidos, produtos químicos, tintas, semicondutores e papel (Pereira Lopes e de Carvalho, 2007; Rabadi et al., 2006).

O problema em questão pertence à classe de problemas $\mathcal{NP}$-difíceis, uma vez que é uma generalização do problema *parallel machine scheduling problem with identical machines and without setup times*, que foi comprovado pertencer à classe de problemas $\mathcal{NP}$-difíceis em Garey e Johnson (1979); Karp (1972). Este fato torna desafiador o desenvolvimento de técnicas eficazes para a sua resolução. Dada a dificuldade de resolução do problema, o uso de técnicas exatas é viável, em geral, apenas em problemas de pequeno e médio porte; para os demais problemas, geralmente são empregadas técnicas heurísticas.

As pesquisas de metodologias para o uso eficiente de materiais e recursos energéticos nas áreas de produção das organizações estão intensificando, dada a escassez destes recursos e a crescente conscientização das pessoas que exigem o desenvolvimento sus-

tentável (Mansouri et al., 2016a). Este processo pode envolver a reutilização de materiais, a reciclagem, o menor consumo de energia, a menor geração de poluentes, o uso consciente de recursos hídricos, o uso de energias renováveis, descentralização do sistema energético (Coelho et al., 2017b) e investimentos em *microgrids* alimentados por energias renováveis (Coelho et al., 2017a), entre outros. Existe escassez na literatura de abordagens de desenvolvimento sustentável para o problema tratado; com isso, este trabalho busca contribuir nessa área e impulsionar novas pesquisas.

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo deste trabalho é produzir novas técnicas exatas e heurísticas para a resolução do problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência de maneira eficaz. São tratadas duas variações do UPMSP-ST. A primeira é o $R_M|S_{ijk}|C_{\max}$ objetivando minimizar o *makespan*, problema este que já vem sendo estudado na literatura há alguns anos. A segunda variação é o $R_M|S_{ijk}|(C_{\max}, TEC)$ objetivando minimizar o consumo total de energia elétrica e o *makespan*, abordagem esta que ainda não é encontrada na literatura.

1.2.2 Objetivos específicos

Para alcançar o objetivo geral se faz necessário a obtenção dos seguintes objetivos específicos:

- Estudar e analisar trabalhos da literatura que tratam o problema abordado e relacionados;
- Estudar e analisar trabalhos da literatura que tratam do tema *green scheduling*;
- Estudar e analisar técnicas exatas, heurísticas e de inteligência artificial para problemas de otimização;
- Desenvolver técnicas para resolver o $R_M|S_{ijk}|C_{\max}$:

  - Propor um novo algoritmo adaptativo com a utilização de diferentes técnicas heurísticas, meta-heurísticas e de inteligência artificial para resolver o problema de maneira eficaz;
  - Realizar experimentos computacionais com instâncias da literatura, com o objetivo de comprovar a eficiência do algoritmo proposto.

- Desenvolver técnicas para resolver o $R_M|S_{ijk}|(C_{\max}, TEC)$:
  - Definir esta nova abordagem do problema;
  - Propor um modelo matemático e implementar métodos exatos para a sua resolução;
  - Propor algoritmos multiobjetivo para resolver de maneira eficaz versões de grande porte do problema;
  - Criar um conjunto de instâncias de pequeno e grande portes para o problema;
  - Realizar experimentos computacionais utilizando as instâncias propostas, com os objetivos de comprovar a importância desta nova abordagem do problema e mostrar a eficiência das técnicas propostas para sua resolução.

1.3 Contribuições

As principais contribuições deste trabalho são:

- Desenvolvimento de um algoritmo ALNS com aprendizado em autômatos para o $R_M|S_{ijk}|C_{\max}$ (conteúdo descrito no Capítulo 4):
  - O algoritmo proposto é baseado na meta-heurística ALNS e é adaptativo, uma vez que utiliza regras de aprendizado em autômatos para ajustar as probabilidades de aplicação das heurísticas de remoção e inserção durante a exploração do espaço de soluções do problema;
  - Um dos principais métodos de inserção utilizado é baseado no algoritmo Húngaro, tendo a vantagem de conseguir resolver problemas de atribuição linear de maneira ótima em tempo polinomial;
  - Parte deste conteúdo foi publicado em Cota et al. (2017a,b).

- Desenvolvimento de uma formulação de programação matemática para o $R_M|S_{ijk}|(C_{\max}, TEC)$ (conteúdo descrito no Capítulo 5):

  - É definida uma abordagem de desenvolvimento sustentável para o problema UPMSP-ST. Nesta abordagem os objetivos são minimizar o *makespan* e o consumo total de energia elétrica;
  - Um modelo matemático de programação linear inteira mista é construído, implementado e analisado;
  - Um dos métodos utilizados para resolver o modelo matemático é o *Smart Pool*, proposto em Coelho et al. (2016a). A ideia utilizada neste método é transformar um problema multiobjetivo em múltiplos problemas de soma ponderada de um único objetivo. Neste trabalho o método *Smart Pool* é aperfeiçoado com o uso do método de Scheffé (Scheffé, 1958) para a geração dos vetores de peso.

- Desenvolvimento de algoritmos multiobjetivo para o $R_M|S_{ijk}|(C_{\max}, TEC)$ (conteúdo descrito no Capítulo 6):

  - Uma versão multiobjetivo do algoritmo ALNS com aprendizagem em autômatos proposto no Capítulo 4;
  - Uma nova versão do algoritmo multiobjetivo MOEA/D que utiliza o ALNS com aprendizagem em autômatos para explorar os subproblemas escalares, ao invés dos operadores de reprodução. Os algoritmos multiobjetivo propostos são utilizados para resolver problemas de grande porte do $R_M|S_{ijk}|(C_{\max}, TEC)$.

No Apêndice A são listadas todas as publicações relacionadas e geradas ao longo do desenvolvimento deste trabalho.

1.4 Estrutura do trabalho

O restante deste trabalho está organizado da seguinte forma:

Capítulo 2 - Problemas de sequenciamento em máquinas: neste capítulo é apresentada a primeira parte da revisão bibliográfica que trata dos problemas de sequenciamento em máquinas. Inicialmente é tratado um pouco da história e importância

dos problemas de sequenciamento. A notação e classificação dos problemas de sequenciamento também é descrita. Posteriormente, a definição das duas versões do problema tratado são apresentadas, juntamente com a revisão bibliográfica de cada uma delas. Ao final, são tratadas as conclusões do capítulo.

Capítulo 3 - Métodos para problemas de sequenciamento em máquinas: neste capítulo é apresentada a segunda parte da revisão bibliográfica que trata de métodos para a resolução de problemas de sequenciamento em máquinas. Uma breve descrição dos problemas de otimização mono-objetivo e multiobjetivo é realizada. Posteriormente, são apresentados alguns métodos da literatura para resolver problemas dessa natureza. Estes métodos foram classificados em heurísticas, meta-heurísticas e métodos exatos. Ao final, são tratadas as conclusões do capítulo.

Capítulo 4 - ALNS com aprendizado em autômatos para o $R_M|S_{ijk}|C_{\max}$: neste capítulo é tratada a resolução do problema $R_M|S_{ijk}|C_{\max}$. São apresentados os conceitos de aprendizagem em autômatos e o algoritmo adaptativo proposto para a resolução do problema. Em seguida, são apresentados os resultados dos experimentos computacionais e as conclusões do capítulo.

Capítulo 5 - Modelo matemático para o $R_M|S_{ijk}|(C_{\max}, TEC)$: este capítulo trata da resolução do problema $R_M|S_{ijk}|(C_{\max}, TEC)$. O modelo matemático proposto para a resolução do problema e o método *Smart Pool* são apresentados. Em seguida, são exibidos os resultados dos experimentos computacionais e as conclusões do capítulo.

Capítulo 6 - Algoritmos multiobjetivo para o $R_M|S_{ijk}|(C_{\max}, TEC)$: neste capítulo é tratada a resolução de versões de grande porte do problema $R_M|S_{ijk}|(C_{\max}, TEC)$. Os algoritmos multiobjetivo propostos são apresentados inicialmente. Posteriormente, são descritos os resultados dos experimentos computacionais e as conclusões do capítulo.

Capítulo 7 - Conclusões: são apresentadas as conclusões deste trabalho e as perspectivas de trabalhos futuros.

Capítulo 2

Problemas de sequenciamento em máquinas

2.1 Introdução

2.1.1 História e importância

> Os problemas de *scheduling* (ou sequenciamento) são amplamente encontrados no processo de tomada de decisão das indústrias de manufatura e serviços. Neste tipo de problema é tratada a alocação de recursos para tarefas ou serviços em determinados períodos de tempo com o objetivo de otimizar um ou mais objetivos (Pinedo, 2008).

Os recursos e tarefas podem aparecer de diversas formas. Em uma indústria de tintas, por exemplo, os recursos podem ser as máquinas que produzem as tintas e as tarefas podem ser os tipos de tintas que podem ser produzidas; neste caso, o problema de sequenciamento pode tratar da alocação das tintas às máquinas. Já em um problema de alocação de contêineres de navios, os recursos podem ser os contêineres e as tarefas as cargas que precisam ser alocadas nesses contêineres; neste contexto, o problema de sequenciamento pode ser a alocação das cargas aos contêineres. As tarefas também podem assumir o papel de serviços, por exemplo, em uma alocação de mão de obra para construção civil, os recursos podem ser as obras de construção civil e as tarefas os serviços prestados pelos profissionais especializados. Este problema de sequenciamento pode envolver a alocação dos serviços prestados pelos profissionais nas obras.

Os objetivos a serem otimizados em problemas de sequenciamento podem tratar de diferentes finalidades, dependendo do problema em questão. Considerando os exemplos anteriores, no problema de sequenciamento em uma indústria de tintas os objetivos podem ser a minimização do tempo da alocação, a minimização do consumo de energia, a minimização do atraso, a maximização da qualidade das tintas, dentre outros. No problema da alocação de contêineres os objetivos podem ser a minimização de espaço ocioso dentro de cada contêiner e a minimização do número de contêineres usados, por exemplo. Por fim, no problema de alocação de mão de obra em construção civil, os objetivos podem ser a minimização do tempo ocioso dos profissionais, a minimização do tempo médio de conclusão das obras e a maximização de qualidade dos serviços, por exemplo.

A seguir, um pouco da história dos problemas de sequenciamento é apresentada. Segundo Baker e Trietsch (2009), se pensarmos em sequenciamento incluindo problemas de alocação pura, desenvolvimento formal de modelos e técnicas de otimização para teoria moderna de sequenciamento, provavelmente o estudo de problemas de sequenciamento tenha iniciado nos anos anteriores à Segunda Guerra Mundial. Os trabalhos formais sobre as propriedades dos problemas de sequenciamento ganharam reconhecimento na década de 1950, e os livros sobre o assunto começaram a surgir na década de 1960.

Os livros publicados até a década de 1980 abordavam em grande parte modelos determinísticos, e poucos modelos estocásticos eram tratados. Alguns destes livros são: *i*) (Muth e Thompson, 1963), dos autores John F. Muth e Gerald L. Thompson; *ii*) (Conway e Maxwell, 1967), dos autores Richard W. Conway, William L. Maxwell e Louis W. Miller; *iii*) (Baker, 1974), do autor Kenneth R. Baker; *iv*) (Coffman, 1976), do autor Edward G. Coffman; *v*) e (French, 1982), do autor Simon French. O livro de Conway e Maxwell (1967) é um dos primeiros a tratar da complexidade computacional de problemas de sequenciamento, fato pouco comum até então, já que a popularização da pesquisa em complexidade computacional não havia difundido.

A partir da década de 1990 surgiu um número maior de trabalhos abordando modelos estocásticos. Alguns exemplos são os livros de Morton e Pentico (1993) e Pinedo (2001). Até o presente momento, o campo de sequenciamento determinístico está bem desenvolvido, e há um crescente número de pesquisas na área de sequenciamento estocástico (Baker e Trietsch, 2009).

Como pode-se observar ao longo da seção, os problemas de sequenciamento têm grande importância na literatura e aparecem de forma abrangente. Este trabalho trata

exclusivamente da classe de problemas de sequenciamento em máquinas. Daqui em diante, os problemas de sequenciamento são abordados apenas neste contexto.

2.1.2 Notação e classificação

Os problemas de sequenciamento em máquinas possuem um conjunto finito de máquinas e um conjunto finito de tarefas e envolve a alocação das tarefas às máquinas. Comumente o total de máquinas é denotado por m e o total de tarefas por n, usa-se também o índice i para se referir a uma máquina e o índice j para se referir a uma tarefa. A seguir, são dadas algumas características comuns nesta relação de tarefas e máquinas, baseadas nas características descritas por Pinedo (2008).

- Tempo de processamento (p_{ij}): É o tempo necessário para processar a tarefa j na máquina i;
- Data de entrada (r_j): É a data (ou instante de tempo) em que a tarefa j pode iniciar o seu processamento;
- Data de vencimento (d_j): É a data em que a tarefa j deve ser entregue. A entrega após a data de vencimento é permitida, mas o atraso é contabilizado e pode acarretar em uma penalidade;
- Peso (w_j): é um fator de prioridade da tarefa, que denota a importância da tarefa j em relação às outras tarefas. Este peso pode representar o custo real de manter a tarefa no sistema. Em um problema com data de entrega para as tarefas, este custo pode representar o gasto com o atraso da tarefa.

Segundo Graham et al. (1979), um problema de sequenciamento pode ser descrito pela tripla $\alpha|\beta|\gamma|$. Desde então, esta tripla tem sido amplamente usada na literatura. O campo α descreve o ambiente da máquina e contém apenas uma entrada. O campo β pode conter uma única entrada, múltiplas entradas ou nenhuma entrada; este campo fornece detalhes de características e restrições de processamento. Já o campo γ descreve o objetivo a ser minimizado.

A seguir, são apresentados alguns tipos de entradas que podem aparecer nos campos que compõem a tripla $\alpha|\beta|\gamma|$. Estas descrições são baseadas nos trabalhos de Allahverdi (2015); Graham et al. (1979); Pinedo (2008).

Alguns tipos de ambientes de máquinas (α) são dados a seguir.

- Única máquina (1): Neste caso tem-se apenas uma única máquina para processar as tarefas;
- Máquinas paralelas idênticas (P_m): Existem m máquinas idênticas em paralelo. O tempo de processar uma tarefa j em qualquer uma das m máquinas é idêntico.
- Máquinas paralelas uniformes (Q_m): Existem m máquinas paralelas com diferentes velocidades, cada máquina i tem uma velocidade v_i. O tempo para processar a tarefa j em uma máquina i é dado por p_j/v_i. Neste problema, todas as tarefas alocadas a uma máquina i são processadas em uma única velocidade (v_i).
- Máquinas paralelas não relacionadas (R_m): Existem m máquinas paralelas independentes. O tempo de processamento de uma tarefa j em uma máquina i é dado por p_{ij}. Desta maneira, o tempo de processamento de uma tarefa é diferente em cada uma das máquinas.
- *Flow shop* (F_m): Existem m máquinas em série. Diferentemente dos ambientes anteriores em que uma tarefa é processada apenas uma vez e em apenas uma única máquina, no *Flow shop* cada tarefa tem de ser processada por todas as máquinas m seguindo uma rota. Sendo assim, cada tarefa j tem de ser processada primeiro pela máquina 1, depois pela máquina 2 e assim por diante. Após a conclusão em uma máquina, a tarefa vai para a fila na próxima máquina.
- *Job shop* (J_m): Neste ambiente existem m máquinas e cada tarefa tem a sua própria rota predeterminada. Desta maneira, cada tarefa será processada respeitando-se a sua rota.
- *Open shop* (O_m): Neste ambiente existem m máquinas e cada tarefa deve ser processada em cada uma das máquinas em qualquer sequência, isto é, não há restrições com relação à ordem das operações de uma tarefa.

Alguns exemplos de características de processamento e restrições (β) são dadas a seguir.

- Data de entrada (r_j): Esta entrada indica que a tarefa j não pode iniciar o seu processamento antes do tempo r_j. Por outro lado, quando o problema tem a característica data de vencimento (d_j), ela não aparece no campo β. Esta característica fica subentendida na função objetivo do problema, que é representado no campo γ.

- Tempo de preparação dependente da sequência (S_{ijk}): Indica que o tempo de preparação depende da ordem de alocação das tarefas e da máquina na qual estão alocadas. Neste caso, existe um tempo de preparação (S_{ijk}) para processar a tarefa k após a tarefa j na máquina i. O tempo de preparação da primeira tarefa k alocada a uma máquina i é dado por S_{i0k}. Já o tempo de preparação da última tarefa k alocada a uma máquina i é dado por S_{ik0}, que tem geralmente custo zero. Existe também o caso em que o tempo de preparação depende da ordem de alocação das tarefas e independe da máquina; neste caso, ele é representado no campo β por S_{jk}.

- Restrições de precedência ($prec$): Esta entrada indica que existem restrições de precedência em uma máquina ou em um ambiente de máquinas paralelas. Estas restrições indicam que uma ou mais tarefas devem ser concluídas antes que outra tarefa possa iniciar o seu processamento.

- Família de tarefas ($fmls$): Esta entrada indica que as tarefas pertencem a famílias de tarefas diferentes. Dentro de uma mesma família de tarefas os tempos de processamento podem ser diferentes, mas não há tempos de preparação. No entanto, se a máquina mudar o processamento para outra família é necessário um tempo de preparação.

- Restrições de elegibilidade da máquina (M_j): Esta entrada indica que a tarefa j só pode ser processada pelas máquinas do conjunto M_j. Esta entrada pode aparecer em β quando se tem máquinas paralelas idênticas (P_m) em α.

Por último, são apresentados exemplos de funções objetivo que são representadas pelo campo γ na tripla $\alpha|\beta|\gamma|$.

- *Makespan* ($C_{\max}$): O *makespan* é o tempo de conclusão C_j da tarefa j que termina de executar por último, ou seja, o tempo máximo do sequenciamento ($\max(C_1, ..., C_n)$, sendo n o total de tarefas). O *makespan* é um indicador de *throughput* do sistema. A minimização do *makespan* significa encontrar um sequenciamento que leva a uma utilização eficiente de todos os recursos em relação ao tempo de processamento de todas as tarefas.

- Tempo total de conclusão ponderado ($\sum w_j C_j$): É a soma ponderada dos tempos de conclusão (C_j) de todas as tarefas j considerando os seus pesos (w_j). A minimização deste objetivo envolve priorizar a utilização de um recurso em relação a

outro. Na minimização deste objetivo as tarefas mais longas devem ser alocadas aos recursos mais baratos, não necessariamente sendo esse o recurso mais rápido para aquela tarefa.

- *Lateness* máximo (L_{max}): Esta função objetivo mede a maior violação da data de vencimento. O cálculo de *lateness* de uma tarefa j é dado por $L_j = C_j - d_j$, sendo d_j a data de vencimento e C_j o tempo de conclusão. O *lateness* máximo é o $\max(L_1, \cdots, L_n)$.

- Atraso máximo ($T_{\max}$): Esta função mede o atraso máximo, ou seja, $\max(T_1, \cdots, T_n)$. Esta função objetivo não pode conter valores negativos, diferentemente do *lateness* máximo. O atraso de uma tarefa j é dado por $T_j = \max(0, C_j - d_j)$.

- Total de atraso ponderado ($\sum w_j T_j$): É a soma ponderada dos atrasos (T_j) de todas as tarefas j.

A partir dos exemplos anteriores é possível entender a variedade de configurações dos problemas de sequenciamento em máquinas. Este trabalho trata especificamente do problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência. São estudadas duas versões deste problema, uma mono-objetivo e outra multiobjetivo. As seções subsequentes trazem os detalhes destas versões.

2.2 Problema $R_M|S_{ijk}|C_{max}$

2.2.1 Definição

No $R_M|S_{ijk}|C_{\max}$ tem-se um conjunto de tarefas $N = \{1, ..., n\}$ e um conjunto de máquinas $M = \{1, ..., m\}$, com as características a seguir:

(a) Cada tarefa $j \in N$ deve ser alocada a uma única máquina $i \in M$;

(b) O tempo para processar $j \in N$ em uma dada máquina $i \in M$ é dado por p_{ij};

(c) Tem-se um tempo de preparação S_{ijk} para processar a tarefa $k \in N$ após a tarefa $j \in N$ na máquina $i \in M$, nesta ordem;

(d) O objetivo é alocar todas as tarefas de N nas máquinas de M buscando minimizar o *makespan*.

Para facilitar o entendimento do problema, a seguir, são apresentados os dados de uma instância com 7 tarefas e 2 máquinas. A Tabela 2.1 exibe os tempos de processamento das tarefas nas máquinas $M1$ e $M2$. Já as tabelas 2.2a e 2.2b, apresentam os tempos de preparação nas máquinas $M1$ e $M2$, respectivamente.

Tabela 2.1: Tempo de processamento nas máquinas $M1$ e $M2$.

	M1	M2
1	20	4
2	25	21
3	28	14
4	17	32
5	43	38
6	9	23
7	58	52

Tabela 2.2: Tempos de preparação

(a) Máquina $M1$.

$M1$	1	2	3	4	5	6	7
1	2	1	8	1	3	9	6
2	4	7	6	3	7	8	4
3	7	3	4	2	3	5	3
4	3	8	3	5	5	2	2
5	8	3	7	9	6	5	7
6	8	8	1	2	2	1	9
7	1	4	5	2	3	5	1

(b) Máquina $M2$.

$M2$	1	2	3	4	5	6	7
1	3	4	6	5	9	3	2
2	1	2	6	2	7	7	5
3	2	6	4	6	8	1	4
4	5	7	8	3	2	5	6
5	7	9	5	7	6	4	8
6	9	3	5	4	9	8	3
7	3	2	6	1	5	6	7

A Figura 2.1 ilustra uma possível solução para a instância. A parte hachurada representa os tempos de preparação. Na máquina $M1$ estão alocadas as tarefas 2, 1 e 7, nesta ordem. E na máquina $M2$ estão alocadas as tarefas 5, 4, 6 e 3, nesta ordem.

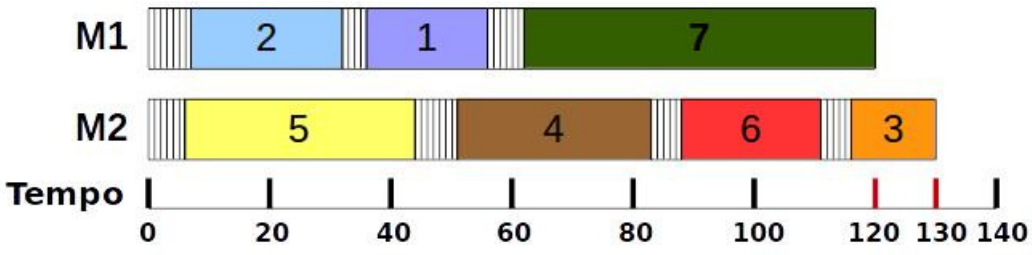


Figura 2.1: Exemplo de um sequenciamento com 2 máquinas e 7 tarefas.

O custo da máquina $M1$ é dado por $C_{M1} = S_{102} + p_{12} + S_{121} + p_{11} + S_{117} + p_{17} =$ 120 unidades de tempo. De maneira semelhante, o custo da máquina $M2$ é dado por $C_{M2} = S_{205} + p_{25} + S_{254} + p_{24} + S_{246} + p_{26} + S_{263} + p_{23} = 130$ unidades de tempo. O *makespan* desta solução é 130 unidades de tempo, ou seja, o tempo gasto pela máquina que termina sua execução por último, neste caso a máquina $M2$.

2.2.2 Revisão da literatura

Esta seção trata a revisão bibliográfica do problema $R_M|S_{ijk}|C_{max}$. Como exposto na Seção 2.1.1, os problemas de sequenciamento em máquinas vem sendo muito abordados na literatura desde a década de 60. Devido ao grande número de trabalhos da literatura tratando problemas de sequenciamento em máquinas, o foco desta seção é apresentar alguns dos principais trabalhos relacionados ao problema $R_M|S_{ijk}|C_{max}$ nas últimas duas décadas. Nesta seção é utilizado como referência o trabalho de Allahverdi (2015), que aborda 500 trabalhos dos últimos anos envolvendo problemas de sequenciamento de máquinas com tempos de preparação.

Na Tabela 2.3 são apresentados alguns trabalhos importantes que tratam problemas similares ao $R_M|S_{ijk}|C_{\max}$. Todos estes trabalhos envolvem problemas de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação, mas cada um deles possui características que os diferem do problema $R_M|S_{ijk}|C_{\max}$.

Na Tabela 2.4 são apresentados alguns dos principais trabalhos abordando a resolução do problema $R_M|S_{ijk}|C_{\max}$. Nas últimas duas décadas foram propostos na literatura dois conjuntos diferentes de instâncias para o problema, um no ano de 2006 e o outro em 2011. A maioria dos trabalhos da literatura utilizam um destes conjuntos para validar as abordagens propostas.

Tabela 2.3: Trabalhos abordando problemas similares ao $R_M|S_{ijk}|C_{\max}$.

Trabalho	Características diferentes do $R_M\|S_{ijk}\|C_{\max}$	Abordagem
(Weng et al., 2001)	Tempos de preparação dependentes somente da máquina e o objetivo é minimizar a média ponderada do tempo de conclusão	Sete heurísticas construtivas
(Kim et al., 2002)	Sequenciamento em lotes e o objetivo é minimizar o tempo total de atraso	Algoritmo *Simulated Annealing*
(Kim et al., 2003)	Sequenciamento em lotes com datas de entrega e o objetivo é minimizar o tempo total de atraso	Três heurísticas construtivas e um *Simulated Annealing*
(Rocha et al., 2008)	Datas de entrega para as tarefas	*Branch-and-bound* e um GRASP
(Logendran et al., 2007)	Entrada dinâmica das tarefas e disponibilidade dinâmica das máquinas e o objetivo é minimizar o tempo total de atraso ponderado	Seis variações do Busca Tabu
(Randall e Kurz, 2007)	Datas de entrega das tarefas e o objetivo é minimizar o atraso total ponderado	Algoritmo Genético adaptativo
(Pereira Lopes e de Carvalho, 2007)	Disponibilidade das máquinas e data de entrega Das tarefas	Algoritmo *Branch-and-price*
(Chen, 2009)	Datas de entrega das tarefas e o objetivo é minimizar o atraso total ponderado	*Simulated Annealing*
(Paula et al., 2010)	Data de entrega das tarefas e o objetivo é minimizar o atraso total ponderado	Algoritmo baseado em relaxação Lagrangiana
(Lee et al., 2013)	Data de entrega das tarefas e o objetivo é minimizar o total de atraso	Algoritmo Busca Tabu
(Lin e Hsieh, 2014)	Data de entrega e prioridade das tarefas e o objetivo é minimizar o total de atraso ponderado	*Iterated hybrid metaheuristic*
(Caniyilmaz et al., 2015)	Restrição das máquinas e data de entrega das tarefas, o objetivo é minimizar a soma do *makespan* e do atraso	Algoritmo Colônia de Abelhas
(Zeidi e MohammadHosseini, 2015)	Data de entrega das tarefas e custos para entrega antecipada e atrasada das tarefas, o objetivo é minimizar a soma total dos atrasos e antecipações	Algoritmo Genético combinado com *Simulated Annealing*

Tabela 2.4: Trabalhos abordando o problema $R_M|S_{ijk}|C_{\max}$.

Trabalho	Abordagem	Instâncias
(Al-Salem, 2004)	Algoritmo *Partitioning Heuristic*	-
(Rabadi et al., 2006)	Conjunto de instâncias disponibilizado em (Scheduling Research, 2005), um modelo matemático e uma *Metaheuristic for Randomized Priority Search*	De Rabadi et al. (2006)
(Helal et al., 2006)	Algoritmo Busca Tabu	De Rabadi et al. (2006)
(Arnaout et al., 2010)	Algoritmo Colônia de Formigas	De Rabadi et al. (2006)
(Ying et al., 2010)	Algoritmo *Restricted Simulated Annealing*	De Rabadi et al. (2006)
(Chang e Chen, 2011)	Algoritmo Genético e um *Simulated Annealing*	De Rabadi et al. (2006)
(Fleszar et al., 2011)	Algoritmo que combina o *Multi-start* e o *Variable Neigborhood Descendent* - VND	De Rabadi et al. (2006)
(Vallada e Ruiz, 2011)	Conjunto de instâncias disponibilizado em (SOA, 2011), um modelo matemático e dois algoritmos Genéticos	De Vallada e Ruiz (2011)
(Lin e Ying, 2014)	Algoritmo Colônia de Abelhas	De Rabadi et al. (2006)
(Arnaout et al., 2014)	Evolução do Colônia de Formigas de Arnaout et al. (2010)	De Rabadi et al. (2006)
(Avalos-Rosales et al., 2015) e (Avalos-Rosales et al., 2013)	Melhoria no modelo matemático de Vallada e Ruiz (2011)	De Vallada e Ruiz (2011)
(Haddad et al., 2014), (Cota et al., 2014a), (Cota et al., 2014b) e (Haddad et al., 2015)	Combinações dos algoritmos VND, *Iterated Local Search* e *Path Relinking*	De Vallada e Ruiz (2011)
(Coelho et al., 2016b)	Estratégias evolutivas guiadas por estruturas de vizinhança	De Vallada e Ruiz (2011)
(Santos et al., 2016)	Estudo de métodos estocásticos de buscas locais	De Vallada e Ruiz (2011)
(Tran et al., 2016)	Algoritmos baseados em decomposição de Benders e *branch-and- check*	De Rabadi et al. (2006)

2.2.2.1 Estado da arte

Dentre os trabalhos da Tabela 2.4, destacam-se alguns importantes no cenário atual. Estes trabalhos são descritos a seguir.

Em Arnaout et al. (2014), os autores propõem um algoritmo Colônia de Formigas (ACOII) que é a uma evolução do algoritmo proposto por alguns dos mesmos autores em Arnaout et al. (2010). Para avaliar o algoritmo é utilizado o conjunto de instâncias de Rabadi, disponibilizado em Scheduling Research (2005). Os resultados do ACOII são comparados aos de diversos outros algoritmos da literatura, como os propostos em Arnaout et al. (2010); Helal et al. (2006); Rabadi et al. (2006); Ying et al. (2010). Nos experimentos computacionais o algoritmo ACOII obteve o melhor desempenho.

Em Avalos-Rosales et al. (2013, 2015), os autores propõem melhorias no modelo matemático proposto por Vallada e Ruiz (2011). Essas melhorias tornaram a implementação do modelo muito mais eficiente que a versão anterior de Vallada e Ruiz (2011). No trabalho é mostrado que a implementação do novo modelo matemático é capaz de resolver instâncias com até 60 tarefas de maneira ótima em um tempo médio de execução de uma hora (como mencionado no Capítulo 1).

Nos trabalhos Cota et al. (2014a,b); Haddad et al. (2014) e Haddad et al. (2015) são propostos diferentes algoritmos que combinam as meta-heurísticas *Variable Neigborhood Descent*, *Iterated Local Search* e *Path Relinking*. O algoritmo de melhor desempenho é o AIRP proposto em Cota et al. (2014a). Nesse trabalho o algoritmo AIRP é executado para o conjunto de instâncias de Vallada & Ruiz, disponibilizado em (SOA, 2011). Os resultados do AIRP são comparados aos do algoritmo GA2 (Vallada e Ruiz, 2011), e o AIRP obteve melhor desempenho.

Em Santos et al. (2016), os autores apresentam um estudo sobre métodos de buscas locais estocásticos, com calibração prévia dos parâmetros utilizando o pacote IRace (López-Ibáñez et al., 2011). Os algoritmos tiveram os parâmetros calibrados apenas para o conjunto de instâncias de Vallada & Ruiz, disponibilizado em (SOA, 2011). Dentre os algoritmos propostos, o SA é o que obteve o melhor desempenho.

Em Tran et al. (2016), são propostos dois algoritmos híbridos baseados na decomposição de Benders e no *Branch-and-Check*. Nesse trabalho os dois algoritmos híbridos são executados para um subconjunto de instâncias do conjunto de Rabadi e os resultados são comparados aos dos algoritmos propostos em Helal et al. (2006); Rabadi et al. (2006). O algoritmo híbrido baseado no *Branch-and-Check* encontrou os melhores resultados.

2.2.3 Modelos matemáticos da literatura

Nesta seção são revisados dois modelos matemáticos da literatura, especificamente o modelo de Vallada e Ruiz (2011) e a melhoria proposta nesse modelo por Avalos-Rosales et al. (2013, 2015). O modelo matemático de Rabadi et al. (2006) não é analisado porque é muito parecido com o de Vallada e Ruiz (2011).

2.2.3.1 Modelo de Vallada e Ruiz (2011)

Em Vallada e Ruiz (2011) é proposta uma formulação de programação linear inteira mista para o $R_M|S_{ijk}|C_{\max}$. Na Tabela 2.5 é dada a notação usada nesta formulação matemática.

Tabela 2.5: Notação da formulação matemática de Vallada e Ruiz (2011).

Parâmetros:
$M = \{1, ..., m\}$: conjunto de máquinas não relacionadas, com um total de m máquinas;
$N = \{1, ..., n\}$: conjunto de tarefas, com um total de n tarefas;
$N_0 = N \cup \{0\}$: conjunto de tarefas com adição da tarefa 0 (tarefa fictícia);
p_{ij}: tempo de processamento da tarefa j na máquina i;
S_{ijk}: tempo de preparação requerido para processar a tarefa k imediatamente após a tarefa j na máquina i;
B: constante suficientemente grande.
Variáveis de decisão:
x_{ijk}: 1 se a tarefa j é alocada imediatamente antes da tarefa k na máquina i e 0 caso contrário;
C_{ij}: tempo de conclusão da tarefa j na máquina i;
$C_{\max}$: tempo máximo de processamento das máquinas (*makespan*).

Neste modelo uma tarefa fictícia 0 é alocada na primeira posição de cada máquina. Esta tarefa fictícia tem tempo de processamento $p_{i0} = 0 \ \forall i \in M$ e tempo de preparação $S_{i0k} = 0 \ \forall i \in M, \forall k \in N$. O modelo matemático é dado pelas Equações (2.1) até (2.10):

$$\min C_{\max} \tag{2.1}$$

Sujeito a:

$$\sum_{i=1}^{m}\sum_{\substack{j=0\\j\neq k}}^{n} x_{ijk} = 1 \qquad \forall k \in N \tag{2.2}$$

$$\sum_{i=1}^{m}\sum_{\substack{k=1\\j\neq k}}^{n} x_{ijk} \leq 1 \qquad \forall j \in N \tag{2.3}$$

$$\sum_{k=1}^{n} x_{i0k} \leq 1 \qquad \forall\, i \in M \tag{2.4}$$

$$\sum_{\substack{h=0\\h\neq k\\h\neq j}}^{n} x_{ihj} \geq x_{ijk} \qquad \forall j,k \in N, \forall i \in M \tag{2.5}$$

$$C_{ik} + B\left(1 - x_{ijk}\right) \geq C_{ij} + S_{ijk} + p_{ik} \qquad \forall j \in N_0, \forall k \in N, j \neq k, \forall i \in M \tag{2.6}$$

$$C_{i0} = 0 \qquad \forall i \in M \tag{2.7}$$

$$C_{ij} \geq 0 \qquad \forall j \in N, \forall i \in M \tag{2.8}$$

$$C_{\max} \geq C_{ij} \qquad \forall j \in N, \forall i \in M \tag{2.9}$$

$$x_{ijk} \in \{0,\, 1\} \qquad \forall j \in N_0, \forall k \in N, j \neq k, \forall i \in M \tag{2.10}$$

A Equação (2.1) tem por objetivo minimizar o *makespan*. As restrições (2.2) garantem que cada tarefa seja atribuída a uma única máquina e que cada tarefa tem uma tarefa predecessora. Nas restrições (2.3) limita-se a 1 o número máximo de tarefas sucessoras de uma dada tarefa. Do mesmo modo, nas restrições (2.4) limita-se a 1 o número máximo de tarefas sucessoras de cada tarefa fictícia. As restrições (2.5) garantem que se uma tarefa j é imediatamente anterior a uma tarefa k, deve existir uma tarefa imediatamente anterior à tarefa j na mesma máquina. Já as restrições (2.6) servem para controlar os tempos de conclusão das tarefas nas máquinas. Se a tarefa k está alocada imediatamente após a tarefa j na máquina i ($x_{ijk} = 1$), o tempo de conclusão relacionado, C_{ik}, deve ser maior ou igual ao tempo de conclusão da tarefa j, C_{ij}, somado ao tempo de preparação entre j e k e o tempo de processamento de k. Caso $x_{ijk} = 0$, a constante B fará com que essas restrições sejam redundantes. As restrições (2.7) e (2.8) definem os tempos de conclusão igual a 0 para as tarefas fictícias e não-negativas para as demais tarefas, respectivamente. As restrições (2.9) definem o tempo máximo

de conclusão (*makespan*). Finalmente, as restrições (2.10) definem que as variáveis de decisão são binárias.

2.2.3.2 Modelo de Rosales (2015)

Em Avalos-Rosales et al. (2013, 2015) os autores propõem mudanças no modelo matemático de Vallada e Ruiz (2011). Neste modelo os resolvedores conseguem efetuar cortes mais eficientes, tornando-o mais rápido que o de Vallada e Ruiz (2011).

A variável C_{ij} no modelo de Vallada e Ruiz (2011) é substituída no modelo de Avalos-Rosales et al. (2013, 2015) pelas duas variáveis abaixo:

- C_j: tempo de conclusão da tarefa j;
- O_i: tempo de conclusão da máquina i.

No modelo de Avalos-Rosales et al. (2013, 2015) uma tarefa fictícia 0 também é considerada no final de cada máquina com as mesmas propriedades descritas anteriormente ($p_{i0} = 0$ e $S_{i0k} = 0$, para $i \in M$ e $k \in N$). Com isso, as restrições (2.5) são trocadas pelas restrições (2.11):

$$\sum_{\substack{k=0 \\ k \neq j}}^{n} x_{ijk} - \sum_{\substack{h=0 \\ h \neq j}}^{n} x_{ihj} = 0 \qquad \forall j \in N, \forall i \in M \tag{2.11}$$

Como a variável C_j não é indexada pela máquina, as restrições (2.6) são substituídas pelas restrições (2.12), e as restrições (2.7) e (2.8) são substituídas pelas restrições (2.13).

$$C_k - C_j + B\left(1 - x_{ijk}\right) \geq S_{ijk} + p_{ik} \qquad \forall j \in N_0, \forall k \in N, j \neq k, \forall i \in M \tag{2.12}$$

$$C_0 = 0 \tag{2.13}$$

No modelo de Avalos-Rosales et al. (2013, 2015) a variável O_i é responsável por armazenar os tempos de conclusão das máquinas $i \in M$; com isso, as restrições (2.9) são substituídas pelas restrições (2.14).

$$C_{\max} \geq O_i \qquad \forall i \in M \tag{2.14}$$

As restrições (2.15) são adicionadas ao modelo a fim de definir os tempos de conclusão de cada máquina (O_i).

$$\sum_{j=0}^{m} \sum_{\substack{k=1 \\ k \neq j}}^{n} (S_{ijk} + p_{ik}) \times x_{ijk} = O_i \qquad \forall i \in M \tag{2.15}$$

2.3 Problema $R_M|S_{ijk}|(C_{\max}, \text{TEC})$

Esta seção trata o problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência com os objetivos de minimizar o *makespan* e o consumo total de energia (ou $R_M|S_{ijk}|(C_{\max}, TEC)$). Esta abordagem do UPMSP-ST ainda não é encontrada na literatura.

2.3.1 Definição

O problema $R_M|S_{ijk}|(C_{\max}, TEC)$ possui atributos adicionais se comparado ao $R_M|S_{ijk}|C_{\max}$. Esses atributos são necessários para o cálculo do consumo de energia e foram inspirados no trabalho de Mansouri et al. (2016a). Neste problema tem-se as características a seguir:

(a) Cada tarefa deve ser alocada a uma única máquina;

(b) Tem-se um tempo de processamento para processar cada tarefa em uma máquina;

(c) Existe um tempo de preparação para calibrar cada máquina para processar uma tarefa. O tempo de preparação depende da ordem de alocação das tarefas na máquina;

(d) Há um conjunto discreto de velocidades de processamento para processar uma tarefa em uma máquina;

(e) Cada máquina possui um consumo de potência de acordo com a velocidade de operação para processar uma tarefa na máquina;

(f) Os objetivos são alocar todas as tarefas nas máquinas buscando minimizar o *makespan* e o consumo total de energia elétrica (TEC).

Os objetivos *makespan* e consumo total de energia são de suma importância na formulação do $R_M|S_{ijk}|(C_{\max}, TEC)$, e têm natureza conflitante. A minimização do *makespan* leva à maximização da produção em altas velocidades de operação, o que conduz a um maior consumo de energia no sequenciamento. Por outro lado, a minimização do TEC passa pela produção em menores velocidades de operação ou nas máquinas de menor consumo de potência, que não necessariamente são aquelas que processam a tarefa no menor tempo, para que o consumo de energia seja menor. A maioria dos motores elétricos utilizados na indústria são máquinas de indução. Estes motores têm alta eficiência, baixos custos de manutenção e controle de velocidade variável permitido por inversores de frequência modernos (método de transmissão de frequência variável). Em aplicações de velocidade variável, a mudança de frequência é um método comum para controlar a velocidade de um motor de indução. Particularmente com os dispositivos eletrônicos disponíveis, uma frequência de fornecimento variável para o estator pode ser usada para controlar as velocidades. No entanto, temos que garantir que a relação tensão-frequência permaneça constante para manter o fluxo constante na máquina. Portanto, o uso de uma máquina em uma velocidade maior leva a um aumento correspondente na entrada de energia (Sen, 2013; Wildi, 2013). Essas características são assumidas na formulação do problema $R_M|S_{ijk}|(C_{\max}, TEC)$.

Para ajudar no entendimento do problema, a seguir, é utilizada uma instância com 2 máquinas, 6 tarefas e 3 velocidades de operação, sendo que cada máquina possui potências nominais diferentes. A Figura 2.2 apresenta a solução ótima do sequenciamento analisando o problema com o único objetivo de minimizar o *makespan*. A parte hachurada representa os tempos de preparação. O *makespan* encontrado é de 142 unidades de tempo e o consumo total de energia é de 138 unidades de energia.

A seguir, é executado o mesmo problema com o único objetivo de minimizar o consumo total de energia. A solução ótima encontrada possui *makespan* de 407 unidades de tempo e o consumo total de 81 unidades de energia.

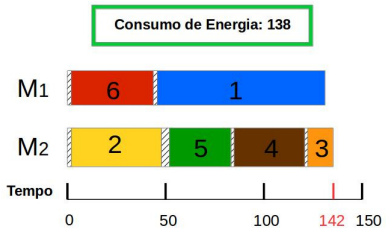


Figura 2.2: Exemplo de um sequenciamento ótimo analisando somente o objetivo *makespan*

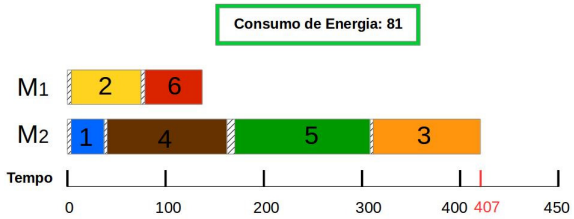


Figura 2.3: Exemplo de um sequenciamento ótimo analisando somente o objetivo de consumo total de energia

Pode-se verificar que existe grande variabilidade no valor das funções objetivo das figuras 2.2 e 2.3, e que o sequenciamento em ambas as figuras são diferentes. Este exemplo sugere que os objetivos são conflitantes e que é importante analisar o consumo total de energia no UPMSP-ST.

2.3.2 Revisão da literatura

Nesta seção é apresentada uma revisão bibliográfica do uso sustentável de energia em problemas de sequenciamento, também chamado sequenciamento verde (ou *green scheduling* no idioma inglês). Este tema ainda é pouco encontrado na literatura em problemas de sequenciamento de máquinas. A seguir, são apresentados alguns trabalhos importantes da literatura que serviram de inspiração para a nova abordagem do problema proposta neste trabalho, a $R_M|S_{ijk}|(C_{\max}, TEC)$.

Em Zhang et al. (2014) é desenvolvido um modelo matemático indexado no tempo para a resolução de um problema de sequenciamento *flow shop* com os objetivos de minimizar os custos de energia e a emissão de carbono. Nesse trabalho é analisada a variação nos custos da energia durante o tempo de consumo.

No trabalho de Ding et al. (2016) os autores tratam um problema de sequenciamento de máquinas *flow shop* em que os objetivos são minimizar o *makespan* e as emissões totais de carbono. Para a sua resolução é proposto um algoritmo *Iterated Greedy* (Ruiz e Stützle, 2005) multiobjetivo.

Em Mansouri et al. (2016a) é tratado um problema de sequenciamento *flow shop* de duas máquinas com tempos de preparação em que os objetivos são minimizar o *makespan* e o consumo total de energia. Para a sua resolução são propostos um modelo matemático e uma heurística construtiva para análise de *trade-off* entre os objetivos.

Em Wang et al. (2016), os autores tratam um problema de sequenciamento em uma única máquina em lote com tarefas de diferentes tipos, com variações no preço de energia por tempo de uso e diferentes taxas de consumo de energia da máquina. Os objetivos são minimizar o *makespan* e os custos totais de energia. Os autores propõem um modelo matemático e a implementação do método exato ϵ-restrito para resolver o problema.

Um problema *job shop* bi-objetivo é tratado em Liu et al. (2016). Nesse problema os objetivos são minimizar o consumo total de energia não processada e o atraso total ponderado. Para a sua resolução, os autores propõem um algoritmo genético multiobjetivo baseado no NSGA-II (Deb et al., 2002).

Em Mansouri e Aktas (2016) é abordado um problema de sequenciamento *flow shop* de duas máquinas com os objetivos de minimizar o *makespan* e o consumo de energia. Os autores propõem heurísticas construtivas e um algoritmo genético multiobjetivo para resolver o problema.

Em Dooren et al. (2017) é tratado um problema de otimização de energia multimáquina com custo de energia diferente por horário de uso, de acordo com a demanda. O objetivo neste problema é minimizar o custo total de energia. Os autores propõem uma heurística construtiva e um algoritmo *late acceptance hill climbing* para resolver o problema.

Um problema de sequenciamento de uma única máquina em lotes com custo de energia diferente por horário de uso é tratado em Cheng et al. (2017). Os objetivos neste problema são minimizar o *makespan* e o custo total da eletricidade. Os autores propõem um modelo matemático, que é resolvido usando o método ϵ-restrito.

A resolução do $R_M|S_{ijk}|(C_{\max}, TEC)$ nesta tese busca encorajar o uso de abordagens no contexto de sequenciamento verde, além de fornecer um modelo matemático e métodos eficientes que possam ser usados em casos práticos na indústria.

2.4 Conclusão

Neste capítulo é apresentada a definição e um pouco da história dos problemas de sequenciamento. Os problemas de sequenciamento vêm sendo tratados desde pouco antes da Segunda Guerra Mundial. Os principais trabalhos na literatura começaram a surgir da década de 1960. Desde então, os problemas de sequenciamento vêm sendo amplamente tratados na literatura, dada a sua importância nas indústrias e os desafios que existem na sua resolução. A notação e classificação dos problemas de sequenciamento em máquinas também é apresentada neste capítulo. Com essa notação é possível identificar de maneira simples as características de um problema desta natureza.

Este trabalho trata duas versões do problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência, uma monoobjetivo e outra multiobjetivo. Neste capítulo é apresentada a definição destas duas versões do problema e uma revisão bibliográfica de trabalhos importantes da literatura que tratam o problema em questão e de outros relacionados.

Capítulo 3

Métodos para problemas de sequenciamento em máquinas

3.1 Introdução

A otimização é uma área da pesquisa operacional que utiliza uma abordagem científica para apoiar a tomada de decisões procurando pela forma mais eficiente de projetar e operar um dado sistema. Esta forma mais eficiente é representada pela melhor combinação de valores para as variáveis do problema (ou solução ótima), considerando seus objetivos e restrições de projeto e operação (Arenales et al., 2015; Winston, 2004).

Na otimização mono-objetivo tem-se um único objetivo a ser maximizado ou minimizado; com isso, uma solução ótima é claramente definida. Uma solução $s^\star$ é ótima caso não exista nenhuma outra solução possível com função de custo melhor que $s^\star$.

A seguir, é apresentada a formulação de um problema de otimização mono-objetivo de minimização (Luenberger e Ye, 2008).

$$\min f(x) \tag{3.1}$$

Sujeito a:

$$g_i(x) \leq 0; \qquad \forall i = 1, ..., p \tag{3.2}$$

$$h_j(x) = 0; \qquad \forall j = 1, ..., q \tag{3.3}$$

$$x \in \mathcal{R}^n \tag{3.4}$$

Nesta formulação, x é um vetor n-dimensional de incógnitas, $x = (x_1, x_2, ..., x_n)$. As funções f, g_i (com $i = 1, 2, ..., p$) e h_j (com $j = 1, 2, ..., q$) são funções de valores reais das variáveis $x_1, x_2, \cdots, x_n$. Tem-se um total de p funções do tipo g_i e um total de q funções do tipo h_j. A função f é a função objetivo do problema e as equações e inequações são as restrições do problema.

Na otimização multiobjetivo tem-se um conjunto de dois ou mais objetivos a serem minimizados e/ou maximizados. Grande parte dos problemas práticos nas indústrias envolvem o alcance de diversos objetivos simultaneamente, respeitando-se um conjunto de restrições. Os objetivos são tratados separadamente como objetivos não comparáveis. Desta maneira, diferentemente da otimização mono-objetivo, em que se busca uma única solução, na otimização multiobjetivo busca-se um conjunto de soluções para representar os compromissos dos objetivos a serem otimizados (Coello et al., 2006; Deb, 2009; Fonseca e Fleming, 1993; Freitas, 2013).

As soluções que representam este compromisso formam um conjunto de soluções chamado conjunto Pareto-ótimo. Segundo Pareto (1896), um vetor (ou conjunto) de soluções é Pareto-ótimo se não existir um outro vetor viável que possa melhorar algum objetivo, sem piorar ao menos um outro objetivo.

A seguir, é dada a formulação de um problema de minimização multiobjetivo.

$$\min(f_1(x), \cdots, f_r(x)) \tag{3.5}$$

Sujeito a:

$$g_i(x) \leq 0; \qquad \forall i = 1, ..., p \tag{3.6}$$

$$h_j(x) = 0; \qquad \forall j = 1, ..., q \tag{3.7}$$

$$x \in \mathcal{R}^n \tag{3.8}$$

Nesta formulação, x é um vetor n-dimensional de incógnitas $x = (x_1, x_2, \cdots, x_n)$. As funções f_k (com $k = 1, 2, \cdots, r$), g_i (com $i = 1, 2, \cdots, p$) e h_j (com $j = 1, \cdots, q$) são

funções de valores reais das variáveis $x_1, x_2, \cdots, x_n$. Os objetivos a serem minimizados são definidos pelas funções f_k, sendo um total de r objetivos. Tem-se um total de p funções do tipo g_i e um total de q funções do tipo h_j. Além disso, as Equações (3.6) e (3.7) definem as restrições de desigualdade e igualdade para o problema, respectivamente.

Nos problemas de otimização multiobjetivo geralmente não se tem uma única solução que otimize de maneira ótima todos os objetivos simultaneamente, já que estes objetivos são comumente conflitantes (Freitas, 2013). Com isso, busca-se na otimização multiobjetivo um conjunto de soluções Pareto-ótimo (ou soluções não dominadas). Para comparar as soluções em problemas de otimização multiobjetivo é utilizado o critério de dominância. Uma solução $x_1 \in X$ Pareto domina uma solução $x_2 \in X$ se:

$$\begin{aligned} &f_i(x_1) \leq f_i(x_2)\ \forall i \in \{1, 2, 3, \cdots, r\} \text{ e} \\ &f_j(x_1) < f_j(x_2) \text{ para algum } \forall j \in \{1, 2, 3, \cdots, r\} \end{aligned} \tag{3.9}$$

A solução x_1 é dita Pareto-ótimo se não há outra solução factível que a domina. Considere o exemplo de um problema de sequenciamento em máquinas com duas funções objetivo de minimização, f_1 e f_2, no qual f_1 é o consumo de energia elétrica e f_2 é o total de atraso. Neste mesmo exemplo existem duas soluções factíveis x_1 e x_2 com os seguintes valores de funções objetivo $f_1(x_1) = 100$, $f_2(x_1) = 50$, $f_1(x_2) = 110$ e $f_2(x_2) = 70$. Observando este exemplo é verificado que a solução x_1 domina a solução x_2. Caso não exista nenhuma outra solução factível no problema que domine a solução x_1, ela é dita Pareto-ótimo ou não dominada.

O processo de otimização multiobjetivo tem como objetivo encontrar uma aproximação representativa do conjunto Pareto-ótimo. A partir destas soluções, o analista responsável pela tomada de decisão pode ponderar os objetivos globais do problema e escolher uma entre as soluções eficientes encontradas (Arroyo e Armentano, 2005).

Nas próximas seções são apresentadas técnicas computacionais para a resolução de problemas de otimização.

3.2 Heurísticas

Heurísticas podem ser definidas como técnicas inspiradas em processos intuitivos que procuram boas soluções em tempo computacional aceitável para a tomada de decisão. No entanto, não existe garantia de otimalidade da solução, nem garantia de quão próximo está da solução ótima.

As próximas subseções apresentam dois tipos clássicos de heurísticas em problemas de otimização.

3.2.1 Heurísticas construtivas

Dada a estrutura de dados escolhida para representar uma solução de um problema de otimização, a heurística construtiva tem por objetivo construir uma solução inicial para o problema. Esta solução é construída elemento a elemento, sendo um por vez.

A escolha do elemento que será inserido a cada passo é definida por uma função de avaliação da heurística, a qual depende do problema abordado. Em heurísticas construtivas clássicas é geralmente utilizada uma função de avaliação do tipo gulosa. Neste tipo de função é estimado o benefício de inserção de cada elemento, e somente o elemento que produz o maior benefício é inserido a cada passo na solução parcial.

No Algoritmo 3.1 é apresentado o pseudocódigo de uma heurística construtiva de minimização que utiliza uma função de avaliação gulosa. O parâmetro $g(.)$ é a função de avaliação e o elemento t_{melhor} é aquele que possui o menor valor segundo a função $g(.)$, ou seja, que produz o maior benefício. Para uma heurística de maximização a mudança é simples, troca-se a condição $\arg\min\{g(t) \mid t \in C\}$ por $\arg\max\{g(t) \mid t \in C\}$.

Em heurísticas construtivas clássicas também são utilizadas funções de avaliação do tipo parcialmente gulosa. Neste tipo de função de avaliação é inserido um nível de aleatoriedade na escolha dos elementos. Desta maneira, a cada chamada da heurística uma solução diferente é gerada.

3.2.2 Buscas locais

As buscas locais são também chamadas heurísticas de refinamento. Elas têm por finalidade refinar uma solução previamente gerada. Para realizar o refinamento de soluções

Algoritmo 3.1: Heurística Construtiva Gulosa

Entrada: Função de avaliação $g(.)$
Saída: Solução s construída
1 Inicialize o conjunto C de elementos candidatos;
2 $s \leftarrow \emptyset$;
3 **enquanto** $(|C| > 0)$ **faça**
4 $\quad t_{melhor} \leftarrow \arg\min\{g(t) \mid t \in C\}$;
5 $\quad s \leftarrow s \cup \{t_{melhor}\}$;
6 $\quad$ Atualize o conjunto C de elementos candidatos;
7 **fim**
8 **Retorne** s;

são utilizados movimentos que dão origem à noção de vizinhanças. Um movimento é uma modificação em uma solução corrente para gerar uma nova solução diferente, denominada vizinha da solução corrente. Uma vizinhança desta solução é formada por um conjunto de movimentos de um mesmo tipo aplicados sobre ela.

Nas buscas locais, a cada iteração caminha-se, de vizinho para vizinho, de acordo com a vizinhança adotada, até que se chegue a um critério de parada.

Duas heurísticas clássicas de refinamento são apresentadas nas subseções a seguir.

3.2.2.1 Método da descida/subida

Este método é chamado de descida em problemas de minimização e de subida em problemas de maximização. A ideia desta técnica é partir de uma solução inicial qualquer e a cada iteração analisar todos os seus vizinhos possíveis, movendo para aquele que tenha a melhor avaliação e que represente uma melhora no valor da solução corrente. Esta técnica é comumente referenciada na literatura inglesa por *best improvement method*, pelo fato de analisar todos os vizinhos a cada iteração e escolher o melhor.

O método de descida para um problema de minimização é apresentado pelo Algoritmo 3.2. Ele recebe como parâmetros uma solução inicial s, uma função de avaliação f e uma dada vizinhança $\mathcal{N}(.)$.

Algoritmo 3.2: Método da descida

Entrada: Função de avaliação $f(.)$, vizinhança $\mathcal{N}(.)$, solução s
Saída: Solução s refinada

```
1 V ← {s' ∈ N(s) | f(s') < f(s)};
2 enquanto (|V| > 0) faça
3 |   Selecione s' ∈ V, sendo s' = arg min{f(s') | s' ∈ V};
4 |   s ← s';
5 |   V ← {s' ∈ N(s) | f(s') < f(s)};
6 fim
7 retorna s;
```

3.2.2.2 Método de primeira melhora

O método de subida/descida gera grande esforço computacional, já que a cada iteração é realizada uma pesquisa exaustiva por todos os vizinhos possíveis. Com o objetivo de evitar esta grande exploração da vizinhança é apresentado o método de primeira melhora. Na literatura inglesa ele é referenciado por *first improvement method*. A ideia deste método é interromper a exploração de uma dada vizinhança assim que um vizinho melhor é encontrado. Somente no pior caso todos os vizinhos são pesquisados. Contudo, assim como no método de subida/descida, o algoritmo fica preso no primeiro ótimo local encontrado.

3.3 Meta-heurísticas

Meta-heurísticas são procedimentos que podem ser aplicados a problemas de otimização de diferentes tipos, diferentemente das heurísticas. As meta-heurísticas são técnicas de caráter geral e possuem mecanismos que buscam evitar a parada prematura em ótimos locais ainda distantes de um ótimo global (Glover e Kochenberger, 2003; Siarry, 2016).

As meta-heurísticas se diferenciam basicamente pelos mecanismos utilizados para não ficarem presos em ótimos locais. Elas podem ser divididas em duas categorias, de acordo com o princípio usado para explorar o espaço de soluções: busca local e busca populacional.

Nas baseadas em buscas locais o espaço de soluções é explorado por meio de estruturas de vizinhanças a partir de uma única solução corrente. Alguns exemplos de meta-heurísticas deste tipo são: *Variable Neighborhood Search* (Hansen et al., 2008),

Iterated Local Search (Lourenço et al., 2003), Busca Tabu (Glover, 1986; Hansen, 1986), *Simulated Annealing* (Kirkpatrick et al., 1983), *Greedy Randomized Adaptive Search Procedures* (Feo e Resende, 1995), entre outras.

Já nas meta-heurísticas baseadas em busca populacional, é mantido um conjunto de boas soluções correntes (também chamado população), e estas são combinadas de maneira a tentar produzir soluções ainda melhores. Os algoritmos do tipo evolucionário pertencem a esta categoria, alguns deles são: Algoritmos Genéticos (Goldberg, 1989), Colônia de Formigas (Dorigo et al., 1996), *Differential Evolution* (Storn e Price, 1997), entre outros.

As meta-heurísticas citadas anteriormente são para a resolução de problemas mono-objetivo. Também existem diversas meta-heurísticas na literatura para tratar problemas multiobjetivo, como: MOEA/D (Zhang e Li, 2007), NSGA-II (Deb et al., 2002), NSGA-III (Deb e Jain, 2014) e SPEA2 (Zitzler et al., 2002).

Nas próximas subseções é feita uma breve revisão de algumas meta-heurísticas referenciadas neste trabalho.

3.3.1 Variable Neighborhood Descent

A meta-heurística *Variable Neighborhood Descent* (VND) (Hansen et al., 2008) se baseia em três princípios básicos, de acordo com os autores de Hansen et al. (2008): *i*) um ótimo local com relação a uma estrutura de vizinhança não é necessariamente um ótimo local relativo a outra estrutura de vizinhança; *ii*) um ótimo global é um ótimo local com relação a todas as estruturas de vizinhanças; *iii*) para muitos problemas, ótimos locais com relação a uma ou mais estruturas de vizinhanças são relativamente próximos.

No VND são realizadas trocas sistemáticas das estruturas de vizinhanças de maneira a produzir um refinamento das soluções. No momento em que uma solução melhor que a corrente é encontrada, ela passa a ser a nova solução corrente, retornando-se à primeira estrutura de vizinhança. O VND é finalizado quando não há melhora na solução corrente em nenhuma das vizinhanças exploradas. O pseudocódigo do VND para um problema de minimização é dado pelo Algoritmo 3.3. Para um problema de maximização a modificação é trivial, troca-se a condição $f(s') < f(s)$ por $f(s') > f(s)$.

O VND é comumente utilizado como módulo de busca local de outras meta-heurísticas, como o *Iterated Local Search* e o GRASP.

Algoritmo 3.3: *Variable Neighborhood Descent*

Entrada: Função de avaliação $f(.)$, conjunto de vizinhanças $V(.)$, total de vizinhanças r, solução s
Saída: Solução s refinada

```
1  k ← 1 ;                                  /* Estrutura de vizinhança corrente */
2  enquanto (k ≤ r) faça
3  |  Encontre o melhor vizinho s' ∈ V^(k)(s);
4  |  se (f(s') < f(s)) então
5  |  |  s ← s';
6  |  |  k ← 1
7  |  fim
8  |  senão
9  |  |  k ← k + 1
10 |  fim
11 fim
12 Retorne s;
```

Uma versão do VND que evita a determinação da ordem das vizinhanças é proposta em Souza et al. (2010), sendo denominada *Random Variable Neighborhood Descent* (RVND). No RVND a cada chamada da meta-heurística, a ordem de execução das buscas locais é embaralhada, uma vez que segundo os autores a determinação da melhor ordem pode depender não somente do problema tratado, mas também da instância.

3.3.2 Iterated Local Search

A ideia da meta-heurística *Iterated Local Search* – ILS (Lourenço et al., 2003) é gerar novas soluções de partida por meio de perturbações na solução ótima local. Perturbação pode ser definida como alterações nos valores dos componentes de uma dada solução, por exemplo, dada uma solução representada por um conjunto de bits, uma perturbação pode ser a troca no valor de cinco bits selecionados aleatoriamente.

O pseudocódigo do ILS é apresentado pelo Algoritmo 3.4. Inicialmente é gerada uma solução, que pode ser obtida por uma heurística construtiva; posteriormente, é realizado um refinamento da solução inicial.

A seguir, entra-se no laço de repetição do algoritmo até que um critério de parada seja satisfeito. Dois exemplos de critérios de parada são o número máximo de iterações e o tempo máximo de execução. A cada iteração a solução corrente é perturbada, e depois é refinada por uma busca local. Após o refinamento, a solução corrente passa

por um critério de aceitação. Este critério define se esta nova solução será aceita. Em caso afirmativo, passa a ser a nova solução corrente, em caso negativo, é descartada. O histórico é responsável por definir o grau de perturbação aplicado, e também o critério de aceitação para avaliar se a solução gerada será aceita. A melhor solução encontrada é retornada ao final do laço.

Algoritmo 3.4: *Iterated Local Search*

Entrada: função de avaliação g
Saída: Solução s refinada
1 $s_0 \leftarrow$ *GeraSolucaoInicial*();
2 $s \leftarrow$ *BuscaLocal*(s_0);
3 **enquanto** *(critério de parada não satisfeito)* **faça**
4 $\quad s' \leftarrow$ *Perturbacao*(s, *histórico*);
5 $\quad s'' \leftarrow$ *BuscaLocal*(s', g);
6 $\quad s \leftarrow$ *CriterioAceitacao*(s, s'', *histórico*, g);
7 **fim**
8 **Retorne** s;

Nos trabalhos Cota et al. (2014a,b); Haddad et al. (2014, 2015) são propostos algoritmos eficazes para resolver o $R_M|S_{ijk}|C_{\max}$, baseados na meta-heurística ILS. Nesses algoritmos é usado o RVND para realizar as buscas locais.

3.3.3 Adaptive Large Neighborhood Search

Esta meta-heurística foi proposta por Shaw em 1998 (Shaw, 1998). Inicialmente tratava-se de uma versão não adaptativa, e por isso, foi chamada *Large Neighborhood Search* (LNS). A ideia desta meta-heurística é melhorar gradualmente uma solução inicial ao combinar um operador de destruição e um operador de reparação (ou uma heurística de remoção e uma de inserção). O operador de destruição remove alguns elementos da solução e o operador de reparação insere esses elementos de novas maneiras na solução. Em outras palavras, o operador de destruição e reparação desempenham o papel de movimento em uma grande vizinhança, dando nome à meta-heurística. Essas operadores podem ser implementadas de diferentes maneiras, combinando características aleatórias e gulosas que afetam o desempenho da meta-heurística.

Mais tarde, em Ropke e Pisinger (2006) foi proposta uma versão adaptativa do LNS, o chamado *Adaptive Large Neighborhood Search* (ALNS). Nesta versão, ao invés de usar apenas um operador de destruição e um de reparação, são usados vários operadores

de destruição e de reparação. Os operadores que tiveram um bom desempenho em iterações anteriores, recebem maior chance (ou peso) de serem escolhidos que os que apresentaram um desempenho fraco. Assim, cada operador recebe uma probabilidade de seleção baseado no seu peso. O pseudocódigo do ALNS é apresentado pelo Algoritmo 3.5.

Algoritmo 3.5: *Adaptive Large Neighborhood Search*

Entrada: Função de avaliação g
Saída: Solução s^*

```
1  s ← GeraSolucaoInicial();
2  s* ← s;
3  enquanto (critério de parada não satisfeito) faça
4  |  Escolha um operador de remoção e inserção, O⁻ e O⁺, usando método roleta;
5  |  Gere uma nova solução s' a partir da aplicação de O⁻e O⁺ em s;
6  |  se (g(s') < g(s)) então
7  |  |  s ← s';
8  |  fim
9  |  senão se (g(s') > g(s) ∧ s' é aceita) então
10 |  |  s ← s';
11 |  fim
12 |  se (g(s') < g(s*)) então
13 |  |  s* ← s';
14 |  fim
15 |  Atualize os pesos w_{O⁻} e w_{O⁺} dos operadores O⁻e O⁺;
16 |  Atualize as probabilidades de seleção de todos os operadores;
17 fim
18 Retorne s*;
```

A meta-heurística ALNS funciona como segue. Ela recebe como parâmetro uma função de avaliação $g(.)$. Inicialmente é gerada uma solução inicial s e a melhor solução é atualizada (s^*). A cada iteração do algoritmo é escolhido um operador de remoção (O^-) e um operador de inserção (O^+) usando um método do tipo roleta. O método roleta utiliza as probabilidades de escolha dos operadores. Por exemplo, para *nop* operadores com pesos w_b, $b \in \{1, 2, ..., nop\}$, seleciona-se o operador O' com a probabilidade $\frac{w_{O'}}{\sum_{b=1}^{nop} w_b}$. Após isso, é gerada uma nova solução s' a partir da aplicação dos operadores selecionados (O^- e O^+). No contexto dos problemas de sequenciamento de máquinas, os operadores de remoção e inserção podem ser heurísticas para remover tarefas e heurísticas para inserir tarefas, respectivamente. A ideia deste mecanismo do ALNS é explorar o espaço de soluções utilizando esses operadores de remoção e inserção. Se a solução corrente s' é melhor que a solução atual s, então a solução atual é atualizada. Se a solução corrente s' é

pior que a solução atual s, ela pode ser aceita segundo um critério de aceitação. O critério geralmente utilizado é o de temperatura da meta-heurística *Simulated Annealing*. Se a solução corrente s' é melhor que a melhor solução s^*, então a melhor solução é atualizada. Ao final de cada iteração, os pesos e as probabilidades de seleção dos operadores são atualizados. Ao final do algoritmo, a melhor solução s^* é retornada.

3.3.4 MOEA/D

O algoritmo multiobjetivo MOEA/D foi proposto em Zhang e Li (2007) e sua principal característica é a setorização da busca na aproximação da fronteira Pareto. No MOEA/D um problema multiobjetivo é dividido em S subproblemas mono-objetivo usando métodos de decomposição e vetores de peso distribuídos uniformemente. A exploração dos subproblemas é feita por meio de operadores de reprodução. O pseudocódigo da versão padrão do MOEA/D é apresentada no Algoritmo 3.6.

Algoritmo 3.6: MOEA/D

entrada: Total de subproblemas S, total de vizinhos T e total de gerações G
saída : População Pop

```
1  w ← métodoScheffé(S);
2  B ← geraVizinhos(T, w, S);
3  para i = 1 até S faça
4  |  Pop_i ← procedimentoConstrutivo ();
5  fim
6  z* ← ∅;
7  z* ← atualizaZ(Pop, z*);
8  enquanto G ≥ 0 faça
9  |  para i = 1 até S faça
10 |  |  Seleciona aleatoriamente dois vizinhos l e k de B_i;
11 |  |  sol_c ← filho gerado da aplicação de operadores genéticos aos pais Pop_l e
   |  |  Pop_k;
12 |  |  atualizaZ(sol_c, z*);
13 |  |  para cada vizinho j ∈ B_i faça
14 |  |  |  se g^te(sol_c|w_j, z*) ≤ g^te(Pop_j|w_j, z*) então
15 |  |  |  |  Pop_j ← sol_c;
16 |  |  |  fim
17 |  |  fim
18 |  fim
19 fim
20 Retorne Pop
```

Inicialmente, são construídos vetores de peso w para todos os subproblemas usando o método proposto por Scheffé (Scheffé, 1958). Este método define a estrutura $\{r, w_{max}\}$-*simplex*, que foi usada em experimentos com misturas de um grupo de componentes. No contexto do MOEA/D, r é o número de objetivos e w_{max} é o total de vetores de peso (tem o mesmo valor de S). Neste método, são gerados $\binom{r+w_{max}-1}{w_{max}}$ pontos no espaço r-dimensional, com $w_{max}+1$ pontos igualmente espaçados e satisfazendo a condição: $||\mathbf{w}||_1 = w_1 + w_2 + ... + w_r = 1$.

No conjunto B_i são armazenados os T vizinhos mais próximos para cada subproblema $i \in S$. Um método é usado para gerar os indivíduos da população (Pop), sendo gerado um indivíduo para cada subproblema. Um vetor z^* mantém a melhor solução encontrada para cada objetivo do problema. Após estes passos, entra-se em um laço de repetição. A cada iteração são realizadas as seguintes etapas: 1) Seleção aleatória de dois vizinhos l e k do subproblema i; 2) Geração de um filho sol_c da aplicação de operadores genéticos nos pais Pop_l e Pop_k; 3) Atualização do vetor z^*; 4) Para cada vizinho j de B_i, é avaliado se a solução sol_c é melhor que a solução Pop_j deste vizinho. Em caso afirmativo, Pop_j recebe sol_c. O processo de avaliação é feito com uma função de decomposição. Os passos são repetidos até que se encerre o número de gerações. Ao final, é retornada a população Pop.

Na versão padrão do MOEA/D é utilizada a função de decomposição *Tchebycheff Approach* (TCH). Segundo Trivedi et al. (2017), nesta função de decomposição o i-ésimo subproblema pode ser definido da seguinte forma:

$$\text{minimize } g^{te}(x|w_i, z^*) = \max_{1 \leq r \leq R} \{w_i^r | f_r(x) - z_r^*|\} \tag{3.10}$$

na qual $z^* = (z_1^*, ..., z_R^*)^T$ é o ponto de referência ideal com:

$$z_r^* \leq \min\{f_r(x)|x \in \Omega\} \ \textit{para } r = 1, 2, ..., R. \tag{3.11}$$

3.4 Método Húngaro

O método Húngaro foi proposto por Kuhn (1955, 2010). Este método é baseado no teorema de König e Egerváry (Jungnickel, 2008). O método Húngaro é capaz de resolver de maneira ótima o problema de emparelhamento ponderado para um grafo bipartido completo, presente na teoria de grafos (Papadimitriou e Steiglitz, 1982). Este problema é equivalente ao problema de atribuição linear.

Segundo Papadimitriou e Steiglitz (1982), a solução de problemas de atribuição linear por meio de força bruta tem uma complexidade de tempo fatorial, por outro lado, o método Húngaro pode resolver estes problemas em tempo polinomial. Para um problema de emparelhamento ponderado para um grafo bipartido com $2n$ nós, o método Húngaro necessita de $\mathcal{O}(n^3)$ operações para resolvê-lo.

Um problema de atribuição linear pode ser descrito como a seguir. Suponha um conjunto M de máquinas e um conjunto N de tarefas, ambos de mesmo tamanho. Uma matriz quadrada $W = w_{n_i,m_j}$ é dada, e representa os custos de atribuição de cada tarefa $n_j \in N$ em cada máquina $m_i \in M$. Um problema de atribuição é dado pela escolha de uma única máquina para cada tarefa.

O método Húngaro consiste em cinco etapas (Goldberger e Tassa, 2008; Lehmann, 1984), dadas a seguir:

1. Subtraia a menor entrada de cada linha de todas as entradas da mesma linha. Dessa forma, cada linha terá pelo menos uma entrada zero e todas as outras entradas são não negativas.

2. Subtraia a menor entrada de cada coluna de todas as entradas da mesma coluna. Dessa forma, cada coluna terá pelo menos uma entrada zero e todas as outras entradas são não negativas.

3. Faça um traço ao longo de linhas e colunas de tal modo que todas as entradas zero da matriz-custo fiquem riscadas. Para isso, use o número mínimo de traços.

4. Teste de otimalidade:

   (a) Seja n número total de linhas ou colunas da matriz de custos. Se o número mínimo de traços necessários para cobrir os zeros é n, então é possível uma alocação ótima de zeros e o procedimento pode ser interrompido.

(b) Se o número mínimo de traços necessários para cobrir os zeros não é menor que n, então uma alocação ótima de zeros não é possível Neste caso, vá para o passo 5.

5. Determine a menor entrada não marcada. Subtraia este valor de todas as entradas não marcadas e adicione-o a todas as entradas com traço (De maneira similar, a operação de adição pode ser aplicada apenas onde as linhas se cruzam (Lehmann, 1984)). Retorne ao passo 3.

A seguir é dado um exemplo para facilitar o entendimento do método. Suponha um problema de atribuição com um conjunto de 5 tarefas (N) e um conjunto de 5 máquinas (M). A matriz W representa o custo das alocações de cada tarefa ($n_j \in N$) em cada máquina ($m_i \in M$), esta matriz é apresentada na Figura 3.1.

$$\mathbf{W} = \begin{array}{c} (n_j, m_i) \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \begin{pmatrix} 20 & 21 & 21 & 15 & \mathbf{10} \\ 18 & 17 & \mathbf{10} & 16 & 18 \\ 19 & 22 & \mathbf{16} & 19 & 20 \\ 12 & 14 & 21 & \mathbf{11} & 17 \\ 24 & 19 & 20 & \mathbf{12} & 23 \end{pmatrix} \end{array}$$

Figura 3.1: Custo de alocação de cada tarefa em cada máquina.

Em D é apresentada uma atribuição aleatória para o problema, que é definida por $D = \{(n_1, m_2), (n_2, m_5), (n_3, m_3), (n_4, m_1), (n_5, m_4)\}$. O custo de D é dado por $W(D) = 21 + 18 + 16 + 12 + 12 = 79$.

Para obter a atribuição ótima, a primeira etapa do método Húngaro é definir a menor entrada de cada linha (S_{linha}). Estes valores estão destacados em negrito na matriz W (Figura 3.1) e são mostrados na Figura 3.2.

Para cada linha, esses valores são subtraídos na mesma linha (etapa 1). O resultado desta operação encontra-se na Figura 3.3.

O próximo passo consiste em subtrair a menor entrada de cada coluna da mesma coluna (S_{coluna}). Esses valores são apresentados na Figura 3.4.

$$S_{linha} = \begin{pmatrix} 10 \\ 10 \\ 16 \\ 11 \\ 12 \end{pmatrix}$$

Figura 3.2: A menor entrada de cada linha.

$$\mathbf{W} = \begin{array}{c} (n_j, m_i) \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \begin{pmatrix} 10 & 11 & 11 & 5 & 0 \\ 8 & 7 & 0 & 6 & 8 \\ 3 & 6 & 0 & 3 & 4 \\ 1 & 3 & 10 & 0 & 6 \\ 12 & 7 & 8 & 0 & 11 \end{pmatrix} \end{array}$$

Figura 3.3: Resultado após a etapa 1.

$$S_{coluna} = \begin{pmatrix} 1 & 3 & 0 & 0 & 0 \end{pmatrix}$$

Figura 3.4: A menor entrada de cada coluna.

Para cada coluna, esses valores são subtraídos na mesma coluna (etapa 2). O resultado dessa operação encontra-se na Figura 3.5.

Depois de subtrair as menores entradas das linhas e colunas, um número mínimo de traços é desenhado para cobrir todos os zeros (etapa 3). Esses traços são apresentados na Figura 3.6.

A próxima etapa é verificar se o resultado é ótimo. Como o número de traços é inferior a 5 (ordem da matriz), o resultado é inviável. Isso significa que a etapa 4(a) não é válida neste momento e a etapa 4(b) é aplicada.

Para aplicar a etapa 5, a menor entrada não marcada na Figura 3.6 é selecionada, que é o valor 2. Este valor é subtraído de todas as entradas não marcadas (Figura 3.7a)

$$\mathbf{W} = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \begin{pmatrix} 9 & 8 & 11 & 5 & 0 \\ 7 & 4 & 0 & 6 & 8 \\ 2 & 3 & 0 & 3 & 4 \\ 0 & 0 & 10 & 0 & 6 \\ 11 & 4 & 8 & 0 & 11 \end{pmatrix} \end{array}$$

Figura 3.5: Resultado após a etapa 2.

$$\mathbf{W} = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \begin{pmatrix} 9 & 8 & 11 & 5 & 0 \\ 7 & 4 & 0 & 6 & 8 \\ 2 & 3 & 0 & 3 & 4 \\ \cancel{0} & \cancel{0} & \cancel{10} & \cancel{0} & \cancel{6} \\ 11 & 4 & 8 & 0 & 11 \end{pmatrix} \end{array}$$

Figura 3.6: Resultado após a etapa 3.

e adicionado a todas as entradas onde as linhas se cruzam (Figura 3.7b). Depois disso, a etapa 3 é executada novamente.

$$\mathbf{W} = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \begin{pmatrix} \mathbf{7} & \mathbf{6} & 11 & 5 & 0 \\ \mathbf{5} & \mathbf{2} & 0 & 6 & 8 \\ \mathbf{0} & \mathbf{1} & 0 & 3 & 4 \\ \cancel{0} & \cancel{0} & \cancel{10} & \cancel{0} & \cancel{6} \\ \mathbf{9} & \mathbf{2} & 8 & 0 & 11 \end{pmatrix} \end{array}$$

(a) Resultado após subtrair o valor 2 de todas as entradas não marcadas.

$$\mathbf{W} = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \begin{pmatrix} 7 & 6 & 11 & 5 & 0 \\ 5 & 2 & 0 & 6 & 8 \\ 0 & 1 & 0 & 3 & 4 \\ \cancel{0} & \cancel{0} & \mathbf{12} & \mathbf{2} & \mathbf{8} \\ 9 & 2 & 8 & 0 & 11 \end{pmatrix} \end{array}$$

(b) Resultado após adicionar o valor 2 à todas entradas onde as linhas se cruzam

Figura 3.7: Resultado após a etapa 5.

Os traços são desenhados novamente (etapa 3) e o teste de otimização é aplicado. O

resultado é apresentado na Figura 3.8. Como o número de traços é igual à ordem da matriz, o resultado ótimo foi encontrado e o processo é concluído.

$$\mathbf{W} = \begin{array}{c} (n_j, m_i) \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \left(\begin{array}{ccccc} 7 & 6 & 11 & 5 & 0 \\ 5 & 2 & 0 & 6 & 8 \\ 0 & 1 & 0 & 3 & 4 \\ 0 & 0 & 12 & 2 & 8 \\ 9 & 2 & 8 & 0 & 11 \end{array} \right) \end{array}$$

Figura 3.8: Resultado após redesenhar os traços.

Depois que o resultado ótimo é encontrado, as atribuições são definidas. Conforme apresentado na Figura 3.9, os valores zero indicam possíveis atribuições. Esses valores foram retirados do resultado final na Figura 3.8.

$$\mathbf{W} = \begin{array}{c} (n_j, m_i) \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \left(\begin{array}{ccccc} & & & & \textcircled{0} \\ & & \textcircled{0} & & \\ \textcircled{0} & & 0 & & \\ 0 & \textcircled{0} & & & \\ & & & \textcircled{0} & \end{array} \right) \end{array}$$

Figura 3.9: Possíveis atribuições.

Para definir a atribuição, as linhas com exatamente uma entrada zero são selecionadas primeiro. Então, n_1 é atribuído a m_5, n_2 é atribuído a m_3 e n_5 é atribuído a m_4. As opções para atribuir n_3 são m_1 e m_3. No entanto, como n_2 já foi atribuído a $m3$, a única opção disponível para n_3 é m_1. Do mesmo modo, a única opção disponível para n_4 é m_2. Então, a atribuição final é dada por $D = \{(n_1, m_5), (n_2, m_3), (n_3, m_1), (n_4, m_2), (n_5, m_4)\}$ e o custo é dado por $W(D) = 10 + 10 + 19 + 14 + 12 = \mathbf{65}$.

Em geral, o método Húngaro é aplicado para resolver problemas de minimização. No entanto, o método pode ser usado para resolver problemas de maximização. Para isso, o problema de maximização é transformado em um de minimização, subtraindo-se o valor

mais alto na matriz de custos ($w_{\max}$) de todos os valores de entrada. Essa conversão é feita substituindo cada w_{n_i,m_j} por $w_{\max} - w_{n_i,m_j}$. Depois disso, todas as etapas são realizadas como um problema de minimização. No final, o custo final é calculado com os valores da matriz de custo original (antes da transformação).

3.5 Conclusão

Neste capítulo é apresentada a definição de otimização mono-objetivo e otimização multiobjetivo. Posteriormente, são apresentados diversos tipos de técnicas para a resolução de problemas de sequenciamento em máquinas. Inicialmente, são tratadas as técnicas heurísticas que englobam as heurísticas e as meta-heurísticas. Dentre as heurísticas, são apresentadas as heurísticas construtivas e as buscas locais. Já dentre as meta-heurísticas, são apresentadas as referenciadas ao longo deste trabalho, a saber: *Variable Neighborhood Search*, *Iterated Local Search*, *Adaptive Large Neighborhooh Search* e MOEA/D. Ao final, é apresentado o método Húngaro, que é capaz de resolver problemas de atribuição linear de maneira ótima em tempo polinomial.

Capítulo 4

ALNS com aprendizado em autômatos para o $R_M|S_{ijk}|C_{max}$

4.1 Introdução

Parte do conteúdo deste capítulo foi recentemente publicado em Cota et al. (2017a,b), com a participação do presente autor. Neste capítulo é tratada a resolução de uma versão clássica do UPMSP-ST com o objetivo de minimizar o *makespan*, este problema pode ser definido pela notação $R_M|S_{ijk}|C_{max}$ (como descrito na Seção 2.2). A resolução deste problema de maneira eficiente é desafiadora dada a sua complexidade e ampla aplicabilidade prática. Na literatura são encontrados muitos trabalhos abordando sua resolução, como foi descrito na seção de revisão bibliográfica.

A proposta deste capítulo é desenvolver um algoritmo adaptativo de propósito geral que resolva o $R_M|S_{ijk}|C_{max}$ de maneira eficiente. Na literatura existem dois grandes conjuntos de instâncias para o problema, que possuem tamanho e características bem distintas. Devido a esta particularidade, grande parte dos trabalhos da literatura utilizam apenas um destes conjuntos de instâncias. A ideia é que o algoritmo proposto neste capítulo consiga resolver de maneira eficaz estes dois conjuntos de instâncias. O algoritmo escolhido para a implementação é a meta-heurística ALNS (descrita na Seção 3.3.3).

Desde a sua proposta, o algoritmo ALNS tem sido aplicado com sucesso a vários problemas, no entanto, com apenas alguns estudos na área de sequenciamento de máquinas. Uma aplicação recente do ALNS a um problema de sequenciamento de máquinas é tra-

tado em Beezão et al. (2017). Neste trabalho é abordado um problema de sequenciamento em máquinas paralelas idênticas com restrições de ferramentas, cujo objetivo é minimizar o *makespan*.

Alguns trabalhos recentes em que o ALNS é usado para resolver outros problemas de sequenciamento são descritos a seguir. Em Vadlamani e Hosseini (2014); Yu et al. (2017) os autores utilizam o método para resolver problemas no contexto da aviação. Já em Bakkehaug et al. (2016); Iris et al. (2017) são abordados problemas de alocação de navios. Um problema de sequenciamento no processamento de imagens de satélites é resolvido em Liu et al. (2017). Na área de roteamento de veículos são encontrados alguns trabalhos recentes, como: *i*) Em Dayarian et al. (2006) os autores tratam a resolução de um problema de roteamento de veículos multi-período; *ii*) Em Mattos Ribeiro e Laporte (2012) os autores resolvem um problema de roteamento de veículos acumulado e capacitado; *iii*) Em Wen et al. (2016) os autores propõem a resolução de um problema de roteamento de ônibus elétricos; *iv*) Em Grimault et al. (2017) os autores tratam um problema de roteamento de caminhões para coleta e entrega de mercadorias; *v*) Em Majidi et al. (2017) os autores abordam um problema parecido com anterior, mas em um contexto de desenvolvimento sustentável, com a preocupação de minimizar as emissões de gases e o consumo de combustível.

As implementações usuais do ALNS empregam um peso simples para cada heurística (às vezes chamada de pontuação em alguns trabalhos), que é aumentada sempre que essa heurística é bem sucedida na melhoria da solução atual. Um método do tipo roleta é usado para selecionar as heurísticas com base em seus valores de pontuação. Este mecanismo simples para adaptar as heurísticas de remoção e inserção tem a desvantagem de ser guloso, podendo levar a uma convergência prematura na seleção de probabilidade de heurísticas.

Neste trabalho, propomos um método de aprendizagem na adaptação das probabilidades de seleção de heurísticas de remoção e inserção. O ALNS proposto usa aprendizagem com autômatos para aprender as probabilidades dos métodos de inserção e remoção, ajustando-os dinamicamente ao longo do processo de otimização. Além disso, propomos um novo método de inserção inspirado no método húngaro (descrito na Seção 3.4).

A demais seções do capítulo seguem organizadas como a seguir. Na Seção 4.2 é tratado o funcionamento dos autômatos com aprendizagem. O algoritmo proposto é apresentado na Seção 4.3. Os experimentos computacionais são abordados na Seção 4.4. As conclusões deste capítulo são tratadas na Seção 4.5.

4.2 Autômatos com aprendizagem

Os autômatos com aprendizagem (ou *Learning Automata* - LA) são métodos estocásticos e consistem em definir uma melhor ação em um ambiente de um conjunto finito de ações disponíveis. Eles representam uma unidade de decisão adaptativa que melhora o seu desempenho por meio de interações repetidas em um ambiente aleatório desconhecido (Alipour, 2012b; Narendra e Thathachar, 1989, 1974).

A ação é escolhida aleatoriamente por meio de uma distribuição de probabilidade dentre um conjunto de ações. A cada iteração, a ação selecionada é usada como entrada no ambiente aleatório para aprendizagem futura.

O ambiente e as *Learning Automata* são descritos por $(\phi, \alpha, \beta, A, \pi, p)$ (Narendra e Thathachar, 2012), onde ϕ é um conjunto de estados internos; α é um conjunto de saídas ou ações dos autômatos com aprendizagem; β é um conjunto de respostas do ambiente; A é um *Learning Automaton*; $\pi : \phi \mapsto \alpha$ é uma função que mapeia o estado corrente (ϕ) para uma saída corrente (α) e p é um vetor de probabilidades que determina a probabilidade de seleção de um estado em cada estágio. Na Figura 4.1 é ilustrada a relação entre as *Learning Automata* e seu ambiente aleatório.

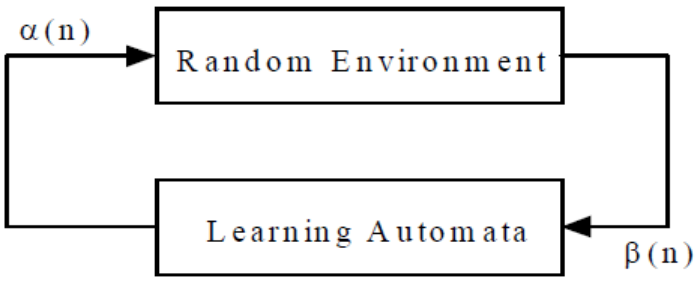


Figura 4.1: Relação entre as *Learning Automata* e seu ambiente aleatório (Alipour, 2012a).

Quando uma ação é aplicada a um ambiente, ela responde o *Learning Automaton* com um sinal de ruído. O *Learning Automaton* usa essa resposta para ajustar as probabilidades da ação interna usando o seu algoritmo de aprendizagem. Seja β uma resposta do ambiente no passo k com $\beta \in \{0, 1\}$, em que as respostas 0 e 1 significam "agradável" e "desagradável", respectivamente. Se a resposta é "agradável", a probabilidade da ação interna é atualizada com a Equação (4.1). Caso contrário, se a resposta for "desagradável", a probabilidade é atualizada com a Equação (4.2) (Vafashoar e Meybodi,

2016).

$$p_j(k+1) = \begin{cases} p_j(k) + a(1 - p_j(k)) & \text{se } i = j \\ p_j(k)(1-a) & \text{se } i \neq j \end{cases} \tag{4.1}$$

$$p_j(k+1) = \begin{cases} p_j(k)(1-b) & \text{se } i = j \\ \frac{b}{r-1} + p_j(k)(1-b) & \text{se } i \neq j \end{cases} \tag{4.2}$$

Para as Equações (4.1) e (4.2), a e b são chamados de parâmetros de *recompensa* e *penalidade*, respectivamente. O índice i indica o método da LA que foi aplicado no momento, o índice j representa todos os métodos da LA e o k representa a iteração corrente.

O número de ações que pode ser escolhida pelo autômato é definido por $r = |\alpha|$. Quando $a = b$, o algoritmo de aprendizagem é denominado recompensa-penalidade linear (L_{R-P}). Quando $a << b$, é chamado penalidade recompensa-ϵ linear ($L_{R\epsilon P}$), e quando b é igual a zero, o algoritmo de aprendizagem é chamado recompensa linear-inação (L_{R-I}) (Vafashoar e Meybodi, 2016).

Neste trabalho, os autômatos com aprendizagem são aplicados para saber quais métodos (ou heurísticas) podem melhorar a solução do problema tratado. Para ilustrar o seu funcionamento, um exemplo é dado a seguir. Considere um conjunto de 3 métodos, $\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$. Esses métodos estão disponíveis para o *Learning Automaton* e uma resposta β é obtida para cada um deles: $\beta(\alpha_1) = 1; \beta(\alpha_2) = 0; \beta(\alpha_3) = 1$. As respostas para os métodos α_1 e α_3 são do tipo "desagradável", com isso, uma penalidade b é aplicada para cada método usando a Equação (4.2). Como a resposta para o método α_2 é do tipo "agradável", uma recompensa a é aplicada usando a Equação (4.1). A próxima etapa k usará as novas probabilidades internas de cada método para selecionar a próxima ação α_i.

Autômatos com aprendizagem podem ser úteis em casos em que não há informações completas sobre o ambiente (Alipour, 2012a,b). Em nossa aplicação, não há informações sobre quais métodos funcionam melhor no início. Enquanto o processo é executado, o *Learning Automaton* atualiza a probabilidade de cada método e aplica aqueles que

melhoram a solução de forma adequada.

4.3 Algoritmo proposto

4.3.1 Representação e avaliação da solução

Uma solução é representada como um vetor de inteiros com m posições, em que m é o número total de máquinas. Cada posição do vetor é associada a uma lista de tarefas alocadas à máquina correspondente. Na Figura 4.2 é ilustrada uma possível solução para uma instância com duas máquinas e seis tarefas. Na máquina M_1 estão alocadas as tarefas 3, 5 e 1, nesta ordem. Na máquina M_2 estão alocadas as tarefas 4, 2 e 6, nesta ordem.

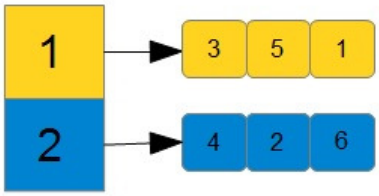


Figura 4.2: Representação de uma solução no LA-ALNS.

A avaliação de uma solução é dada pelo seu *makespan* (o tempo máximo de conclusão de todas as máquinas).

4.3.2 Algoritmo LA-ALNS

O algoritmo proposto é denominado LA-ALNS, combinando ALNS e LA. Este novo algoritmo é uma implementação da meta-heurística *Adaptive Large Neighborhood Search* (Ropke e Pisinger, 2006) que usa aprendizagem em autômatos para aprender a melhor escolha momentânea dos métodos de remoção e inserção. Existe um *Learning Automaton* para os métodos de inserção (LA_{N^+}) e outro para os métodos de remoção (LA_{N^-}). A solução inicial é gerada por uma heurística construtiva baseada na fase construtiva da meta-heurística GRASP (Feo e Resende, 1995). As buscas locais são realizadas por um RVND. O pseudocódigo do LA-ALNS é apresentado no Algoritmo 4.1.

Algoritmo 4.1: LA-ALNS

entrada: $t_{\max}, K, q, a_1, a_2, a_3, b_1$
saída : s_{melhor}

Defina LA $(\phi^-, \alpha^-, \beta, A^-, \pi^-, p^-)$;
Defina LA $(\phi^+, \alpha^+, \beta, A^+, \pi^+, p^+)$;
$s \leftarrow$ `procedimentoConstrutivo` ();
$s_{melhor} \leftarrow s$;
$T \leftarrow (0,1 \times f(s))/\ln 2$;
enquanto *tempoAtual* $\leq t_{\max}$ **faça**
    Selecione α_i^- usando o método da roleta;
    Selecione α_j^+ usando o método da roleta;
    Remova q tarefas de s usando α_i^-;
    Insira as tarefas removidas de s usando α_j^+;
    $s' \leftarrow$ `RVND(`s`)`;
    se $f(s') < f(s_{melhor})$ **então**
        $s_{melhor} \leftarrow s'$; $s \leftarrow s'$;
        $a \leftarrow a_1$; $b \leftarrow 0$;
        Atualize p^- e p^+ usando as Equações (4.1)-(4.2);
    fim
    senão se $f(s') < f(s)$ **então**
        $s \leftarrow s'$;
        $a \leftarrow a_2$; $b \leftarrow 0$;
        Atualize p^- e p^+ usando as Equações (4.1)-(4.2);
    fim
    senão se `aleatório` $< e^{-\frac{(f(s')-f(s))}{T}}$ **então**
        $s \leftarrow s'$;
        $a \leftarrow a_3$; $b \leftarrow 0$;
        Atualize p^- e p^+ usando as Equações (4.1)-(4.2);
    fim
    senão
        $a \leftarrow 0$; $b \leftarrow b_1$;
        Atualize p^- e p^+ usando as Equações (4.1)-(4.2);
    fim
    se $k \mod K = 0$ **então**
        `atualizaLA`(A^-, A^+, p^-, p^+);
        $k \leftarrow 0$;
    fim
    $k++$;
    $T \leftarrow 0,99 \times T$;
fim
retorne s_{melhor}

Os parâmetros de entrada do Algoritmo 4.1 são: $t_{\max}$; K; q; a_1; a_2; a_3; e b_1. O parâmetro $t_{\max}$ é o tempo limite para execução. A cada K iterações as probabilidades dos métodos de remoção e inserção são atualizadas. Este parâmetro foi definido pela Equação $6 \times \max(|\alpha^-|, |\alpha^+|)$, obtida previamente por testes empíricos. Nestes testes foram utilizadas instâncias de diferentes tamanhos com objetivo de identificar o número suficiente de iterações para o aprendizado das LAs. O parâmetro q define o total de tarefas que são removidas, e depois inseridas na solução corrente a cada iteração do Algoritmo. Este parâmetro foi definido como 5% do número total de tarefas da instância. O parâmetro q também foi obtido previamente por testes empíricos. Nestes testes foram utilizadas 400 instâncias e variações no parâmetro q de 4% à 30%. O a_1, a_2 e a_3 são parâmetros de recompensa. Eles são usados para atualizar as probabilidades quando a solução corrente é aceita. O parâmetro b_1 é uma penalidade e é usado quando a solução corrente não é aceita. A temperatura inicial é calculada de modo que uma solução corrente tenha 50% de chance de aceitação se esta for 10% pior do que a solução inicial.

Os passos do algoritmo a cada iteração são:

1. Um método do tipo roleta é usado para selecionar um método de remoção $\alpha_i^- \in \alpha^-$ e um método de inserção $\alpha_j^+ \in \alpha^+$. O método roleta usa as probabilidades dos métodos de remoção e inserção;

2. Após a seleção dos métodos de remoção e inserção, q tarefas são removidas da solução corrente (s') usando o método α_i^- e então inseridas usando o método α_j^+. A aplicação de métodos de remoção e inserção funciona como uma perturbação tendenciosa aplicada à solução corrente;

3. O método de buscas locais $RVND$ é aplicado à solução corrente s';

4. Após isso, as probabilidade p^- e p^+ são atualizadas. Se a solução s' é aceita, a atualização usa a Eq. (4.1), caso contrário, se a solução s' não é aceita, a atualização usa a Eq. (4.2);

5. A aprendizagem ocorre em todas as iterações, mas apenas após K iterações os valores de probabilidade são atualizadas dentro de cada LA. Esta atualização é feita periodicamente para dar tempo às LAs para experimentarem algumas ações no ambiente e somente depois atualizar as probabilidades de seleção;

6. Estes passos são repetidos até o fim do tempo de execução ($t_{\max}$), quando é retornada a melhor solução encontrada.

Um solução pode ser aceita em três diferentes situações e para cada situação um parâmetro de recompensa (a_1, a_2 e a_3) é aplicado para atualizar as *Learning Automata*. As três situações são: *i*) a_1: Se a solução encontrada é melhor que a melhor solução; *ii*) a_2: Se a solução encontrada é melhor que a solução corrente; *iii*) a_3: Se a solução encontrada é pior que a solução corrente, mas é aceita de acordo com o critério de temperatura. Caso contrário, se a solução encontrada não é aceita, o parâmetro de penalidade (b) é usado para atualizar as LA. As discussões realizadas em Vafashoar e Meybodi (2016) foram usadas para definir estes parâmetros. Os valores usados foram: *i*) $a_1 = 0,2$; *ii*) $a_2 = 0,1$; *iii*) $a_3 = 0,05$; e *iv*) $b_1 = 0,02$.

4.3.3 Heurística construtiva

A heurística construtiva usada para gerar a solução inicial é inspirada na fase construtiva da meta-heurística GRASP.

Na heurística construtiva um laço de repetições é executado até que um tempo limite seja atingido. O tempo limite usado é 1% de $t_{\max}$ (tempo limite para execução do LA-ALNS). A cada iteração uma nova solução é gerada e a melhor solução é armazenada. Ao final do processo, a melhor solução encontrada é retornada. Para a geração de uma nova solução a cada iteração, é usada uma heurística parcialmente gulosa, a qual é baseada na regra *Adaptive Shortest Processing Time* (Baker, 1974). Os passos desta heurística são:

1. Considere que o conjunto $N = \{1, ..., n\}$ contém todas as tarefas e o conjunto $M = \{1, ..., m\}$ contém todas as máquinas;

2. Todas as tarefas são inseridas na lista de candidatos LC;

3. Todos os pares (i, j) de máquina $i \in M$ e tarefa $j \in LC$ são ordenadas pela função de avaliação g. Esta função calcula o custo de inserção da tarefa j no final da máquina i, considerando os tempos de processamento e preparação;

4. Entre os 20% melhores pares, um par (i, j) é selecionado aleatoriamente. A tarefa j é inserida no final da máquina i e a tarefa j é removida de LC;

5. O processo é repetido até que todas as tarefas de LC sejam alocadas.

4.3.4 Heurísticas de remoção e inserção

Nesta subseção são apresentadas as seis heurísticas de remoção e as seis heurísticas de inserção usadas no LA-ALNS. Primeiramente, são apresentadas as heurísticas de remoção, e após isso, as heurísticas de inserção. Antes de executar cada heurística de inserção, as q tarefas removidas são embaralhadas.

4.3.4.1 Remoção aleatória

Este método remove um total de q tarefas da solução corrente aleatoriamente.

4.3.4.2 Remoção maior custo gulosa

Este método classifica as tarefas usando a soma do tempo de processamento e do tempo de preparação considerando a posição alocada na solução corrente. Estes valores são armazenados no conjunto Y. As q tarefas mais custosas de Y são removidas.

4.3.4.3 Remoção maior custo semi-gulosa

Este método é uma versão semi-gulosa do método anterior e usa o mesmo conjunto Y. Neste método q tarefas são removidas aleatoriamente de um subconjunto de Y que contém 20% das tarefas de maior custo.

4.3.4.4 Remoção máquina aleatória

Este método seleciona aleatoriamente uma máquina e remove as primeiras q tarefas da máquina. Se o tamanho desta máquina é menor que q, outra máquina é selecionada aleatoriamente até que q tarefas sejam removidas.

4.3.4.5 Remoção máquina maior custo

Este método seleciona a máquina com maior custo e remove as primeiras q tarefas da máquina. Se o tamanho desta máquina é menor que q, outra máquina com maior custo é selecionada até que q tarefas sejam removidas.

4.3.4.6 Remoção Shaw

A ideia deste método é remover tarefas que são consideradas similares e cuja reinserção pode gerar melhores soluções. Este método foi proposto inicialmente por Shaw (1998). O critério de similaridade escolhido nessa implementação para cada tarefa é a soma dos tempos de processamento e preparação considerando a posição alocada na solução corrente. Inicialmente, uma tarefa j' é selecionada aleatoriamente. Então, é calculada a relação $R(j', j)$ entre a tarefa j' e todas as outras tarefas j alocadas a todas as máquinas. Ao final, as q tarefas mais similares a tarefa j' são removidas.

4.3.4.7 Inserção gulosa

Este método realiza a inserção gulosa de q tarefas na solução corrente. Para cada tarefa $j \in q$ é procurada a melhor posição entre todas as máquinas. A melhor posição é aquela que proporciona o menor custo para a máquina. Então, a tarefa é alocada na melhor posição.

4.3.4.8 Inserção semi-gulosa

Este método é uma versão semi-gulosa do método anterior. Para cada tarefa $j \in q$ é escolhida uma posição aleatória entre as 20% melhores posições. Após isso, a tarefa j é alocada nesta posição.

4.3.4.9 Inserção Lambda

Este método divide as q tarefas em dois subconjuntos com o mesmo tamanho, subconjuntos q_1 e q_2. As tarefas de q_1 são inseridas em posições aleatórias em todas as máquinas. As tarefas de q_2 são inseridas usando o método inserção gulosa.

4.3.4.10 Inserção ILS

Este método é inspirado no método de perturbação do algoritmo AIRP (Cota et al., 2014a). Para cada tarefa $j \in q$ é selecionada uma máquina aleatória i, e a tarefa j é inserida na melhor posição de i. A melhor posição é aquela que fornece o menor custo para a máquina.

4.3.4.11 Inserção arrependimento

Este método insere primeiro as tarefas com maior custo de arrependimento. O custo de arrependimento é dado pela diferença entre o custo da melhor posição e o custo da segunda melhor posição considerando todas as máquinas. A cada passo, a tarefa com maior custo de arrependimento é inserida na melhor posição.

4.3.4.12 Inserção Húngaro

Este método usa o método Húngaro (Kuhn, 1955) para inserir as q tarefas removidas. Com descrito na Seção 3.4, o método Húngaro é capaz de resolver problemas de atribuição linear otimamente em tempo polinomial. A ideia deste método de inserção é usar o método Húngaro para resolver subproblemas do problema principal de maneira ótima.

Os passos deste método de inserção são:

- Considere m o número de máquinas da solução corrente. As m primeiras tarefas de q são removidas;
- Gere uma matriz quadrada em que o índice das colunas representa as tarefas removidas de q, as linhas representam as máquinas e os dados contém o custo da melhor alocação de cada tarefa em cada máquina. Este custo é dado pela melhor posição de alocação de cada tarefa removida em cada máquina;
- Use o método Húngaro para definir a alocação ótima desta matriz;
- Insira as tarefas nas máquinas seguindo a alocação ótima;
- Esse processo é repetido até que o número de tarefas restantes de q seja menor que m.

Ao final, as tarefas restantes são inseridas usando o método de inserção gulosa.

Para ilustrar melhor o funcionamento do método inserção Húngaro, a seguir, é dado um exemplo de uma iteração do método. Considere uma instância com 3 máquinas e que as 3 primeiras tarefas de q são 5, 2 e 4. A Figura 4.3 mostra a matriz quadrada gerada para ser resolvida usando o método Húngaro.

Figura 4.3: Exemplo do método de inserção Húngaro.

$$\textbf{Exemplo} = \begin{array}{c} (\text{máquina}, \text{tarefa}) \\ \text{máquina}_1 \\ \text{máquina}_2 \\ \text{máquina}_3 \end{array} \begin{array}{c} \begin{array}{ccc} \text{tarefa}_5 & \text{tarefa}_2 & \text{tarefa}_4 \end{array} \\ \begin{pmatrix} 17 & 20 & 6 \\ 7 & 22 & 13 \\ 15 & 3 & 9 \end{pmatrix} \end{array}$$

4.3.5 Procedimento de busca local

O procedimento de busca local usado é o *Random Variable Neighborhood Descent* (descrito na Subseção 3.3.1). O RVND implementado neste trabalho usa três estruturas de vizinhanças, descritas a seguir.

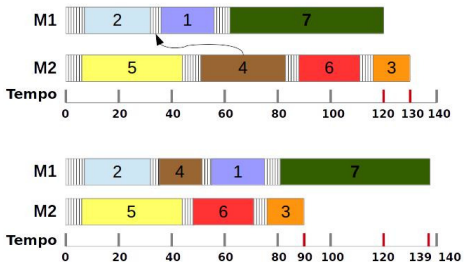
1. **Múltipla inserção** – $\boldsymbol{NS^{MI}(s)}$: esta sigla vem do termo *Neighborhood search multiple insertion* no idioma inglês. Esta estrutura de vizinhança usa o movimento de realocação de uma tarefa de uma máquina em alguma posição de outra máquina (ou da mesma máquina). Um exemplo deste movimento é apresentado na Figura 4.4a, na qual a tarefa 4 da máquina 2 é transferida para a segunda posição da máquina 1.

2. **Troca na mesma máquina** – $\boldsymbol{NS^{SSM}(s)}$: esta sigla vem do termo *Neighborhood search swap in the same machine* no idioma inglês. Esta estrutura de vizinhança usa o movimento de troca de duas tarefas de uma mesma máquina. Na Figura 4.4b é apresentado um exemplo deste movimento, na qual as tarefas 5 e 6 da máquina 2 são trocadas.

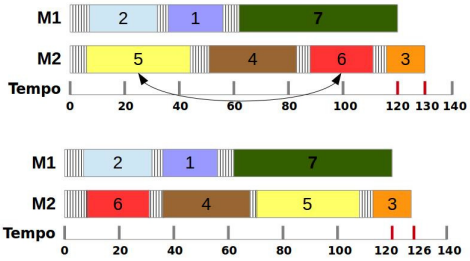
3. **Troca em máquinas diferentes** – $\boldsymbol{NS^{SDM}(s)}$: esta sigla vem do termo *Neighborhood search swap between different machines* no idioma inglês. Esta estrutura de vizinhança usa o movimento de troca de duas tarefas de máquinas diferentes. Na Figura 4.4c um exemplo deste movimento é apresentado, na qual a tarefa 7 da máquina 1 é trocada com a tarefa 5 da máquina 2.

O RVND usa três métodos de busca local selecionados aleatoriamente para explorar o espaço de busca. Cada método de busca local usa uma estrutura de vizinhança descrita anteriormente. Os métodos de busca local são dados a seguir.

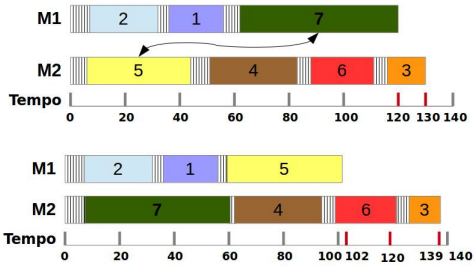
Figura 4.4: Exemplos de movimentos de vizinhanças.



(a) Movimento de múltipla inserção.



(b) Movimento de troca na mesma máquina.



(c) Movimento de troca em máquinas diferentes.

1. **Busca local FI_{MI}**: esta busca local usa a estrutura de vizinhança $NS^{MI}(s)$ e a estratégia *first improvement*. A busca local FI_{MI} foi proposta em Cota et al. (2014a), chamada FI^1_{MI} na Seção III.G. O pseudocódigo desta busca local é dado no Algoritmo 4.2;

2. **Busca local FI_{SDM}**: esta busca local usa a estrutura de vizinhança $NS^{SDM}(s)$ e a estratégia *first improvement*. A busca local FI_{SMD} foi proposta em Haddad et al. (2014), chamada *Local Search with Swaps Between Different Machines* na Seção 3.4.2. O pseudocódigo desta busca local é dado no Algoritmo 4.3;

3. **Busca Local FI_{SSM}**: esta busca local usa a estrutura de vizinhança $NS^{SSM}(s)$ e a estratégia *first improvement*. Todas as máquinas são classificadas em ordem decrescente pelo tempo conclusão. As máquinas são selecionadas a partir das máquinas de tempos de conclusão mais longo até as de tempo de conclusão mais curto. Para cada máquina, são analisadas todas as trocas entre suas tarefas. Se um vizinho $s' \in NS^{SSM}(s)$ proporciona um menor de tempo de conclusão na máquina analisada, o método de busca é encerrado e a solução s' é retornada para o RVND; caso contrário, a busca continua até que todas as trocas envolvendo

todas as máquinas sejam analisadas. O pseudocódigo desta busca local é dado no Algoritmo 4.4.

Algoritmo 4.2: Busca local FI_{MI}

```
Entrada: s, f(.)
Saída: s
Seja M o conjunto de máquinas;
Seja K_1 o conjunto M ordenado em ordem decrescente pelos tempos de conclusão;
Seja K_2 o conjunto M ordenado em ordem crescente pelos tempos de conclusão;
para cada máquina k_1 ∈ K_1 faça
    para cada tarefa j ∈ k_1 faça
        para cada máquina k_2 ∈ K_2 faça
            para cada posição i de 0 até tamanho k_2 + 1 faça
                Seja s' o resultado da realocação de j na posição de i;
                se f(s') ≤ f(s) então
                    s ← s';
                    Pare a Busca;
                fim
            fim
        fim
    fim
fim
Retorne s;
```

4.4 Experimentos computacionais

Nesta seção são apresentados os experimentos computacionais. Dois conjuntos de instâncias foram usados para comparar os resultados do LA-ALNS com os de outros algoritmos da literatura. Nos experimentos são considerados o conjunto de instâncias disponível em Scheduling Research (2005), proposto por Rabadi et al. (2006), e o conjunto disponível em SOA (2011), proposto por Vallada e Ruiz (2011). O algoritmo LA-ALNS foi executado em um computador Core i7, 1,9 Ghz, 6 GB de memória RAM e sistema operacional Windows 7. O algoritmo foi implementado na linguagem JAVA com a IDE Netbeans 8.0.2. Os resultados completos dos experimentos computacionais com o algoritmo LA-ALNS são disponibilizados em `http://www.decom.ufop.br/prof/marcone/projects/upmsp.html`

Algoritmo 4.3: BuscaLocal FI_{SDM}

Entrada: s
Saída: s

```
1  Seja M o conjunto de máquinas;
2  Seja K_1 o conjunto M ordenado em ordem decrescente pelos tempos de conclusão;
3  Seja K_2 o conjunto M ordenado em ordem crescente pelos tempos de conclusão;
4  para cada máquina k_1 ∈ K_1 faça
5      para cada tarefa j ∈ k_1 faça
6          para cada máquina k_2 ∈ K_2 faça
7              para cada tarefa i ∈ k_2 faça
8                  se k_1 ≠ k_2 então
9                      Seja s' o resultado da troca de j com i;
10                     se f(s') ≤ f(s) então
11                         s ← s';
12                         Pare a busca;
13                     fim
14                 fim
15             fim
16         fim
17     fim
18 fim
19 Retorne s;
```

Algoritmo 4.4: BuscaLocal FI_{SSM}

Entrada: s
Saída: s

```
1  Seja M o conjunto de máquinas;
2  Seja K_1 o conjunto M ordenado em ordem decrescente pelos tempos de conclusão;
3  para cada máquina k_1 ∈ K_1 faça
4      para cada tarefa j ∈ k_1 faça
5          para cada tarefa i ∈ k_1 faça
6              se i ≠ j então
7                  Seja s' o resultado da troca de j com i;
8                  se f(s') ≤ f(s) então
9                      s ← s';
10                     Pare a busca;
11                 fim
12             fim
13         fim
14     fim
15 fim
16 Retorne s;
```

4.4.1 Experimentos com o conjunto de instâncias de Rabadi

Inicialmente, o algoritmo LA-ALNS foi executado para um conjunto de instâncias disponíveis em Scheduling Research (2005). Este conjunto de instâncias contém problemas com combinações de 20, 40, 60, 80, 100 e 120 tarefas e 2, 4, 6, 8 e 10 máquinas. Este conjunto é dividido em três grupos de nível de domínio, definidos abaixo:

1. Grupo *Balanced*: os tempos de processamento e de preparação são balanceados. Ambos foram gerados por distribuição uniforme $U[50, 100]$;

2. Grupo *Process Domain*: os tempos de processamento são dominantes. Os tempos de processamento foram gerados por distribuição uniforme $U[125, 175]$ e os tempos de preparação foram gerados por distribuição uniforme $U[50, 100]$;

3. Grupo *Setup Domain*: os tempos de preparação são dominantes. Os tempos de preparação foram gerados por distribuição uniforme $U[125, 175]$ e os tempos de processamento foram gerados por distribuição uniforme $U[50, 100]$.

4.4.1.1 Comparando os algoritmos LA-ALNS, AIRP, ACO e ACOII

Nesta subseção os resultados do LA-ALNS foram comparados com os resultados do AIRP (Cota et al., 2014a), do ACO (Arnaout et al., 2010) e do ACOII (Arnaout et al., 2014). O AIRP foi executado no mesmo computador acima. O ACO e ACOII não foram re-implementados, considera-se os resultados reportados em Arnaout et al. (2014). Neste trabalho os algoritmos foram executados em um computador Pentium 4 com 2 GB de RAM.

Como o critério de parada no LA-ALNS e no AIRP é o tempo limite (ou $t_{\max}$), foi aplicado o tempo médio de execução usado no ACOII (Arnaout et al., 2014). Este tempo é dividido pelo fator de conversão 2,18. Em PassMark (2017) é verificado que o computador usado nas execuções com o LA-ALNS e o AIRP é 2,18 vezes mais rápido que o computador usado com o ACO e o ACOII. O tempo limite máximo utilizado com o LA-ALNS e o AIRP é de 6 minutos e 30 segundos.

A métrica usada para comparar os quatro algoritmos é o *relative percentage deviation* (RPD), que é definida na Eq. (4.3).

$$RPD_i = \frac{\bar{f}_i^{Alg} - f_i^*}{f_i^*} \times 100 \tag{4.3}$$

O $\bar{f}_i^{Alg}$ é o valor da solução encontrada pelo algoritmo Alg para a i-ésima instância. O f_i^* é um ponto de referência para a i-ésima instância. Os pontos de referência utilizados são as soluções encontradas pelo algoritmo ACO, que estão disponíveis em Scheduling Research (2005) e Arnaout et al. (2010).

Como o LA-ALNS e o AIRP têm natureza estocástica, estes algoritmos foram executados cinco vezes para cada instância, e apenas o valor médio é considerado. O ACOII foi executado apenas uma vez para cada instância, como descrito em Arnaout et al. (2014).

Na Tabela 4.1 são mostrados os resultados médios da métrica RPD para os três algoritmos. As colunas m e n representam o número total de máquinas e de tarefas, respectivamente. Cada combinação de máquinas e tarefas têm 15 instâncias, sendo um total de 1350 instâncias.

Os valores identificados por x indicam instâncias que não estão disponíveis em Scheduling Research (2005) e, por isso, não foram utilizadas nos experimentos. Os melhores resultados estão destacados em negrito. Os resultados negativos indicam que os resultados encontrados foram melhores que os pontos referência (f^*) utilizados, ou seja, os resultados do algoritmo ACO. Pode-se observar que o LA-ALNS foi o melhor na maioria dos casos, isto é, em 67%. Na Figura 4.5a é apresentado o diagrama de caixa dos resultados. Este diagrama sugere que o algoritmo LA-ALNS tem os melhores resultados em mais casos.

Testes estatísticos foram realizados para verificar se existe diferença estatística entre os resultados. A Análise de Variância (ANOVA) [1] (Montgomery, 2007) foi aplicada com 95% de confiança ($threshold = 0,05$) e foi encontrado como resultado $p\text{-}value = 0,0001$. O valor de p é um indicativo forte que existe diferença estatística entre os resultados. Para identificar estas diferenças foi usado o teste Tukey HSD (Montgomery, 2007) com 95% de confiança. Os resultados deste teste são mostrados na Figura 4.5b.

O teste Tukey HSD indica que os resultados do LA-ALNS são diferentes estatisti-

[1] A premissa de normalidade dos dados foi verificada usando o teste Shapiro-Wilk (Shapiro e Wilk, 1965).

Tabela 4.1: Resultados médios para os algoritmos LA-ALNS, AIRP e ACOII.

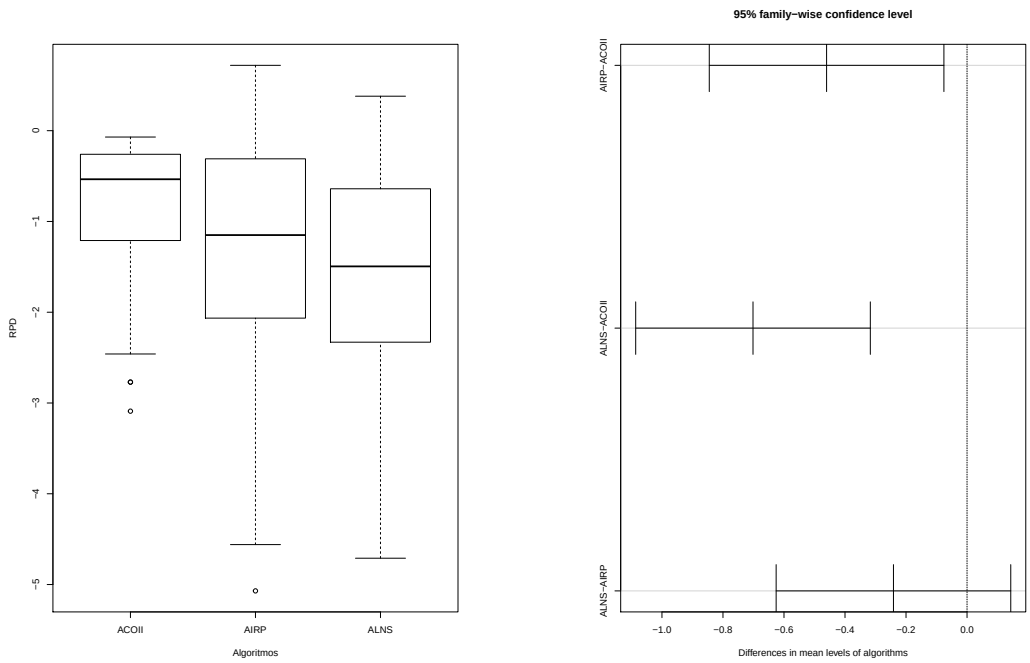
m	n	Balanced			Process			Setup		
		ACOII	AIRP	LA-ALNS	ACOII	AIRP	LA-ALNS	ACOII	AIRP	LA-ALNS
2	20	-0,20	-0,17	**-0,21**	-0,07	-0,08	**-0,11**	-0,17	-0,15	**-0,18**
	40	**-0,12**	0,22	0,04	**-0,18**	0,03	-0,05	**-0,24**	-0,06	-0,14
	60	**-0,26**	0,33	0,15	**-0,13**	0,24	0,13	**-0,16**	0,17	0,05
	80	**-0,16**	0,63	0,38	**-0,10**	0,37	0,24	**-0,10**	0,28	0,18
	100	**-0,27**	0,60	0,16	-1,04	-1,43	**-1,72**	-0,14	-0,39	**-0,66**
	120	**-0,14**	0,72	0,30	-0,16	-0,45	**-0,71**	-0,11	-0,37	**-0,61**
4	20	-1,24	**-1,30**	**-1,30**	-0,43	**-0,73**	-0,67	-0,72	-0,92	**-0,93**
	40	-1,10	-1,63	**-1,99**	-0,49	-0,71	**-0,91**	-0,38	-0,92	**-1,09**
	60	-0,71	-0,98	**-1,37**	-0,25	-0,41	**-0,69**	-0,33	-0,68	**-0,94**
	80	-0,42	-0,33	**-0,92**	-0,16	-0,17	**-0,50**	-0,21	-0,41	**-0,73**
	100	-0,43	-0,26	**-0,99**	-0,26	-0,18	**-0,62**	-0,26	-0,13	**-0,57**
	120	-0,45	-0,07	**-0,69**	-0,24	-0,29	**-0,50**	-0,12	0,02	**-0,38**
6	20	**-1,52**	-1,51	-1,45	-0,53	**-0,57**	-0,54	-0,53	**-0,71**	-0,66
	40	-1,67	-2,91	**-3,16**	-1,03	-1,68	**-1,81**	-0,63	-1,55	**-1,66**
	60	-1,35	-2,50	**-2,88**	-0,49	-1,50	**-1,76**	-0,46	-1,35	**-1,56**
	80	-0,96	-1,71	**-2,31**	-1,45	-1,52	**-1,59**	-1,01	-1,20	**-1,44**
	100	-0,79	-1,14	**-2,04**	-0,32	-0,70	**-1,34**	-0,47	-0,92	**-1,55**
	120	-0,88	-0,78	**-1,63**	-0,32	-0,56	**-0,91**	-0,75	-0,81	**-1,15**
8	20	-2,11	**-2,29**	-2,07	x	x	x	-0,92	**-1,64**	-1,54
	40	-3,09	**-4,07**	-3,92	x	x	x	-1,42	**-2,47**	-2,42
	60	-1,49	-2,71	**-3,18**	x	x	x	-1,36	-1,99	**-2,02**
	80	-0,99	-2,14	**-2,67**	x	x	x	-0,51	-1,64	**-1,87**
	100	-1,19	-2,55	**-3,25**	x	x	x	-0,54	-1,64	**-1,93**
	120	-0,93	-1,52	**-2,20**	x	x	x	-0,29	-1,04	**-1,43**
10	20	-2,77	**-3,60**	-3,06	-1,33	-2,65	-2,42	-1,23	**-2,60**	-2,40
	40	-1,94	**-5,07**	-4,39	-0,61	-2,74	-2,35	-0,66	**-2,97**	-2,58
	60	-2,77	-4,56	**-4,71**	-2,07	**-2,96**	-2,92	-2,13	**-2,79**	-2,79
	80	-2,46	-3,11	**-3,68**	-1,87	-2,33	**-2,49**	-1,93	-2,22	**-2,66**
	100	-1,32	-2,83	**-3,71**	-0,48	-1,47	**-2,04**	-0,72	-1,67	**-2,24**
	120	-1,13	-1,98	**-3,06**	-0,52	-1,47	**-1,82**	-0,55	-1,16	**-1,56**

camente dos resultados do ACOII. Apesar do algoritmo LA-ALNS ter obtido o melhor desempenho na maioria dos casos, os seus resultados não são diferentes significativamente dos resultados do algoritmo AIRP.

4.4.1.2 Comparando os algoritmos LA-ALNS e Branch&Check

Nesta seção os resultados do LA-ALNS foram comparados com os do algoritmo *Branch&Check* (Tran et al., 2016). Esta comparação está sendo realizada separada porque em Tran et al. (2016), os autores usam uma métrica diferente e apenas um subconjunto de instâncias de Rabadi.

A métrica usada para comparar os dois algoritmos é o RPD, Eq. (4.3), no entanto com o uso de um ponto de referência (f^*) diferente. O ponto de referência é definido pelas Eqs.(4.4), (4.5) e (4.6).



(a) Diagrama de caixa dos resultados. (b) Teste Tukey HSD dos resultados.

Figura 4.5: Resultados com o conjunto de instâncias de Rabadi para os algoritmos LA-ALNS, AIRP e ACOII.

$$LB1 = \frac{1}{m}\sum_{j=1}^{n}\min[p_{ij} + S_{ijk}] \qquad \forall i \in M, k \in N \tag{4.4}$$

$$LB2 = \max\left\{\min[p_{ij} + S_{ijk}]\right\} \qquad \forall j, k \in N, i \in M \tag{4.5}$$

$$LB = \max\left\{LB1, LB2\right\} \tag{4.6}$$

Estes pontos de referência foram propostos por Al-Salem (2004) e estão disponíveis em Scheduling Research (2005) para todas as instâncias do conjunto de Rabadi.

Na Tabela 4.2 são apresentados os resultados médios da métrica RPD para os dois algoritmos. Em Tran et al. (2016) estão disponíveis as informações do tempo limite e do computador usado para executar o *Branch&Check*, um Intel Core i7 3,00 GHz com 12 GB de memória RAM. Nesta Tabela, o tempo limite de Tran et al. (2016) é multiplicado pelo fator de conversão 3,28, porque de acordo com PassMark (2017), o computador usado nas execuções com o *Branch&Check* é 3,28 mais rápido que o computador usado com o ACO e o ACOII. Na tabela são considerados apenas os resultados do *Branch&Check*

que usam tempo limite similar aos usados com o LA-ALNS.

Tabela 4.2: Resultados médios para os algoritmos LA-ALNS e Branch&Check.

Grupo Balanced							
Máquinas	Tarefas	LA-ALNS		Branch&Check			
		RPD	Tempo(segundos)	*RPD*	Tempo(segundos)	*RPD*	Tempo(segundos)
2	40	2,31	58,34	2,21	98	**2**	196,8
	60	1,98	116,97	1,51	98	**1,4**	196,8
	80	1,80	191	2,02	98	**1,7**	196,8
	100	**1,51**	287,33	12,07	98	3,62	196,8
	120	**1,36**	400,46	24,29	98	8,13	196,8
4	60	**3,84**	88,62	8,74	98	4,33	196,8
	80	**3,45**	142,84	60,07	98	5,59	196,8
	100	**2,96**	255,69	Infinito	98	70,63	196,8
	120	**2,75**	333,58	Infinito	98	Infinito	196,8
6	100	**4,25**	249,63	Infinito	98	Infinito	196,8
	120	**3,73**	345,41	Infinito	98	Infinito	196,8
8	120	**4,35**	378,44	Infinito	98	Infinito	196,8

Os melhores resultados estão destacados em negrito. Observa-se que o algoritmo LA-ALNS encontrou melhores resultados em mais casos que o *Branch&Check*. Verifica-se então, a eficiência do LA-ALNS para o conjunto de instâncias de Rabadi. Nesta fase dos experimentos não foram realizados testes estatísticos porque o *Branch & Check* não encontrou uma solução viável em todas as combinações de instâncias para o limite de tempo especificado (identificado pelo termo infinito na Tabela 4.2).

4.4.1.3 Comparando o algoritmo LA-ALNS com um maior tempo de execução

Nesta subseção o algoritmo LA-ALNS é avaliado com um maior tempo limite para a execução. Para isso, é criada a configuração LA-ALNS* que tem o dobro do tempo limite do LA-ALNS, usado na Subseção 4.4.1.1. Os resultados do LA-ALNS* são comparados aos do algoritmo AIRP, que obteve melhor desempenho que os algoritmos ACO e ACOII anteriormente. A métrica utilizada para comparação dos algoritmos é a mesma usada na Subseção 4.4.1.1.

Nestes experimentos o algoritmo LA-ALNS* foi executado cinco vezes para cada instância e apenas o melhor resultado foi considerado. A Tabela 4.3 mostra os resultados desta comparação.

Os melhores resultados estão destacados em negrito e os resultados identificados

Tabela 4.3: Resultados para os algoritmos LA-ALNS* e AIRP.

m	n	Balanced		Process		Setup	
		AIRP	LA-ALNS*	AIRP	LA-ALNS*	AIRP	LA-ALNS*
2	20	-0,17	**-0,22**	-0,08	**-0,11**	-0,15	**-0,20**
	40	0,22	**-0,11**	0,03	**-0,05**	-0,06	**-0,28**
	60	0,33	**0,01**	0,24	**0,13**	0,17	**-0,10**
	80	0,63	**0,13**	0,37	**0,24**	0,28	**0,04**
	100	0,60	**0,03**	-1,43	**-1,72**	-0,39	**-0,73**
	120	0,72	**0,29**	-0,45	**-0,71**	-0,37	**-0,61**
4	20	-1,30	**-1,44**	-0,73	**-0,67**	-0,92	**-0,98**
	40	-1,63	**-2,19**	-0,71	**-0,91**	-0,92	**-1,24**
	60	-0,98	**-1,65**	-0,41	**-0,69**	-0,68	**-1,08**
	80	-0,33	**-1,22**	-0,17	**-0,50**	-0,41	**-0,85**
	100	-0,26	**-1,18**	-0,18	**-0,62**	-0,13	**-0,66**
	120	-0,07	**-0,70**	-0,29	**-0,50**	0,02	**-0,38**
6	20	-1,51	**-1,52**	**-0,57**	-0,54	**-0,71**	**-0,71**
	40	-2,91	**-3,55**	-1,68	**-1,81**	-1,55	**-1,92**
	60	-2,50	**-3,19**	-1,50	**-1,76**	-1,35	**-1,83**
	80	-1,71	**-2,80**	-1,52	**-1,59**	-1,20	**-1,80**
	100	-1,14	**-2,25**	-0,70	**-1,34**	-0,92	**-1,64**
	120	-0,78	**-1,65**	-0,56	**-0,91**	-0,81	**-1,34**
8	20	-2,29	**-2,30**	x	x	**-1,64**	-1,63
	40	-4,07	**-4,51**	x	x	-2,47	**-2,73**
	60	-2,71	**-3,51**	x	x	-1,99	**-2,39**
	80	-2,14	**-2,95**	x	x	-1,64	**-2,13**
	100	-2,55	**-3,39**	x	x	-1,64	**-2,04**
	120	-1,52	**-2,16**	x	x	-1,04	**-1,45**
10	20	-3,60	**-3,64**	**-2,65**	-2,42	-2,60	**-2,67**
	40	-5,07	**-5,16**	**-2,74**	-2,35	-2,97	**-3,06**
	60	-4,56	**-5,53**	**-2,96**	-2,92	-2,79	**-3,11**
	80	-3,11	**-4,26**	-2,33	**-2,49**	-2,22	**-2,85**
	100	-2,83	**-3,88**	-1,47	**-2,04**	-1,67	**-2,35**
	120	-1,98	**-3,05**	-1,47	**-1,82**	-1,16	**-1,85**

por x indicam as instâncias que não estão disponíveis em Scheduling Research (2005). Observa-se o algoritmo LA-ALNS* obteve os melhores resultados na maiorias dos casos. Analisando os resultados completos, o LA-ALNS* foi o melhor em 95% dos casos. Um diagrama de caixa destes resultados é apresentado na Figura 4.6.

Para verificar se existe diferença estatística entre os resultados foi aplicado o teste ANOVA [2] com 95% de confiança ($threshold = 0,05$). Neste teste foi encontrado o resultado $p\text{-}value = 0,0223$, este valor de p é um indicativo forte que existe diferença estatística entre os resultados.

Apesar da configuração LA-ALNS* ter o dobro do tempo limite do AIRP, ela mostra que o algoritmo LA-ALNS tende a encontrar melhores soluções quando se tem um maior tempo de execução.

[2] A premissa de normalidade dos dados foi verificada usando o teste Shapiro-Wilk.

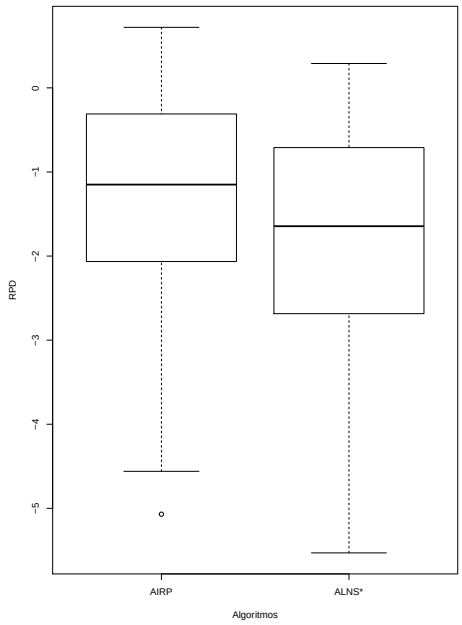


Figura 4.6: Diagrama de caixa dos resultados dos algortimos LA-ALNS* e AIRP.

4.4.2 Experimentos com o conjunto de instâncias de Vallada & Ruiz

O conjunto de instâncias de Vallada & Ruiz está disponível em SOA (2011). Este conjunto de instâncias contém problemas com 50, 100, 150, 200 e 250 tarefas, e 10, 15, 20, 25 e 30 máquinas. Cada combinação de tarefas e máquinas contém 40 instâncias. Neste conjunto a primeira tarefa alocada a cada máquina tem sempre custo de preparação igual a zero. Os tempos de processamento de ambas as combinações foram gerados pela distribuição uniforme $U[1, 99]$. Os tempos de preparação consistem em quatro grupos diferentes, cada um gerado por uma distribuição uniforme diferente, como descrito a seguir: *i*) Grupo 1: $U[0, 9]$; *ii*) Grupo 2: $U[0, 49]$; *iii*) Grupo 3: $U[0, 99]$; and *iv*) Grupo 4: $U[0, 24]$.

4.4.2.1 Comparando os algoritmos LA-ALNS, AIRP e GA2

Nesta subseção os resultados do LA-ALNS foram comparados com os do AIRP e GA2 (Vallada e Ruiz, 2011). O AIRP e o LA-ALNS foram executados no mesmo computador. Para o GA2, são considerados os resultados disponíveis em Vallada e Ruiz (2011). Neste artigo, o algoritmo GA2 foi executado em um computador Intel Core 2 Duo 2,4 GHz com 2GB de RAM.

O critério de parada para o LA-ALNS, AIRP e GA2 é o tempo limite (ou $t_{\max}$). Este tempo limite é definido pela Eq. 4.7, como em Vallada e Ruiz (2011), onde n é o número de tarefas e m o número de máquinas.

$$t_{\max} = n \times (m/2) \times 50 \quad \text{milissegundos} \tag{4.7}$$

Como pode ser verificado em PassMark (2017), o computador usado nas execuções com o LA-ALNS e AIRP é 1,13 vezes mais rápido que o computador usado com o GA2. Portanto, o tempo de execução com o LA-ALNS e AIRP é dividido pelo fator 1,13. O tempo limite máximo utilizado com o LA-ALNS e o AIRP é de 2 minutos e 45 segundos.

A métrica usada para comparar os três algoritmos é o RPD, como na Eq. (4.3). Contudo, o ponto de referência (f^*) é dado pela melhor solução para cada instância disponível em SOA (2011). A métrica aplicada é a mesma usada em Vallada e Ruiz (2011).

Devido à sua natureza estocástica, os algoritmos LA-ALNS e AIRP foram executados cinco vezes para cada instância, e apenas o valor médio é considerado. O algoritmo GA2 também foi executado cinco vezes para cada instância, como é reportado em Vallada e Ruiz (2011).

Os resultados médios da métrica RPD para os três algoritmos são apresentados na Tabela 4.4. Cada combinação de máquinas e tarefas tem 40 instâncias, sendo um total de 1000 instâncias.

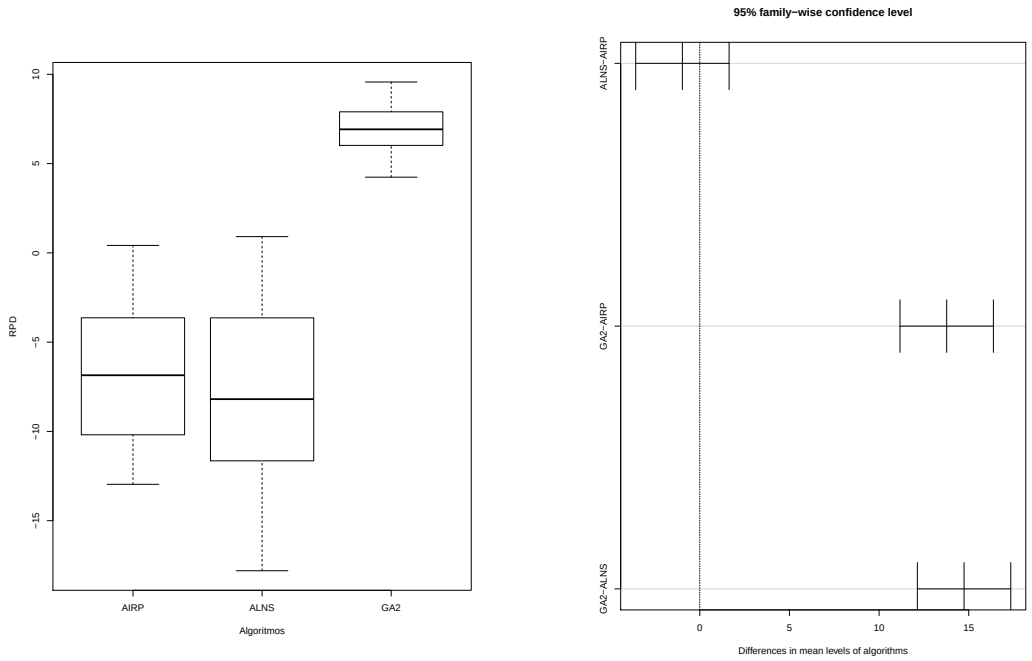
Os melhores resultados estão destacados em negrito. Os resultados negativos indicam que os resultados foram melhores que os pontos de referência. Pode-se observar que o LA-ALNS foi o melhor em mais casos, isto é, em 64% dos casos. O diagrama de caixa destes resultados é mostrado na Figura 4.7a. O diagrama de caixa também indica que o LA-ALNS encontrou os melhores resultados.

O teste ANOVA [3] foi aplicado com 95% de confiança ($threshold = 0,05$) para verificar se existe diferença estatística entre os resultados. Encontrou-se como resultado *p-value* $= 2 \times 10^{-16}$. O valor de p indica que existe diferença estatística entre os resultados. Para identificar estas diferenças foi aplicado o teste Tukey HSD com 95% de confiança.

[3] A premissa de normalidade dos dados foi verificada usando o teste Shapiro-Wilk.

Tabela 4.4: Resultados médios para os algoritmos LA-ALNS, AIRP e GA2.

Máquinas	Tarefas	GA2	AIRP	LA-ALNS
10	50	6,49	-1,07	**-2,11**
	100	5,54	0,42	**-2,14**
	150	5,28	-1,15	**-4,05**
	200	4,24	**-1,91**	-0,44
	250	4,38	**-2,66**	0,91
15	50	9,2	-4,40	**-6,05**
	100	7,32	-3,16	**-6,61**
	150	6,8	-3,64	-3,64
	200	6,21	**-4,54**	-2,16
	250	5,12	**-5,93**	-1,94
20	50	9,57	-6,80	**-8,30**
	100	8,59	-6,62	**-10,77**
	150	7,4	-6,85	**-8,54**
	200	6,71	**-7,64**	-6,24
	250	6,92	**-7,76**	-4,49
25	50	8,1	-8,80	**-11,65**
	100	8,07	-10,50	**-15,30**
	150	7,05	-9,72	**-13,31**
	200	6,82	-10,19	**-10,43**
	250	6,02	**-11,18**	-8,19
30	50	9,4	-8,59	**-11,67**
	100	7,9	-11,40	**-17,80**
	150	7,17	-11,73	**-16,16**
	200	7,09	-12,16	**-12,78**
	250	5,91	**-12,96**	-11,33



(a) Diagrama de caixa dos resultados.

(b) Teste Tukey HSD dos resultados.

Figura 4.7: Resultados com o conjunto de instâncias de Vallada & Ruiz.

Os resultados do teste Tukey HSD são apresentados na Figura 4.7b. Estes resultados indicam que os resultados do LA-ALNS são diferentes estatisticamente dos do algoritmo GA2. Embora o LA-ALNS tenha encontrado os melhores resultados na maioria dos casos, os seus resultados não são diferentes estatisticamente dos do algoritmo AIRP

4.4.2.2 Comparando os algoritmos LA-ALNS e SA

Nesta subseção são realizadas as comparações dos resultados dos algoritmos LA-ALNS e SA (Santos et al., 2016). A comparação com o algoritmo SA é realizada separadamente porque este algoritmo é o único que realizou calibração dos parâmetros para as instâncias de Vallada & Ruiz. A calibração foi realizada usando o pacote IRace.

A métrica utilizada para comparar os algoritmos e o critério de parada do algoritmo LA-ALNS são os mesmos utilizados na Subseção 4.4.2.1. Em Santos et al. (2016) o tempo de execução do algoritmo SA também é dividido por um fator de conversão, já que o computador usado em Santos et al. (2016) é mais rápido que o utilizado em Vallada e Ruiz (2011). Na Tabela 4.5 são apresentados os resultados médios da métrica RPD para os dois algoritmos.

Tabela 4.5: Resultados médios para os algoritmos LA-ALNS e SA.

Máquinas	Tarefas	LA-ALNS	SA
10	50	-2,11	**-2,62**
	100	-2,14	**-8,28**
	150	-4,05	**-12,33**
	200	-0,44	**-16,56**
	250	0,91	**-16,85**
15	50	-6,05	**-5,08**
	100	-6,61	**-10,49**
	150	-3,64	**-14,31**
	200	-2,16	**-19,35**
	250	-1,94	**-22,77**
20	50	-8,30	**-7,50**
	100	-10,77	**-11,50**
	150	-8,54	**-16,30**
	200	-6,24	**-20,76**
	250	-4,49	**-23,65**
25	50	-11,65	**-9,09**
	100	-15,30	**-13,38**
	150	-13,31	**-17,92**
	200	-10,43	**-22,32**
	250	-8,19	**-25,10**
30	50	-11,67	**-9,76**
	100	-17,80	**-15,23**
	150	-16,16	**-18,78**
	200	-12,78	**-23,24**
	250	-11,33	**-26,14**

Os melhores resultados estão destacados em negrito. Observa-se claramente que o algoritmo SA encontrou os melhores resultados em todos os casos. Na Figura 4.8 é apresentado um diagrama de caixa dos resultados, que também ilustra o melhor desempenho do algoritmo SA.

O teste estatístico ANOVA [4] com 95% de confiança ($threshold = 0,05$) foi aplicado para verificar se existe diferença estatística entre os resultados dos dois algoritmos. O resultado encontrado foi $p\text{-}value = 2,53 \times 10^{-5}$, este valor de p dá um indicativo forte de que existe diferença estatística entre os resultados.

O melhor desempenho do algoritmo SA pode ser explicado por duas características importantes, dadas a seguir. A primeira característica é o critério de parada nas instâncias de Vallada & Ruiz, que é muito pequeno. Uma instância com 50 tarefas tem em média apenas 12 segundos para ser resolvida e uma instância com 250 tarefas tem em média apenas 2 minutos. Em um limite de tempo pequeno as heurísticas mais simples tendem a obter melhor desempenho, já que as técnicas de busca empregadas

[4] A premissa de normalidade dos dados foi verificada usando o teste Shapiro-Wilk.

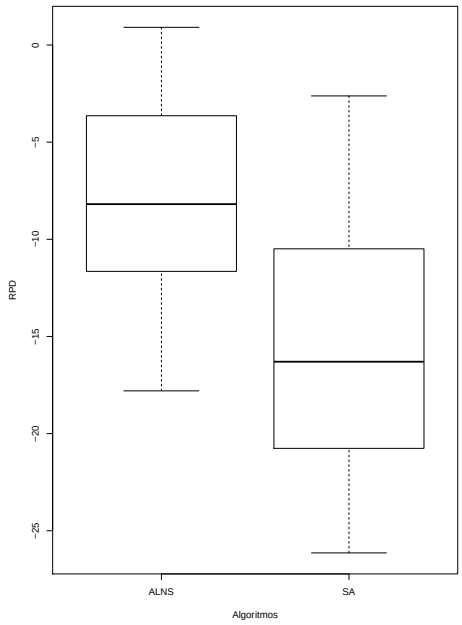


Figura 4.8: Diagrama de caixa dos resultados dos algortimos LA-ALNS e SA.

exigem um menor esforço computacional. Já as heurísticas mais complexas, como o LA-ALNS, precisam de um esforço computacional bem maior para realizar a busca de maneira eficiente. A segunda característica é que o algoritmo SA teve seus parâmetros calibrados especificamente para as instâncias de Vallada & Ruiz, este fato tende a gerar uma heurística especialista para determinado conjunto de instâncias. Por outro lado, o algoritmo com esta calibração de parâmetros pode não ser tão eficiente em outros conjuntos de instâncias.

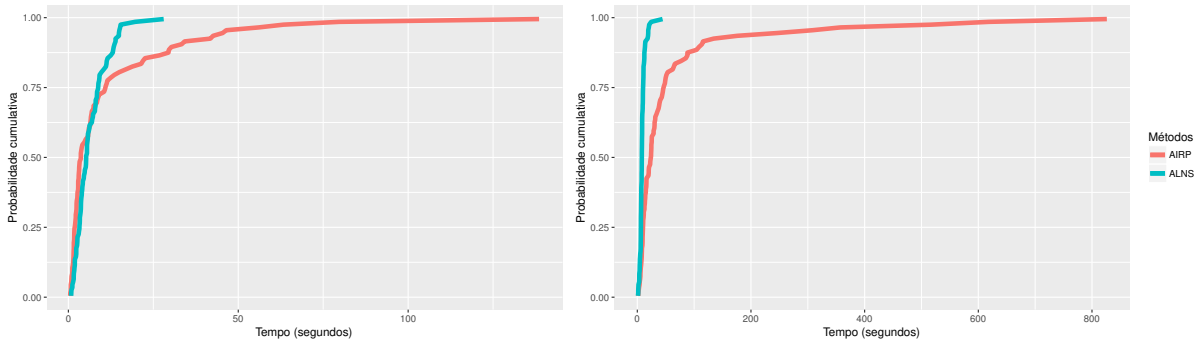
4.4.3 Convergência

Nesta subseção é apresentada uma análise empírica de convergência para o algoritmo LA-ALNS.

Inicialmente, é realizado um teste de probabilidade empírica (Aiex et al., 2007) comparando os resultado dos algoritmos LA-ALNS e AIRP. Para realizar o teste foram selecionadas aleatoriamente várias instâncias dos conjuntos de Vallada & Ruiz e de Rabadi. Dentre estas, foram usadas duas instâncias para ilustrar o comportamento típico dos algoritmos.

Foi definido um alvo para cada instância. Para a instância do conjunto de Vallada & Ruiz, é usado o ponto de referência (f^*) utilizado nos experimentos da Subseção 4.4.2. Para a instância do conjunto de Rabadi, é usado o ponto de referência (f^*) utilizado nos experimentos da Subseção 4.4.1.1. Neste último caso, o ponto de referência é multiplicado pelo fator $0,015$, devido a dificuldade de encontrar a solução alvo em um limite de tempo pequeno.

Os algoritmos LA-ALNS e AIRP foram executados 100 vezes para cada instância. Quando o alvo era encontrado, o algoritmo era finalizado e o tempo gasto era armazenado. Na Figura 4.9 são mostrados os resultados.



(a) Instância com 200 tarefas e 15 máquinas do conjunto de Vallada & Ruiz.

(b) Instâncias com 100 tarefas e 4 máquinas do conjunto de Rabadi.

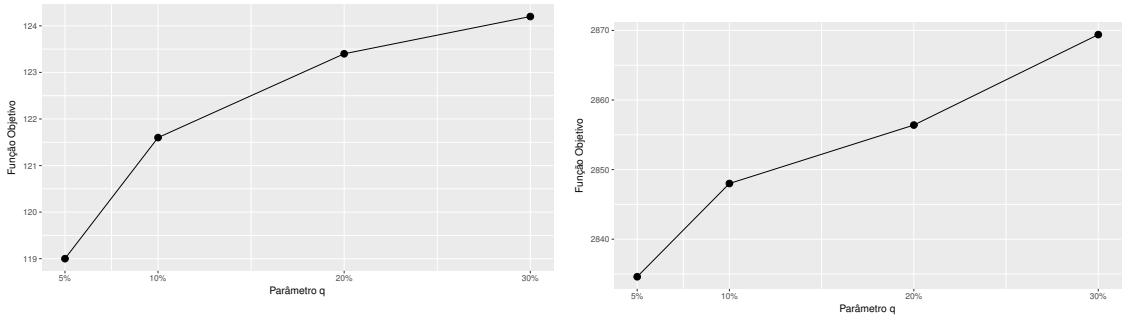
Figura 4.9: Resultados para o teste de probabilidade empírica.

Observa-se que o algoritmo LA-ALNS é capaz de encontrar o alvo sempre em um limite de tempo pequeno, enquanto o algoritmo AIRP precisa de um longo tempo para encontrar o alvo em algumas execuções. Os resultados corroboram a eficiência do LA-ALNS, um algoritmo adaptativo que encontra bons resultados em um limite tempo pequeno.

Além disso, nesta seção é realizado uma análise da diversificação e intensificação da busca no algoritmo LA-ALNS. Um importante elemento neste processo é o parâmetro q. Ele define o número de tarefas que são removidas e inseridas a cada iteração do algoritmo.

Como descrito na Seção 4.3.2, o parâmetro q foi definido como 5% do número total de tarefas na implementação do LA-ALNS. Este valor foi obtido por meio de testes empíricos. Para verificar a influência deste parâmetro na diversificação e intensificação da busca, são testadas variações nos valores de q. As duas instâncias usadas previamente

foram executadas cinco vezes para cada valor diferente do parâmetro q usando o mesmo tempo limite. Os valores usados para q foram 5%, 10%, 20% e 30%. Os resultados médios são mostrados na Figura 4.10.



(a) Instância do conjunto de Vallada & Ruiz.

(b) Instância do conjunto de Rabadi.

Figura 4.10: Resultados para a análise de diversificação e intensificação da busca

Observa-se que para os valores de q maiores que 5% os valores da função objetivo pioraram. Isso ocorre porque as heurísticas de remoção e inserção precisam de mais tempo para processar valores mais altos de q. O uso de altos valores de q aumenta a amplitude da busca em grandes vizinhanças, mas isso torna o processo mais custoso.

4.4.4 Evolução da aprendizagem com autômatos

Esta subseção tem o objetivo de analisar o desempenho dos métodos de remoção e inserção no LA-ALNS.

Para realizar os testes foi selecionado aleatoriamente um conjunto de instâncias de tamanho médio do tipo *Balanced* usado na Tabela 4.1. Neste tipo de conjunto os tempos de preparação e processamento são balanceados. O conjunto com 100 tarefas e 10 máquinas foi selecionado. Estas instâncias foram executadas pelo LA-ALNS com um tempo limite de 4,58 minutos (mesmo limite de tempo usado na Subseção 4.4.1.1). A cada minuto as probabilidades de seleção foram armazenadas.

Nas tabelas 4.6 e 4.7 são mostradas as probabilidade médias de seleção para os métodos de inserção e remoção, respectivamente.

Os gráficos das figuras 4.11 e 4.12 ilustram os resultados das tabelas 4.6 e 4.7, res-

Tabela 4.6: Probabilidades médias de seleção para os métodos de inserção.

Após	Lambda	Gulosa	Semi-Gulosa	Húngaro	ILS	Arrependimento
1 minuto	13,07%	19,68%	18,30%	19,82%	12,86%	16,26%
2 minutos	12,18%	23,24%	18,21%	19,26%	11,99%	15,13%
3 minutos	11,61%	22,67%	16,60%	22,82%	11,88%	14,41%
4 minutos	11,08%	24,78%	15,94%	23,87%	11,13%	13,19%

Tabela 4.7: Probabilidades médias de seleção para os métodos de remoção.

Após	Aleatório	Gulosa	Semi-Gulosa	Máquina Maior Custo	Máquina Aleatória	*Shaw*
1 minuto	14,23%	14,79%	14,12%	18,73%	23,69%	14,43%
2 minutos	13,44%	14,17%	14,03%	24,78%	20,28%	13,30%
3 minutos	13,16%	14,55%	14,34%	27,60%	17,78%	12,57%
4 minutos	11,65%	14,69%	13,49%	30,13%	17,73%	12,30%

pectivamente. Os gráficos ilustram as probabilidades de seleção a partir do minuto 1, é importante destacar que no instante zero todos os métodos têm a mesma probabilidade de seleção.

Entre os métodos de inserção, o método inserção Gulosa e inserção Húngaro contribuíram mais para melhores resultados e suas probabilidades aumentaram ao longo do tempo. Os bons resultados do método de inserção Gulosa podem ser explicados porque sempre faz alocações nas melhores posições. Os bons resultados do método de inserção Húngaro podem ser explicados porque este método é capaz de resolver subproblemas do problema principal otimamente em um tempo polinomial. Esses resultados contribuem para validar uma das hipóteses deste trabalho, que está relacionado à aplicabilidade do método húngaro para o problema.

Entre os métodos de remoção, a remoção "máquina maior custo" contribuiu mais para o melhor desempenho. Este método sempre remove tarefas da máquina que tem o maior custo, que é o *makespan*.

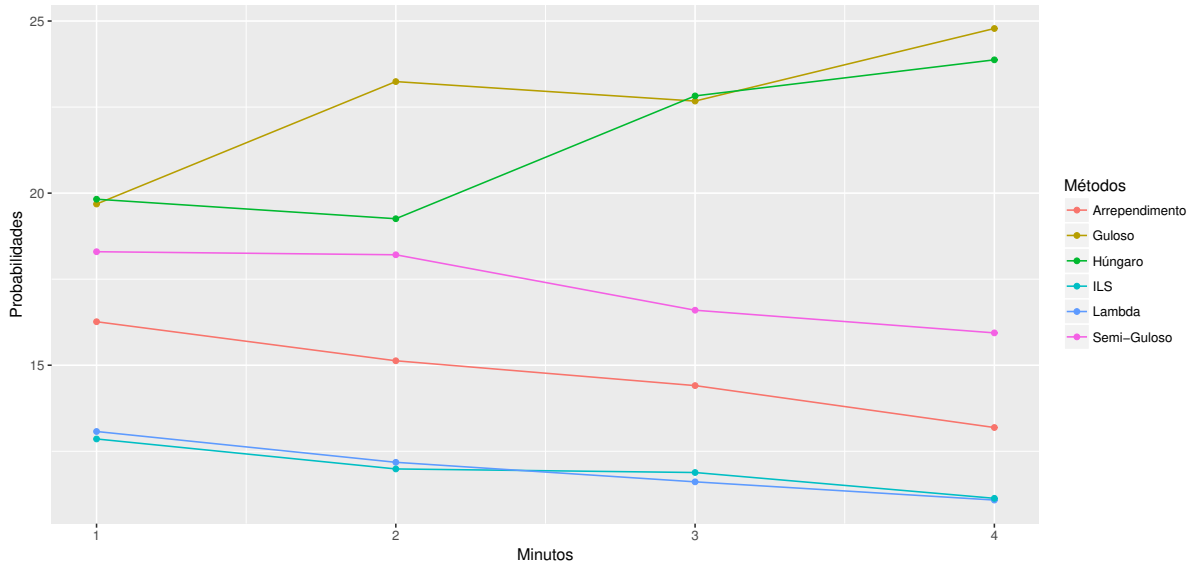


Figura 4.11: Probabilidades dos métodos de inserção.

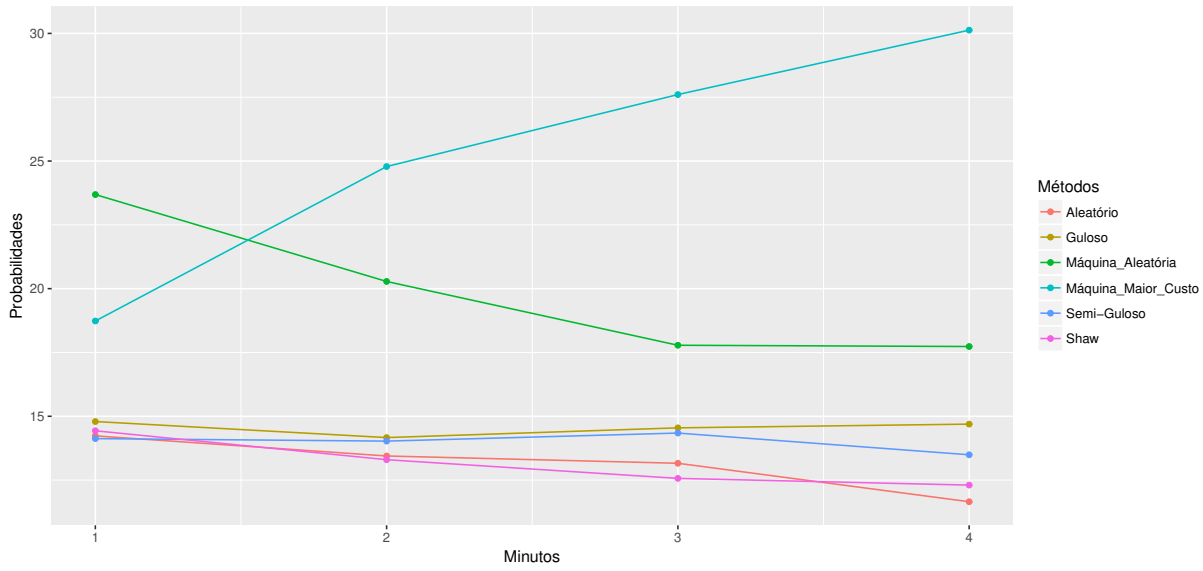


Figura 4.12: Probabilidades dos métodos de remoção.

4.5 Conclusão

Neste capítulo é tratada a resolução do problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência com o objetivo de minimizar o *makespan*. Um algoritmo adaptativo é proposto para resolver diferentes variações do problema. Este algoritmo é um *Adaptive Large Neighborhood Search* usando aprendizado com autômatos para aprender e ajustar as probabilidades das heurísticas de remoção e inserção ao longo da busca. Neste algoritmo foram implementados seis métodos de remoção e seis métodos de inserção. Estes métodos são usados para explorar o espaço de busca como uma perturbação. Um dos principais métodos de inserção é inspirado no método Húngaro e é capaz de resolver subproblemas otimamente em tempo polinomial. Um *Random Variable Neighborhood Descent* é usado para realizar as buscas locais. O uso da aprendizagem com autômatos no *Adaptive Large Neighborhood Search* e a aplicação do método Húngaro no problema abordado são as principais contribuições desta parte do trabalho.

Como o foco deste capítulo foi propor um algoritmo adaptativo, foram usados dois conjuntos diferentes de instâncias da literatura nos experimentos computacionais. Ambos os conjuntos são muito diferentes em tamanho e características. A maioria dos algoritmos na literatura emprega apenas um desses conjuntos para comparação e resultados. Os resultados do algoritmo proposto foram comparados com os de seis algoritmos da literatura. O LA-ALNS encontrou os melhores resultados em mais casos para um dos conjuntos de instâncias. No outro conjunto de instâncias o LA-ALNS obteve desempenho pior que apenas um algoritmo da literatura, o algoritmo SA. Mas, o algoritmo SA é o único que teve seus parâmetros calibrados especificamente para um dos conjuntos de instâncias. Este fato favorece o algoritmo no conjunto de instâncias em que realizou o processo de calibração. Um estudo sobre o desempenho dos métodos de inserção e remoção também foi realizado. Este estudo mostra que o inserção Húngaro é realmente um método interessante no ALNS, apresentando bons resultados entre os métodos de inserção propostos.

Pode-se concluir que o algoritmo proposto é eficiente e robusto para resolver diferentes variações do problema. O problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência é um problema geral e aparece em indústrias com grande variabilidade de características. Portanto, o algoritmo proposto aqui é indicado para resolver casos práticos.

Capítulo 5

Modelo matemático multiobjetivo para o $R_M|S_{ijk}|(C_{max}, \text{TEC})$

5.1 Introdução

No setor de manufatura, as fontes de energia predominantes são gás natural e eletricidade. De acordo com a Administração de Informações de Energia dos Estados Unidos, o setor industrial consome cerca de 54% do total de energia entregue no mundo. A energia é utilizada no setor industrial para uma ampla gama de propósitos, tais como processo e montagem, vapor e cogeração, aquecimento e resfriamento de processos, iluminação, aquecimento e ar condicionado para edifícios. O consumo de energia em atividades industriais também está relacionado à emissão de gases do efeito estufa na atmosfera. Portanto, a redução do consumo de energia elétrica nas indústrias é de suma importância.

Nessa direção, o presente capítulo trata da resolução de um problema importante de sequenciamento de máquinas em um contexto de *green scheduling*. O problema abordado é o $R_M|S_{ijk}|(C_{max}, TEC)$ com os objetivos de minimizar o consumo total de energia elétrica e o *makespan* (detalhado na Seção 2.3). Este problema desempenha um papel importante na indústria de manufatura, permitindo que os gerentes maximizem a produtividade em relação aos recursos disponíveis e as tarefas a serem processadas. Para a resolução do $R_M|S_{ijk}|(C_{max}, TEC)$, um modelo matemático multiobjetivo é construído, implementado e analisado. A fim de abordar o modelo proposto e gerar um conjunto de soluções não dominadas de boa qualidade, em um tempo computacional restrito, é usado o método *Multi-Objective Smart Pool Search Matheuristic* (ou *Smart Pool*), recen-

temente proposto em Coelho et al. (2016a), e o método clássico ϵ-restrito. Basicamente, o método *Smart Pool* transforma um problema multiobjetivo em múltiplos problemas de soma ponderada de um único objetivo.

A demais seções do capítulo seguem organizadas como a seguir. Na Seção 5.2 é apresentado o modelo matemático multiobjetivo proposto. O método *Smart Pool* é tratado na Seção 5.3. Os experimentos computacionais são discutidos na Seção 5.4. Ao final, na Seção 5.5, são tratadas as conclusões deste capítulo.

5.2 Modelo matemático proposto

Em contraste às abordagens clássicas do problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência, como mostrado nos trabalhos Avalos-Rosales et al. (2013, 2015); Rabadi et al. (2006); Vallada e Ruiz (2011), que minimizam apenas o *makespan*, novas características são adicionadas à formulação proposta neste trabalho. Esses recursos são necessários para considerar o consumo de energia elétrica, como velocidade de processamento variável e potência da máquina para diferentes velocidades. A inclusão destas características é inspirada no trabalho de Mansouri et al. (2016a) que trata um problema de sequenciamento de máquinas *flow shop*. O novo modelo matemático proposto neste capítulo também é baseado no modelo proposto por Avalos-Rosales et al. (2015).

Abaixo, a notação do novo modelo matemático é dada:

- $M = \{1, ..., m\}$: conjunto de máquinas, sendo m o número de máquinas;
- $N = \{1, ..., n\}$: conjunto de tarefas com n representado o número de tarefas;
- $L = \{1, ..., o\}$: conjunto de o diferentes modos de operação, cada modo está relacionado a uma velocidade de operação e consumo de potência correspondentes (denominado simplesmente modo de operação);
- $N_0 = N \cup \{0\}$: conjunto de tarefas com inclusão da tarefa fictícia 0;
- p_{ij}: tempo de processamento da tarefa j na máquina i [minutos];
- S_{ijk}: tempo de preparação necessário para alocar a tarefa k na máquina i após a tarefa j [minutos];

- B: constante suficientemente grande;
- π_i: consumo de potência da máquina i na velocidade normal de operação [kW];
- v_l: constante multiplicadora da velocidade na operação normal, com $l \in L$;
- λ_l: constante multiplicadora da potência na operação normal, com $l \in L$.

Neste trabalho é assumido que as constantes v_l e λ_l são as mesmas em todas as máquinas. Os valores destas constantes são inspirados no estudo proposto em Ahilan et al. (2013), que também é usado por Mansouri et al. (2016a,b).

Como exemplo, se a tarefa j tem um tempo de processamento de 100 unidades de tempo em uma determinada máquina i e a constante multiplicadora da velocidade $v_l = 0,8$, então o tempo de processamento de j, dado por (p_{ij}/v_l), torna-se 125 unidades de tempo. Se a constante multiplicadora da velocidade for $v_l = 1,2$, então o tempo de processamento de j torna-se 83,3. Da mesma forma, a velocidade da máquina e a potência influenciam no consumo de energia, assim, em nosso modelo, assumimos que quanto maior a velocidade, maior o consumo de energia. Na velocidade normal de operação, $v_l = 1,0$ e $\lambda_l = 1,0$, o tempo de processamento da tarefa j na máquina i é dado por p_{ij}, enquanto o consumo de energia é dado por π_i. A constante v_l é uma função não decrescente de λ_l. A relação entre as constantes v_l e λ_l é dada abaixo.

$$v_l \text{ e } \lambda_l = \begin{cases} v_l = 1 \text{ e } \lambda_l = 1, & \text{velocidade normal de operação da máquina} \\ 0 < v_l < 1 \text{ e } 0 < \lambda_l < 1, & \text{velocidade mais lenta que a normal, então} \\ & \text{a máquina consome menos potência} \\ v_l > 1 \text{ e } \lambda_l > 1, & \text{velocidade mais rápida que a normal, então} \\ & \text{a máquina consome mais potência} \end{cases}$$

As variáveis de decisão usadas no modelo matemático são:

$$x_{ijkl} = \begin{cases} 1, & \text{se a tarefa } k, \text{ no modo de operação } l, \text{ está alocada imediatamente} \\ & \text{após a tarefa } j \text{ na máquina } i \\ 0, & \text{caso contrário} \end{cases}$$

As variáveis auxiliares usadas no modelo são:

- C_j: tempo de conclusão da tarefa j;
- O_i: tempo de conclusão da máquina i;
- $C_{\max}$: tempo máximo de conclusão de todas as tarefas (ou *makespan*);
- TEC: consumo total de energia [kWh].

Este modelo usa uma tarefa fictícia 0 alocada no início de cada máquina. Esta tarefa tem tempo de processamento $p_{i0} = 0 \ \forall i \in M$ e de preparação $S_{i0k} = 0 \ \forall i \in M, \forall k \in N$. O modelo matemático é dado pelas Eqs. (5.1) a (5.12):

$$\min C_{\max} \tag{5.1}$$

$$\min TEC \tag{5.2}$$

Sujeito a:

$$\sum_{i=1}^{m} \sum_{\substack{j=0 \\ j \neq k}}^{n} \sum_{l=1}^{o} x_{ijkl} = 1 \qquad \forall k \in N \tag{5.3}$$

$$\sum_{i=1}^{m} \sum_{\substack{k=1 \\ j \neq k}}^{n} \sum_{l=1}^{o} x_{ijkl} \leq 1 \qquad \forall j \in N \tag{5.4}$$

$$\sum_{k=1}^{n} \sum_{l=1}^{o} x_{i0kl} \leq 1 \qquad \forall \ i \in M \tag{5.5}$$

$$\sum_{\substack{k=0 \\ k \neq j}}^{n} \sum_{l=1}^{o} x_{ijkl} - \sum_{\substack{h=0 \\ h \neq j}}^{n} \sum_{l=1}^{o} x_{ihjl} = 0 \qquad \forall j \in N, \forall i \in M \tag{5.6}$$

$$C_k - C_j + B\left(1 - x_{ijkl}\right) \geq S_{ijk} + \frac{p_{ik}}{v_l} \qquad \forall j \in N_0, \forall k \in N, \ j \neq k, \forall l \in L, \ \forall i \in M \tag{5.7}$$

$$C_0 = 0 \tag{5.8}$$

$$\sum_{j=0}^{m}\sum_{\substack{k=1\\k\neq j}}^{n}\sum_{l=1}^{o}\left(S_{ijk}+\frac{p_{ik}}{v_l}\right)x_{ijkl}=O_i \qquad \forall i\in M \tag{5.9}$$

$$C_{\max}\geq O_i \qquad \forall i\in M \tag{5.10}$$

$$TEC\geq\sum_{i=1}^{m}\sum_{j=0}^{n}\sum_{\substack{k=1\\j\neq k}}^{n}\sum_{l=1}^{o}\left(\lambda_l\times\frac{\pi_i}{60}\times\frac{p_{ik}}{v_l}\right)x_{ijkl} \tag{5.11}$$

$$x_{ijkl}\in\{0,\,1\} \qquad \forall j\in N_0,\forall k\in N,\; j\neq k,\forall i\in M,\; \forall l\in L \tag{5.12}$$

Os objetivos deste modelo matemático são minimizar o *makespan* (5.1) e o consumo total de energia (5.2). As restrições (5.3) garantem que cada tarefa será alocada a apenas uma máquina, além disso, terá apenas uma antecessora e irá trabalhar com um modo de operação único. As restrições (5.4) definem que cada tarefa terá no máximo uma tarefa sucessora. De maneira análoga, as restrições (5.5) garantem que cada tarefa fictícia terá no máximo uma tarefa sucessora. As restrições (5.6) asseguram a ordem correta de alocação das tarefas, se uma tarefa j é antecessora de uma tarefa k, deve existir uma tarefa h que anteceda j. As restrições (5.7) são responsáveis pelo cálculo do tempo acumulado de cada tarefa, se $x_{ijkl} = 1$, então, o tempo acumulado de k (C_k) é igual ao de j (C_j) mais a soma do tempo de preparação da máquina (S_{ijk}) e o tempo de processamento da tarefa k no modo de operação l $\left(\frac{p_{ik}}{v_l}\right)$. Se $x_{ijkl} = 0$, a constante B irá garantir que as restrições sejam satisfeitas. As restrições (5.8) definem o tempo acumulado das tarefas fictícias como 0. Nas restrições (5.9) são realizados os cálculos dos custos acumulados para cada máquina, esse custo é dado pela soma do tempo de preparação e processamento de todas as tarefas alocadas a uma determinada máquina. O valor de $C_{\max}$ é definido pelas restrições (5.10). Nas restrições (5.11) é realizado o cálculo do consumo total de energia (TEC). Para executar este cálculo, é necessário considerar os tempos de processamento (p_{ik}/v_l), a potência de cada máquina na velocidade normal de operação (dado por π_i), e as constantes multiplicadoras (λ_l) e (v_l). A potência da máquina é dividida por 60 porque a potência é dada em kW, enquanto o tempo de processamento é dado em minutos e o TEC é calculado em kWh. Esta restrição é equivalente ao cálculo de integral do consumo de energia. Finalmente, as restrições (5.12) definem que as variáveis são binárias.

O modelo matemático proposto tem n^2mo variáveis binárias e $n + m + 2$ variáveis contínuas. O modelo também tem $2n + 3m + nm + 2n^2mo + 2$ restrições. Portanto, o aumento de velocidades variáveis leva a um aumento linear no número de variáveis binárias e restrições.

5.3 Matheuristic Smart Pool

O método *Multi-Objective Smart Pool Search Matheuristic* (ou *Smart Pool*) (Coelho et al., 2016a) tem o objetivo de encontrar soluções não dominadas perto da fronteira Pareto. Este método do tipo *matheuristic* consiste na resolução de modelos matemáticos multiobjetivo usando otimizações por resolvedores matemáticos, dentro de um limite de tempo predefinido para cada resolução chamada pelo procedimento. Para este fim, são gerados diferentes problemas de PLIM com diferentes pesos para agregar as duas funções objetivo envolvidas na formulação proposta. O nome *Smart Pool* vem justamente do propósito do método. Durante a otimização com resolvedores matemáticos são encontradas várias soluções factíveis ao longo do caminho. A ideia do método *Smart Pool* é testar essas soluções e identificar soluções não dominadas.

O Algoritmo 5.1 apresenta o pseudocódigo do método *Smart Pool*.

Algoritmo 5.1: *Multi-Objective Smart Pool Search Matheuristic*

Entrada: Número de vetores de peso w_{max} e limite de tempo para o resolvedor matemático $timeLim$

Saída: Conjunto de soluções não dominadas Xe

```
1  W ← GeraçãoVetorPesos(w_max);
2  X_e ← ∅ ;
3  para cada (w_1, w_2) ∈ W faça
4  |   model ← modelo PLIM com pesos w_1, w_2 ;
5  |   poolSol, poolEval ← Resolvedor(model,timeLim);
6  |   poolEval ← avaliação de cada solução s ∈ poolSol em relação a ambas as
   |   funções objetivo ;
7  |   para n ← 1 até |poolSol| faça
8  |   |   addSolution(Xe, poolSol[n], poolEval[n])
9  |   fim
10 fim
11 retorna Xe ;
```

Nesta versão do método, os pesos foram gerados usando o método proposto por

Scheffé (Scheffé, 1958) (descrito na Subseção 3.3.4). Este método é muito popular para a geração de pesos em algoritmos evolutivos multiobjetivo, tais como NSGA-III e MOEA/D. Ele é mais eficiente do que o usado na versão original do *Smart Pool* (Coelho et al., 2016a). Isso foi verificado anteriormente com experimentos computacionais usando o modelo matemático proposto neste capítulo. Outras abordagens para a geração de vetores de peso também poderiam ter sido usadas, como por exemplo, a proposta em Meneghini e Guimarães (2017).

No Algoritmo 5.1, linha 4 é gerado um modelo matemático com critério único dado pela soma ponderada dos objetivos com os pesos w_1 e w_2 para os objetivos *makespan* e consumo total de energia, respectivamente. Posteriormente, na linha 5 é chamado o resolvedor caixa-preta para otimizar o modelo. Durante a busca e otimização de cada problema MILP, geralmente, diferentes soluções viáveis podem ser encontradas. Nesse sentido, o conjunto de soluções viáveis obtidas em cada execução do otimizador é retornado e armazenado no conjunto $poolSol$. O parâmetro $timeLim$ define o tempo máximo que pode ser gasto pelo resolvedor na busca de cada problema MILP com um conjunto de pesos exclusivo.

Finalmente, na linha 8 é verificado qual dessas soluções são não dominadas, filtrando $poolSol$ com a dominância Pareto, usando o procedimento *addSolution*, detalhado em Lust e Teghem (2010) e representado no Algoritmo 5.2. Este procedimento tem como parâmetro de entrada um conjunto de soluções não dominadas X_e, uma solução s e uma função de avaliação $z(s)$. O objetivo do método *addSolution* é continuar atualizando o conjunto de soluções não dominadas X_e, inicialmente vazio (linha 2 do Algoritmo 5.1), filtrando as soluções extraídas da árvore BB (ou outro método usado pelo resolvedor matemático) e criando uma aproximação da fronteira Pareto.

Algoritmo 5.2: *addSolution*

Entrada: X_e, s, $z(s)$
Saída: X_e

```
1  Add ← verdadeiro ;
2  para cada  x ∈ X_e faça
3  |  se  z(x) ⪯ z(s) então
4  |  |  Add ← falso ;
5  |  |  Pare ;
6  |  fim
7  |  se  z(s) ≺ z(x) então
8  |  |  X_e ← X_e \ x;
9  |  fim
10 fim
11 se  Add = verdadeiro então
12 |  X_e ← X_e ∪ s;
13 fim
14 retorna X_e ;
```

5.4 Experimentos computacionais

Os experimentos computacionais foram realizados em um computador Core i7, 1,9 GHz, 6 GB de memória RAM e sistema operacional Ubuntu 16.04. O método *Smart Pool* foi implementado no OptFrame 2.3 (Coelho et al., 2011) e o resolvedor utilizado foi o IBM ILOG CPLEX, versão 12.5. Os resultados completos dos experimentos computacionais deste capítulo são disponibilizados em `http://www.decom.ufop.br/prof/marcone/projects/upmsp.html`

5.4.1 Geração de instâncias

Para executar os experimentos, um novo conjunto de instâncias foi criado para a formulação multiobjetivo do $R_M|S_{ijk}|(C_{\max}, TEC)$. Este conjunto foi inspirado nas instâncias propostas por Vallada e Ruiz (2011) e disponíveis em SOA (2011). O novo conjunto de instâncias tem 80 problemas, considerando combinações de 6, 8, 10, 12 e 15 tarefas com

2, 3, 4 e 5 máquinas. As combinações de 6, 8, 10 e 12 tarefas têm 3 modos de operação ($o = 3$), representando velocidade normal, lenta e rápida, enquanto as combinações de 15 tarefas têm 5 modos de operação ($o = 5$). A Tabela 5.1 resume as características das instâncias e a origem destas características.

Tabela 5.1: Características das instâncias.

Parâmetros	Níveis	Baseado em
Número de tarefas (n):	6, 8, 10, 12, 15	Vallada e Ruiz (2011)
Número de máquinas (m):	2, 3, 4, 5	Vallada e Ruiz (2011)
Modos de operação (o):	3 ou 5	Mansouri et al. (2016a) e Ahilan et al. (2013)
Tempo de processamento (p_{ij}):	$U[1, 99]$	Vallada e Ruiz (2011)
Tempos de preparação dependentes da sequência (S_{ijk}):	$U[1, 9]$, $U[1, 49]$, $U[1, 99]$, $U[1, 124]$	Vallada e Ruiz (2011)
Potência da máquina (π_i):	$U[40, 200]$	-
Constante multiplicadora da velocidade (v_l):	$1,2$; $1,1$; 1; $0,9$; $0,8$	Mansouri et al. (2016a) e Ahilan et al. (2013)
Constante multiplicadora da potência (λ_l):	$1,5$; $1,25$; 1; $0,8$; $0,6$	Mansouri et al. (2016a) e Ahilan et al. (2013)

5.4.2 Métodos usados para resolver o modelo matemático proposto

O método ϵ-restrito e o *Smart Pool* (descrito na Seção 5.3) foram usados para resolver o modelo proposto. A justificativa para escolha destes métodos é dada a seguir. O método ϵ-restrito (Hames et al., 1971) é considerado clássico na literatura para resolução de problemas multiobjetivo (Arenales et al., 2015). Este método é capaz de determinar quaisquer soluções pertencentes à fronteira Pareto-ótimo independente de o espaço ser convexo, não convexo ou discreto (Deb, 2001). Além disso, este método tem bom desempenho em problemas de pequeno porte e com um número pequeno de objetivos. Neste capítulo são tratadas apenas instâncias de pequeno e médio porte, com no máximo 15 tarefas. Já a escolha do método *Smart Pool* se deve ao bom desempenho dele na resolução de problemas multiobjetivo de programação linear inteira mista no contexto de *MicroGrids* (Coelho et al., 2016a).

O método ϵ-restrito é um método do tipo escalar. Métodos escalares utilizam técnicas para transformar um problema multiobjetivo em mono-objetivo. O método ϵ-restrito transforma um problema multiobjetivo em mono-objetivo criando restrições adicionais ao problema. Neste método, a cada passo, apenas um dos objetivos é escolhido e mantido na função objetivo, os demais objetivos são transformados em restrições de desigualdade.

A seguir, nas Equações (5.13) e (5.14) é apresentado um exemplo de uso do método ϵ-restrito para a minimização de um problema.

$$\min f_1(x) \tag{5.13}$$

Sujeito a:

$$f_i(x) \leq \epsilon_i, \qquad i = 2, ..., r \tag{5.14}$$

$$x \in \mathcal{R}^n \tag{5.15}$$

Neste problema x é a variável de decisão e tem-se um total de r objetivos. O objetivo f_1 é definido como o mais importante, sendo considerado o único objetivo nesta formulação. Os demais objetivos f_i (com $i = 2, ..., r$) são transformados em restrições de desigualdade. O parâmetro ϵ_i é o limite superior do objetivo f_i. O conjunto Pareto-ótimo pode ser obtido com variações do parâmetro ϵ_i.

Para aplicação nesta pesquisa, inicialmente, geramos para cada instância um modelo matemático com um único objetivo, minimizando apenas o $C_{\max}$ e outro modelo matemático de um único objetivo minimizando apenas o *TEC*. Em cada caso, o outro objetivo foi mantido como uma restrição. Um procedimento semelhante foi feito em Toro et al. (2017), no contexto de problemas de roteamento de veículos multiobjetivo, para verificar qual objetivo deve ser selecionado no método ϵ-restrito. A Tabela 5.2 apresenta o tempo médio (em segundos) gasto para as execuções dos modelos matemáticos de um único objetivo. Estes valores foram agrupados pelas instâncias com o mesmo número de tarefas

Tabela 5.2: Tempo médio para resolver versões com um único objetivo do problema (método ϵ-restrito).

Conjunto de instâncias	Minimizando o $C_{\max}$	Minimizando o *TEC*
6	**0,113**	0,333
8	**0,313**	1,360
10	**1,082**	42,744
12	**6,989**	556,885
15	**224,263**	2402,626

Pode-se observar que o modelo que minimiza o $C_{\max}$ é o mais eficiente. Assim, no método ϵ-restrito, o objetivo de minimizar o *makespan* ($C_{\max}$) é definido como o objetivo mais importante e o consumo total de energia (TEC) é considerado uma restrição.

A implementação do método ϵ-restrito é detalhada a seguir. Inicialmente, foi definido o ponto máximo f_b e o ponto mínimo f_a para o objetivo TEC, em cada instância. Para calcular f_a, foi usado o modelo matemático de objetivo único que minimiza apenas o TEC. Da mesma forma, para calcular o f_b, foi usado o modelo matemático de um único objetivo que minimiza apenas o C_{max}. Assim, foi possível calcular os valores correspondentes do TEC. Dez valores diferentes para ϵ foram criados por uma distribuição uniforme ($U[f_a, f_b]$) para cada instância.

Para o método *Smart Pool*, duas configurações diferentes foram definidas ($SmartPool_1$ e $SmartPool_2$), garantindo que o método resolva um número aproximado de problemas gerados pelo método ϵ-restrito. A configuração $SmartPool_1$ resolve 10 problemas e a configuração $SmartPool_2$ resolve 20 problemas. Cada problema pode ser otimizado com um limite de tempo máximo de 20 segundos. Este limite de tempo foi obtido previamente por testes empíricos. Como mencionado anteriormente neste trabalho, resolvedores matemáticos precisam de um limite de tempo expressivo para resolver instâncias de problemas desta natureza. Com isso, obtivemos um limite de tempo mínimo para resolver cada instância do problema de maneira satisfatória. A Tabela 5.3 mostra as características dessas configurações.

Tabela 5.3: Configurações propostas do *Smart Pool*

Sigla	w_{max}	$timeLim$ (segundos)	número de problemas PLIM
$SmartPool_1$	10	20	10
$SmartPool_2$	20	20	20

5.4.3 Analisando o trade-off entre os dois objetivos

A seguir, são apresentas as estimativas da fronteira Pareto encontradas pelo método ϵ-restrito para duas instâncias selecionadas aleatoriamente. Esses conjuntos de soluções não dominadas são mostrados nas figuras 5.1 e 5.2. Os pontos azuis indicam as soluções Pareto para o $R_M|S_{ijk}|(C_{\max}, TEC)$ e o ponto em vermelho mostra a solução ótima para o $R_M|S_{ijk}|C_{\max}$.

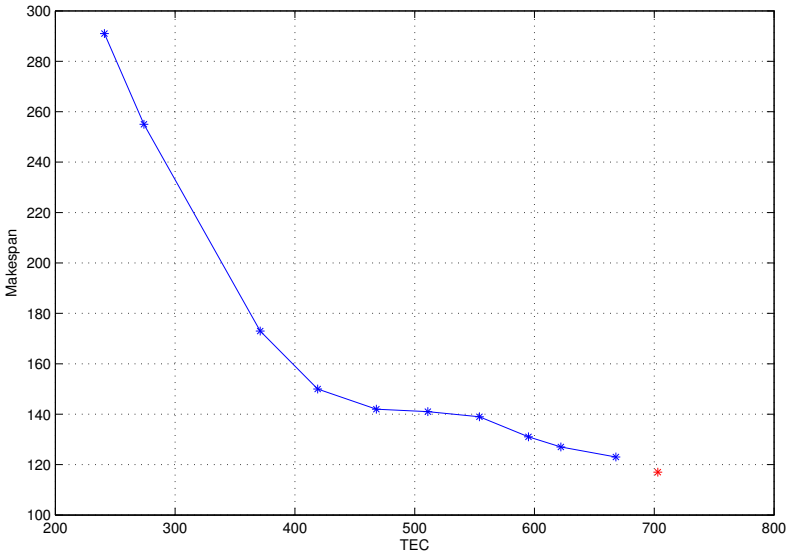


Figura 5.1: Exemplo 1: Fronteira Pareto para uma determinada instância selecionada aleatoriamente

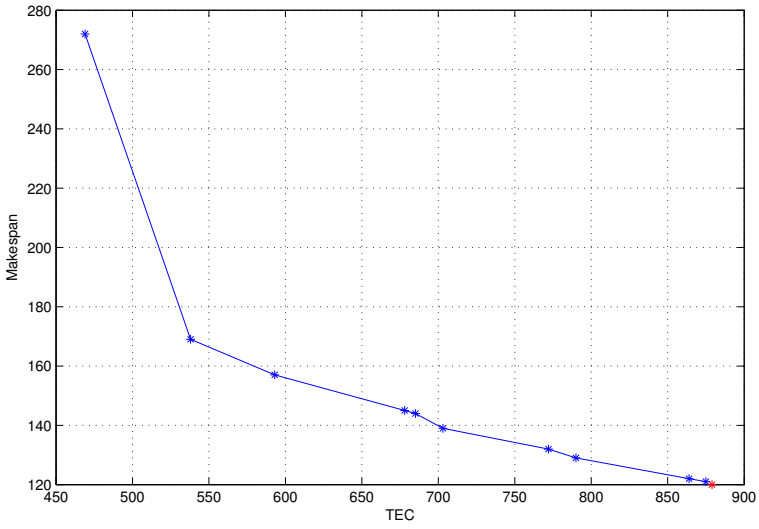


Figura 5.2: Exemplo 2: Fronteira Pareto para uma determinada instância selecionada aleatoriamente

Pode-se observar que há uma grande variabilidade nos valores dos dois objetivos. O ponto na cor vermelho é importante para destacar que somente essa solução seria analisada em uma versão de único objetivo do problema. Conforme ilustrado na Figura 5.1, por exemplo, cerca de 700 unidades de energia seriam necessárias para implementar a solução que minimiza o *makespan* (portanto, maximizando a produção). Estes gráficos ilustram que ambos os objetivos ($C_{\max}$ e TEC) são conflitantes e enfatizam a importância desta abordagem multiobjetivo para o tomador de decisão. Ao selecionar uma solução diferente na fronteira Pareto, o tomador de decisão pode optar por sacrificar algumas unidades de *makespan* para obter um menor consumo de energia.

Para melhor ilustrar o conflito entre os dois objetivos, a Tabela 5.4 mostra os valores máximos e mínimos encontrados, para cada objetivo, pelo método ϵ-restrito. Esses valores estão agrupados pelas instâncias com o mesmo número de tarefas. Pode-se verificar que há uma grande diferença entre os valores máximo e mínimo de ambos os objetivos. Nos conjuntos com 12 tarefas, há uma diferença de 1565 unidades em relação ao consumo total de energia, mostrando a importância desse objetivo para o problema.

Tabela 5.4: Valores máximo e mínimo das soluções encontradas pelo método ϵ-restrito

Conjunto de	$C_{\max}$			TEC		
instâncias	Max	Min	Diferença	Max	Min	Diferença
6	383,0	23,0	360,0	713,0	57,0	656,0
8	359,0	36,0	323,0	1223,0	146,0	1077,0
10	524,0	55,0	469,0	1125,0	212,0	913,0
12	634,0	59,0	575,0	1738,0	173,0	1565,0
15	469,0	54,0	415,0	1649,0	220,0	1429,0

5.4.4 Comparando os resultados dos métodos ϵ-restrito e Smart Pool

A Tabela 5.5 mostra o tempo médio necessário para encontrar as soluções não dominadas pelos métodos ϵ-restrito e *Smart Pool*. O tempo médio é dado pelo tempo gasto (em segundos) divido pelo número de soluções não dominadas encontradas. Esses valores são agrupados pelas instâncias com o mesmo número de tarefas.

Tabela 5.5: Tempo médio para encontrar soluções não dominadas – método ϵ-restrito e as configurações do *Smart Pool*.

Conjunto de instâncias	ϵ-restrito	$SmartPool_1$	$SmartPool_2$
6	0,208 (± 0,073)	**0,079** (± 0,026)	0,150 (± 0,058)
8	0,644 (± 0,289)	**0,312** (± 0,329)	0,779 (± 0,468)
10	1,780 (± 1,421)	**0,807** (± 0,231)	1,277 (± 0,459)
12	3,994 (± 3,781)	**1,371** (± 0,712)	1,953 (± 1,063)
15	54,663 (± 88)	**6,998** (± 5,104)	8,172 (± 5,009)

Os melhores resultados estão destacadas em negrito e para cada resultado é apresentado o desvio padrão entre parênteses. Observa-se que o método $SmartPool_1$ alcançou os melhores resultados em todos os casos. Também pode ser verificado que o método ϵ-restrito levou muito tempo para resolver as instâncias com 15 tarefas.

Para verificar a qualidade dos conjuntos obtidos de soluções não dominadas, dois indicadores de qualidade foram considerados: o hipervolume (HV) e a cobertura entre conjuntos (CS) (do termo *coverage between two sets* no idioma inglês) (Zitzler e Thiele, 1999).

O indicador HV de um conjunto P calcula o volume da região entre os pontos $po_i \in P$ e um ponto de referência rp. Para cada solução $po_i \in P$, um hipercubo hy_i é construído de acordo com o ponto de referência rp. Em problemas de maximização, o ponto de referência geralmente é (0,0), enquanto em problemas de minimização, o limite superior deve ser determinado para cada objetivo. O hipervolume de uma estimativa de fronteira Pareto é a soma dos hipercubos que cada conjunto de soluções contém. Neste capítulo, o ponto de referência rp são os limites superiores $(f_b^{C_{max}}, f_b^{TEC})$ de cada objetivo, obtidos anteriormente para calcular os valores de ϵ para cada instância. Na Figura 5.3 é ilustrado o exemplo do hipervolume para uma instância. Os pontos em vermelho são os limites superiores e o ponto de referência é $rp = (200, 400)$, os pontos em preto são as soluções não dominadas encontradas. Os valores do hipervolume foram normalizados entre 0 e 1, dividindo o somatório dos hipercubos pela área máxima que contém a Pareto, descrito na Eq. (5.16).

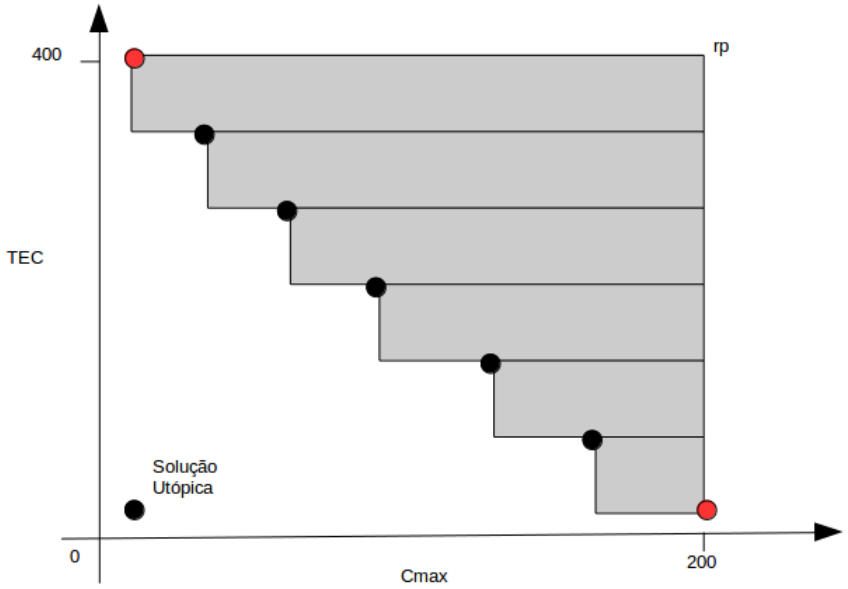


Figura 5.3: Exemplo da métrica de qualidade hipervolume

$$\frac{\sum_{i=1}^{P} hy_i}{(f_b^{C_{max}} - f_a^{C_{max}}) \times (f_b^{TEC} - f_a^{TEC})} \tag{5.16}$$

O indicador CS determina a porcentagem de soluções de outro conjunto (X'') que um determinado conjunto (X') domina. A Eq. (5.17) mostra o cálculo do indicador cobertura entre conjuntos.

$$CS(X', X'') = \frac{|a'' \in X''; \exists a' \in X' : a' \quad \text{cobre} \quad a''|}{|X''|} \tag{5.17}$$

A operação a'cobre a'' determina que a' domina a'' ou a' é igual a a''. Os resultados do indicador CS são mapeados de [0,1]. Se o resultado de CS for igual a 1 indica que todos os pontos X'' são dominados ou iguais aos de X'.

As tabelas 5.6 e 5.7 mostram os resultados obtidos para ambos os indicadores. Os valores também são agrupados pelas instâncias com mesmo número de tarefas.

Tabela 5.6: Resultados médios dos métodos ϵ-restrito e *SmartPool* para o indicador HV.

Conjunto de instâncias	ϵ-restrito	$SmartPool_1$	$SmartPool_2$
6	0,708 ($\pm$ 0,086)	0,810 ($\pm$ 0,070)	**0,821** ($\pm$ 0,061)
8	0,768 ($\pm$ 0,093)	0,814 ($\pm$ 0,118)	**0,826** ($\pm$ 0,116)
10	0,791 ($\pm$ 0,055)	0,845 ($\pm$ 0,078)	**0,858** ($\pm$ 0,064)
12	0,863 ($\pm$ 0,093)	0,934 ($\pm$ 0,072)	**0,942** ($\pm$ 0,068)
15	0,833 ($\pm$ 0,088)	0,884 ($\pm$ 0,113)	**0,909** ($\pm$ 0,087)

Tabela 5.7: Resultados médios dos métodos ϵ-restrito e *SmartPool* para o indicador CS.

Conjunto de instâncias	ϵ-restrito $\times$ $SmartPool_1$	$SmartPool_1$ $\times$ ϵ-restrito	ϵ-restrito $\times$ $SmartPool_2$	$SmartPool_2$ $\times$ ϵ-restrito
6	0,067 ($\pm$ 0,129)	**0,111** ($\pm$ 0,136)	0,043 ($\pm$ 0,0701)	**0,132** ($\pm$ 0,13)
8	0,097 ($\pm$ 0,096)	**0,116** ($\pm$ 0,098)	0,057 ($\pm$ 0,0978)	**0,116** ($\pm$ 0,093)
10	0,063 ($\pm$ 0,054)	**0,092** ($\pm$ 0,122)	0,053 ($\pm$ 0,0611)	**0,090** ($\pm$ 0,112)
12	**0,062** ($\pm$ 0,065)	0,053 ($\pm$ 0,069)	0,061 ($\pm$ 0,0384)	**0,076** ($\pm$ 0,096)
15	**0,125** ($\pm$ 0,088)	0,027 ($\pm$ 0,057)	**0,084 ($\pm$ 0,0638)**	0,045 ($\pm$ 0,081)

Os melhores resultados estão destacados em negrito. Para a métrica HV, o método $SmartPool_2$ alcançou os melhores resultados em todos os casos, no entanto, os valores são parecidos entre si. Isso indica que todos os métodos apresentaram boa convergência para a verdadeira fronteira Pareto dos problemas. Isso é esperado para o método ϵ-restrito, uma vez que depende da solução exata fornecida pelo resolvedor. No método *Smart Pool*, o resolvedor não funciona necessariamente até a otimalidade, devido ao parâmetro de limite de tempo. No entanto, para as instâncias consideradas, o método obteve uma boa convergência analisando os valores da métrica HV. Quanto ao indicador CS, o método *Smart Pool* também obteve os melhores resultados na maioria dos casos com as duas configurações. Mas, pode-se observar que ambos os métodos alcançaram baixa cobertura em relação ao outro, sugerindo que ambos os métodos combinam soluções de alta qualidade em relação aos conjuntos estimados de Pareto. Para ilustrar melhor os resultados obtidos, as figuras 5.4 e 5.5 apresentam os gráficos de diagrama de caixa.

A seguir, é realizada uma análise gráfica dos métodos ϵ-restrito e *Smart Pool*, usando duas instâncias selecionadas aleatoriamente. Uma instância tem 12 tarefas e 3 máquinas

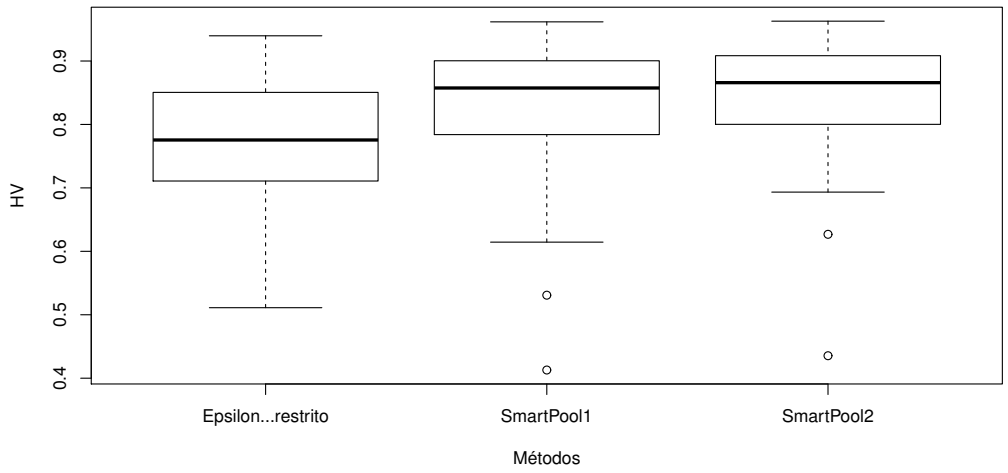


Figura 5.4: Diagrama de caixa dos resultados dos métodos ϵ-restrito e *SmartPool* para o indicador HV

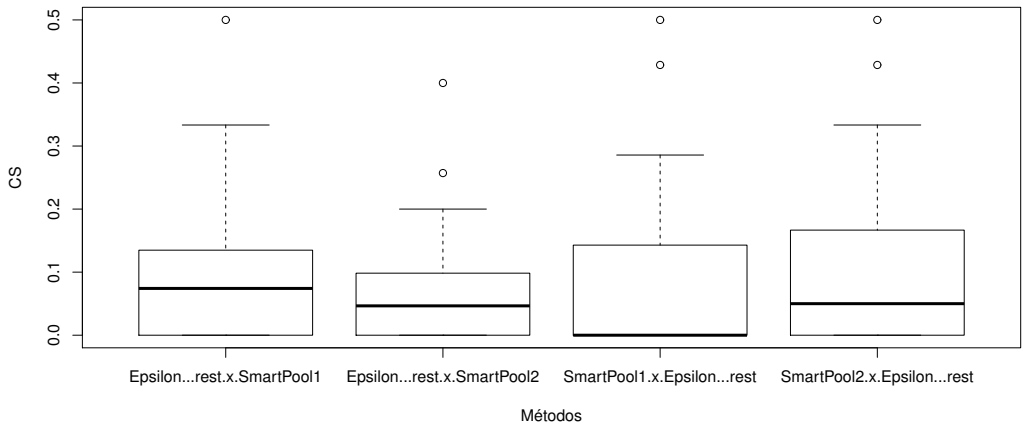


Figura 5.5: Diagrama de caixa dos resultados dos métodos ϵ-restrito e *SmartPool* para o indicador CS

e o outro tem 15 tarefas e 2 máquinas. Uma nova configuração do *Smart Pool* foi criada, o $SmartPool_3$, com $w_{max} = 100$ e $timeLim = 600$ (segundos). Esta proposta de combinação dos parâmetros pode mostrar a evolução do método com mais pesos e limite de tempo longo para resolver cada MILP. As figuras 5.6 e 5.7 mostram os resultados dos métodos.

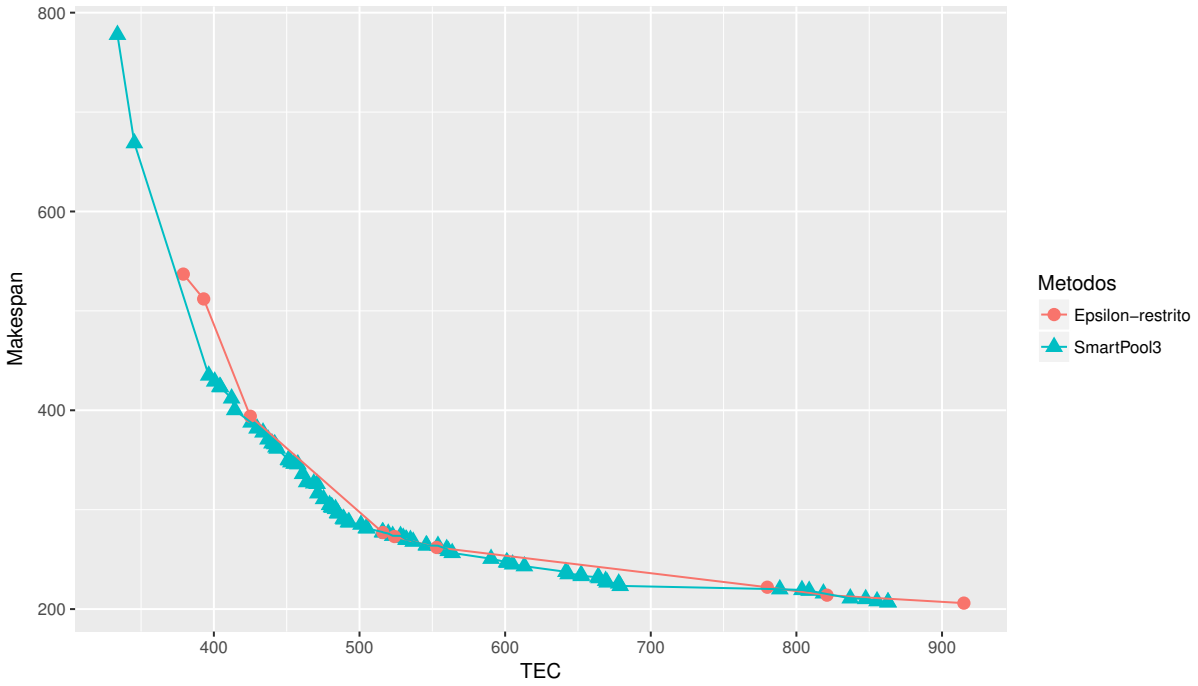


Figura 5.6: Fronteira Pareto obtida para os métodos ϵ-restrito e $SmartPool_3$ para uma instância com 12 tarefas e 3 máquinas.

Pode-se verificar que o método *Smart Pool* conseguiu encontrar uma boa aproximação da fronteira Pareto com um número muito maior de soluções.

5.5 Conclusões

Este capítulo trata a resolução de uma abordagem multiobjetivo do problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência, no contexto de *green scheduling*, em que os principais objetivos são minimizar o *makespan* e o consumo total de energia, formalmente definido por $R_M|S_{ijk}|(C_{\max}, TEC)$.

Motivado pela falta de abordagens semelhantes na literatura, destaca-se a possibili-

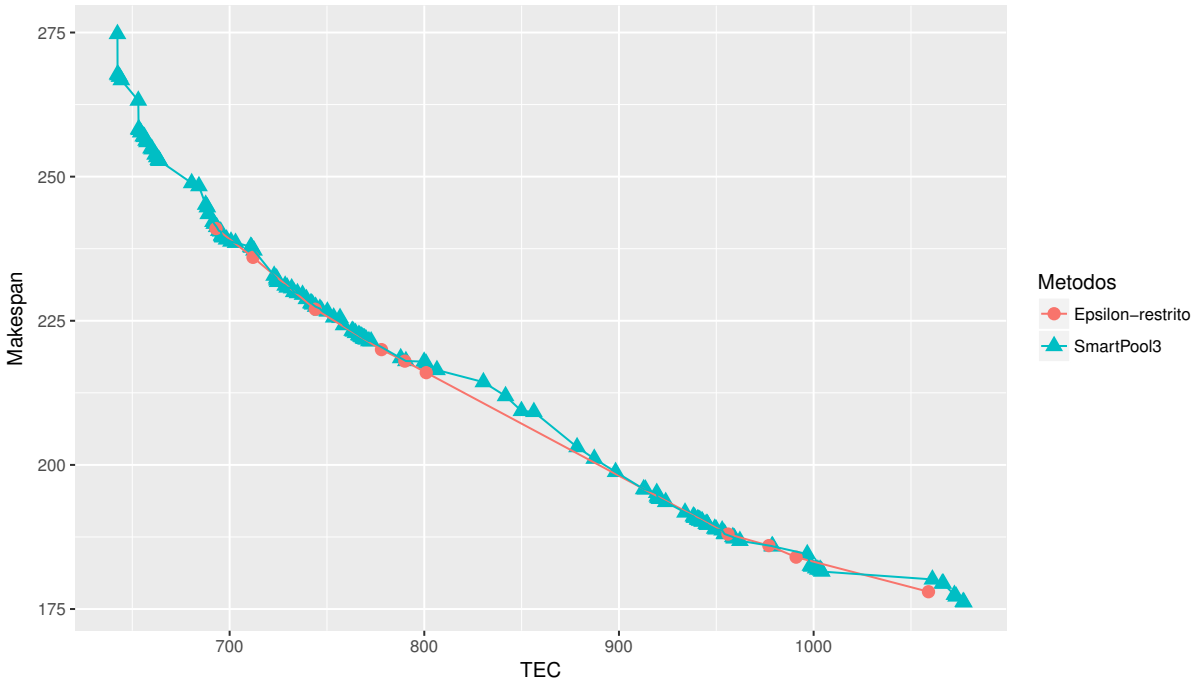


Figura 5.7: Fronteira Pareto obtida para os métodos ϵ-restrito e $SmartPool_3$ para uma instância com 15 tarefas e 2 máquinas.

dade de fornecer sequenciamentos de boa qualidade com baixo consumo de energia. Para este propósito, é proposto um modelo matemático para a resolução do $R_M|S_{ijk}|(C_{\max}, TEC)$. Este modelo foi inspirado em trabalhos recentes da literatura, que estão se movendo para a consideração do consumo de energia em problemas de sequenciamento. O modelo matemático proposto emprega recursos adicionais que possibilitaram a análise do consumo de energia das máquinas, tais como: opções de velocidade variável para o processamento de uma tarefa e as correspondentes entradas de potência da máquina.

Para realizar os experimentos computacionais foi criado um novo conjunto de instâncias. O uso do *Multi-Objective Smart Solution Pool Matheuristic* (*Smart Pool*) foi considerado para encontrar conjuntos de soluções não dominadas em um tempo computacional restrito, e foi comparado ao método clássico ϵ-restrito. Foi verificado e analisado o *trade-off* entre os objetivos *makespan* e o consumo total de energia. Observou-se que o tempo médio, por soluções não dominadas, encontrado pelo *Smart Pool* foi menor do que o método ϵ-restrito, principalmente para as instâncias de maior porte (com 15 tarefas). Assim, em aplicações da vida real, esse método proporciona mais possibilidades para os tomadores de decisão na indústria. O indicadores hipervolume e CS também sugerem uma melhor performance do *Smart Pool*.

Em geral, as contribuições deste capítulo podem ser aplicadas a outros problemas de sequenciamento. O modelo matemático proposto e o método *Smart Pool* podem ser facilmente adaptados a vários outros problemas de sequenciamento de máquinas. Nesta pesquisa, mostra-se que os objetivos *makespan* e consumo total de energia são conflitantes; e que o consumo total de energia é muito importante porque representa um grande custo para a indústria. A eficiência do *Smart Pool* também é revelada, mostrando que esta *matheuristic* pode ser aplicada a inúmeros outros problemas de forma simples.

Capítulo 6

Algoritmos multiobjetivo para o $R_M|S_{ijk}|(C_{max}, TEC)$

6.1 Introdução

Neste capítulo é tratada a resolução de versões de grande porte do problema $R_M|S_{ijk}|(C_{max}, TEC)$ com os objetivos de minimizar o consumo total de energia elétrica e o *makespan* (este problema é detalhado na seção 2.3). No capítulo anterior (Capítulo 5), foram resolvidos apenas problemas de pequeno e médio porte, dada a deficiência dos métodos exatos em resolver problemas grandes em um tempo restrito.

Para tratar o problema em questão são propostos dois algoritmos multiobjetivo, seguindo a mesma ideia do algoritmo LA-ALNS, proposto no Capítulo 4 para resolver o problema mono-objetivo $R_M|S_{ijk}|C_{max}$, que obteve bom desempenho. O primeiro algoritmo proposto é uma versão multiobjetivo do algoritmo LA-ALNS, chamada MO-ALNS. O segundo algoritmo é uma combinação do algoritmo multiobjetivo MOEA/D (Zhang e Li, 2007) (descrito na Subseção 3.3.4) e o algoritmo mono-objetivo LA-ALNS (do Capítulo 4), este algoritmo é chamado MOEA/D + LA-ALNS.

Como descrito no Capítulo 4, o algoritmo ALNS tem sido aplicado com sucesso a diversos problemas de sequenciamento, como problemas aplicados na aviação, na alocação de navios, na alocação de máquinas e no roteamento de veículos. Também são encontradas versões multiobjetivo deste algoritmo para resolver problemas de sequenciamento, como em Rifai et al. (2016), no qual é proposto um ALNS multiobjetivo para resolver um problema de *flow shop*. Inspirados nestas aplicações e no bom desempenho da

versão mono-objetivo para a resolver o problema $R_M|S_{ijk}|C_{max}$, propomos neste capítulo uma versão multiobjetivo do ALNS, chamado MO-ALNS, para resolver o problema em questão.

O algoritmo MOEA/D tem sido amplamente aplicado com sucesso para resolver diversos problemas de otimização multiobjetivo, como pode ser verificado em Trivedi et al. (2017). Devido a este bom desempenho, neste capítulo é proposto um novo algoritmo chamado MOEA/D + LA-ALNS que combina o MOEA/D e o LA-ALNS mono-objetivo (do Capítulo 4). Na versão original do algoritmo MOEA/D um problema multiobjetivo é decomposto em S subproblemas de otimização mono-objetivo usando métodos de decomposição. Os subproblemas são gerados usando vetores de pesos distribuídos uniformemente; a exploração dos subproblemas é realizada por meio de operadores de reprodução. O algoritmo proposto MOEA/D + LA-ALNS usa o algoritmo mono-objetivo LA-ALNS para explorar os subproblemas escalares do MOEA/D ao invés dos operadores de reprodução. A ideia de usar o LA-ALNS veio do bom desempenho do algoritmo para resolver problemas desta natureza.

A demais seções do capítulo seguem organizadas como a seguir. Na Seção 6.2 são apresentados os dois algoritmos propostos, MO-ALNS e MOEA/D + LA-ALNS. Na Seção 6.3 são abordados os experimentos computacionais. As conclusões deste capítulo são tratadas na Seção 6.4.

6.2 Algoritmos propostos

6.2.1 Representação e avaliação da solução

Os dois algoritmos propostos neste capítulo utilizam a mesma representação e avaliação da solução.

Uma solução é representada por duas estruturas de dados, uma para representar as alocações das tarefas nas máquinas e a outra para representar os modos de operação em que cada tarefa é processada. Cada modo de operação está relacionado a uma velocidade de operação e consumo de potência correspondentes.

Para representar a alocação das tarefas é usado um vetor de inteiros de m posições, sendo m o número total de máquinas. Uma lista é associada a cada posição deste vetor, representando as tarefas alocadas a cada máquina (mesma estrutura usada na

Subseção 4.3.1, para versão mono-objetivo do problema). Já os modos de operação são representados por um vetor de inteiros. Na Figura 6.1 é ilustrada uma possível solução para uma instância com duas máquinas, sete tarefas e três modos de operação. Na máquina M_1 estão alocadas as tarefas 7, 3 e 4, nesta ordem. Na máquina M_2 estão alocadas as tarefas 2, 1, 6 e 5, nesta ordem. As alocações dos modos de operação estão da seguinte forma: *i*) modo de operação 1: tarefas 3 e 4; *ii*) modo de operação 2: tarefas 1, 5 e 7; *iii*) e modo de operação 3: tarefas 2 e 6.

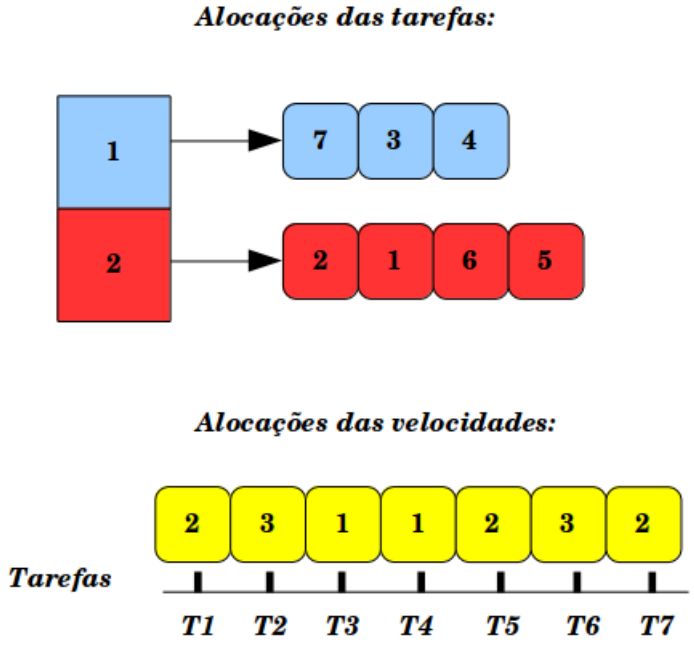


Figura 6.1: Representação de uma solução no MO-ALNS e no MOEA/D + LA-ALNS.

A avaliação de uma solução é dada pelo seu *makespan* (tempo máximo de conclusão de todas as máquinas) e pelo total de consumo de energia, descrito na Eq. 5.11 do capítulo 5.

6.2.2 Algoritmo MO-ALNS

O algoritmo MO-ALNS é uma versão multiobjetivo do LA-ALNS proposto no capítulo 4, por isso, muitas características são idênticas ou similares. O pseudocódigo do MO-ALNS é apresentado no Algoritmo 6.1.

Algoritmo 6.1: MO-ALNS

entrada: $t_{\max}, K, q, a_1, a_2, a_3, b_1$
saída : D

Defina LA $(\phi^-, \alpha^-, \beta, A^-, \pi^-, p^-)$;
Defina LA $(\phi^+, \alpha^+, \beta, A^+, \pi^+, p^+)$;
$s \leftarrow$ `procedimentoConstrutivo` ();
$D \leftarrow$ `addSolution(D,s,f)` ; `/* Conjunto de soluções não dominadas */`
$T \leftarrow (0,05 \times (\sum_{r=1}^{R} 0,5 \times f_r(s)))/\ln 2$;
enquanto *tempoAtual* $\leq t_{\max}$ **faça**
  $s'' \leftarrow null$;
  Selecione α_i^- e α_j^+ usando o método da roleta;
  Remova q tarefas de s usando α_i^-;
  Insira as tarefas removidas de s usando α_j^+;
  $s' \leftarrow$ `MO-RVND(s)`;
  se $(f(s') \prec f(s))$ **então**
    $D \leftarrow$ `addSolution(D,s',f)`;
    $a = a_1$; $b = 0$;
    Atualize p^- e p^+ usando as Equações (4.1)-(4.2);
  fim
  senão se $(f(s') \not\prec f(s)) \wedge (f(s) \not\prec f(s'))$ **então**
    $D \leftarrow$ `addSolution(D,s',f)`;
    $a = a_2$; $b = 0$;
    Atualize p^- e p^+ usando as Equações (4.1)-(4.2);
  fim
  senão se `aleatório` $< \min\left\{1, \exp\left(\frac{\sum_{r=1}^{R} 0,5 \times f_r(s')}{T}\right)\right\}$ **então**
    $s'' \leftarrow s'$;
    $a = a_3$; $b = 0$;
    Atualize p^- e p^+ usando as Equações (4.1)-(4.2);
  fim
  senão
    $a = 0$; $b = b_1$;
    Atualize p^- e p^+ usando as Equações (4.1)-(4.2);
  fim
  Selecione aleatoriamente s em (D, s'');
  se $k \mod K = 0$ **então**
    `atualizaLA`(A^-, A^+, p^-, p^+);
  fim
  $k++$;
  $T \leftarrow 0,99 \times T$;
fim
Retorne D

Os parâmetros de entrada do Algoritmo 6.1 são exatamente os mesmos do algoritmo LA-ALNS do capítulo 4 e possuem os mesmos valores. Inicialmente, é construída uma solução inicial s usando um método construtivo parcialmente guloso muito parecido com o usado no LA-ALNS (descrito na Subseção 4.3.3). A única diferença é que no início do método é selecionado um modo de operação aleatório para o processamento de cada tarefa.

Após a geração da solução inicial, é criado um conjunto de soluções não dominadas D e a solução s é adicionada a ele usando o método *addSolution* (descrito na Subseção 5.3). A seguir, é definida a temperatura inicial. Ela é muito semelhante a usada no LA-ALNS, só que no MO-ALNS cada função objetivo é multiplicada pelo peso 0,5. O fator multiplicador também é diferente, usa-se 0,05 ao invés de 0,1, este valor foi obtido previamente por testes empíricos.

Os passos do Algoritmo a cada iteração são:

1. Um método de remoção $\alpha_i^- \in \alpha^-$ e um método de inserção $\alpha_j^+ \in \alpha^+$ são selecionados por um método do tipo roleta, usando as probabilidades dos métodos. Os métodos de remoção e inserção utilizados são os mesmo do LA-ALNS;

2. q tarefas são removidas da solução corrente com o método α_i^-, e depois, inseridas com o método α_j^+. Estes últimos dois passos são exatamente idênticos as dois primeiros passos do algoritmo LA-ALNS;

3. Um método de busca local multiobjetivo *Random Variable Neighborhood Descent* (MO-RVND) é aplicado à solução corrente s';

4. Após estes passos, assim como no LA-ALNS, as probabilidades p^- e p^+ são atualizadas. Se a solução s' é aceita, usa-se a Eq. (4.1), caso contrário, se a solução s' não é aceita, usa-se a Eq. (4.2). Se a solução corrente s' domina a solução atual s ou se não existe dominação entre ambas, é realizada uma tentativa de inserção da solução s' no conjunto de soluções não dominadas D (usando o método *addSolution*). Se a solução corrente é aceita segundo o critério de temperatura, uma solução s'' recebe a solução s';

5. A seguir, uma nova solução atual s é selecionada aleatoriamente entre todas as soluções do conjunto D e a solução s'';

6. Também como no LA-ALNS, a cada K iterações os valores de probabilidade são

atualizadas dentro de cada LA. A atualização é feita de maneira periódica para dar tempo às LAs de "aprenderem" as melhores ações com o ambiente;

7. Ao final do tempo de execução ($t_{\max}$) é retornado o conjunto de soluções não dominadas D.

Uma solução s' pode ser aceita em três situações diferentes e para cada um delas um parâmetro de recompensa (a_1, a_2 e a_3) é aplicado na atualização das LA. Estas três situações são: *i*) a_1: Se a solução encontrada s' domina a solução atual s; *ii*) a_2: Se não existe dominação entre a solução encontrada s' e a solução atual s; *iii*) a_3: Se a solução encontrada s' é dominada pela solução atual s, mas é aceita de acordo com o critério de temperatura. Caso a solução encontrada não seja aceita, o parâmetro de penalidade (b_1) é usado para atualizar as LA.

6.2.2.1 Procedimento de busca local

O procedimento de busca local usado é uma versão multiobjetivo do *Random Variable Neighborhood Descent* (MO-RVND). Este procedimento é muito parecido com o RVND usado no LA-ALNS (descrito na Subseção 4.3.5).

O MO-RVND utiliza as três estruturas de vizinhanças do RVND e uma estrutura de vizinhança adicional. Esta estrutura adicional é chamada Troca de velocidades – $NS^{SS}(s)$ (a sigla vem do termo *neighborhood search swap speed* no idioma inglês). Um movimento desta vizinhança constitui a troca simples do modo de operação de uma tarefa.

No MO-RVND são usados quatro métodos de busca local selecionados aleatoriamente para explorar o espaço de busca. Cada método de busca local utiliza uma estrutura de vizinhança. As três primeiras buscas locais são muito parecidas com as buscas locais FI_{MI}, FI_{SDM} e FI_{SSM} (descritas na Subseção 4.3.5). A única diferença é no critério de aceitação, na versão multiobjetivo uma solução corrente s' é aceita se ela domina a solução atual s ou se não existe dominação entre ambas. O mesmo se aplica ao critério de aceitação do MO-RVND.

A quarta busca local é chamada FI_{SS}^{MO}, que usa a estrutura de vizinhança $NS^{SS}(s)$ e a estratégia *first improvement*. O pseudocódigo desta busca local é dado no Algoritmo 6.2.

Algoritmo 6.2: Busca Local FI_{SS}^{MO}

Entrada: s
Saída: s
1 Seja M o conjunto de máquinas;
2 Seja L o conjunto de modos de operação em ordem aleatória;
3 Seja K_1 o conjunto M ordenado em ordem decrescente pelos tempos de conclusão;
4 **para cada** *máquina* $k_1 \in K_1$ **faça**
5 **para cada** *tarefa* $j \in k_1$ **faça**
6 **para cada** *modo de operação* $l_1 \in L$ **faça**
7 Seja s' o resultado da troca do modo de operação de j para l_1;
8 **se** $(f(s') \prec f(s)) \vee (f(s') \not\prec f(s) \wedge f(s) \not\prec f(s'))$ **então**
9 $s \leftarrow s'$;
10 Pare a busca;
11 **fim**
12 **fim**
13 **fim**
14 **fim**
15 **Retorne** s;

O seu funcionamento é dado a seguir. Inicialmente, todas as máquinas são classificadas em ordem decrescente pelo tempo de conclusão (conjunto K) e o conjunto de modos de operação (L) é embaralhado. Após isso, as máquinas são selecionadas a partir das máquinas de tempos de conclusão mais longo até as de tempo de conclusão mais curto. Para cada máquina, são analisadas as trocas do modo de operação para todas as tarefas alocadas. Se um vizinho s' domina a solução atual s ou se não existe dominação entre ambas, o método de busca é encerrado e a solução s' é retornada para o MO-RVND; caso contrário, a busca continua até que todas as trocas de modo de operação envolvendo todas as tarefas sejam analisadas.

6.2.3 Algoritmo MOEA/D + LA-ALNS

Uma das limitações do algoritmo MO-ALNS, apresentado na seção anterior, é que ele não tem controle da diversificação na aproximação da fronteira Pareto. Como pode ser verificado na linha (31) do Algoritmo 6.1, o MO-ALNS apenas seleciona uma solução aleatoriamente entre as soluções do conjunto D e a solução s''. Por outro lado, uma das principais qualidades do algoritmo multiobjetivo MOEA/D é a setorização da busca na aproximação da fronteira Pareto. Para fazer isso, ele divide o problema multiobjetivo em S problemas mono-objetivo com base em agregações usando diferentes vetores de

peso.

O algoritmo proposto nesta seção é o MOEA/D + LA-ALNS, que utiliza o LA-ALNS (descrito na Seção 4.3.2) para resolver os subproblemas do MOEA/D. O pseudocódigo do MOEA/D + LA-ALNS é apresentado no Algoritmo 6.3.

Algoritmo 6.3: MOEA/D + LA-ALNS

entrada: $S, T, G, t_{\max}, K, q, a_1, a_2, a_3, b_1$
saída : D

```
1  w ← métodoScheffé(S);
2  B ← geraVizinhos(T, w, S);
3  Pop ← ∅;
4  para i = 1 até S faça
5  |  Pop_i ← procedimentoConstrutivo ();
6  fim
7  z* ← ∅;
8  z* ← atualizaZ(Pop, z*);
9  enquanto iteraçãoAtual ≤ G faça
10 |  para i = 1 até S faça
11 |  |  v ← vizinho selecionado aleatoriamente de B_i;
12 |  |  sol_v ← LA-ALNS(Pop_v, t_max, K, q, a_1, a_2, a_3, b_1);
13 |  |  atualizaZ(sol_v, z*);
14 |  |  para cada vizinho j ∈ B_i faça
15 |  |  |  se g^te(sol_v|w_j, z*) ≤ g^te(Pop_j|w_j, z*) então
16 |  |  |  |  Pop_j ← sol_v;
17 |  |  |  fim
18 |  |  fim
19 |  fim
20 fim
21 para i = 1 até S faça
22 |  D ← addSolution (D, Pop_i, f);
23 fim
24 Retorne D
```

Os parâmetros de entrada do Algoritmo 6.3 podem ser divididos em dois grupos, os parâmetros do MOEA/D e os parâmetros do LA-ALNS. Os parâmetros do MOEA/D são descritos a seguir. O parâmetro S é o total de subproblemas gerados por agregação, T é o total de vizinhos e G é o total de gerações. Já os parâmetros do LA-ALNS são os demais $(t_{\max}, K, q, a_1, a_2, a_3, b_1)$, que são os mesmos descritos na Seção 4.3.2. Os parâmetros foram definidos com os seguintes valores: $S = 50$; $T = 2$; $G = 50$ e $t_{\max} = 20 \times n$ milissegundos, sendo n o número total de tarefas do problema. Estes valores foram obtidos previamente por testes empíricos. Os demais parâmetros possuem os mesmos

valores usados na Seção 4.3.2.

Inicialmente, é construído um vetor de pesos w para todos os subproblemas usando o método de Scheffé (descrito na Subseção 3.3.4 e também utilizado para gerar os pesos do método *Smart Pool* na Subseção 5.3). A seguir, para cada subproblema i, são armazenados no conjunto B_i os T vizinhos mais próximos. O método construtivo parcialmente guloso do algoritmo MO-ALNS (Seção 6.2.2) é usado para gerar os indivíduos da população (Pop). Cada um destes indivíduos gerados aleatoriamente é associado a um subproblema $i \in S$. Após a geração da população, é construído o vetor z^* com a melhor solução para cada objetivo.

Os passos do Algoritmo a cada iteração são:

1. Um vizinho v do subproblema i é selecionado aleatoriamente em B_i;
2. O vizinho v é otimizado usando o algoritmo LA-ALNS. O algoritmo LA-ALNS é o mesmo proposto na Seção 4.3.2 com algumas pequenas mudanças. Não é gerada uma solução inicial na linha (3) do Algoritmo 4.1, porque a solução inicial (Pop_v) é passada como parâmetro pelo MOEA/D. A função de avaliação usada no LA-ALNS e no RVND (incluindo as buscas locais) passa a ser a função de decomposição *Tchebycheff Approach* (ou TCH, descrita na Subseção 3.3.4) aplicada às funções objetivo (*makespan* e consumo total de energia);
3. O vetor de melhores soluções z^* é atualizado;
4. Para cada vizinho j de B_i é analisado se a solução sol_v é melhor que a solução correspondente Pop_j usando a função de decomposição TCH. Em caso positivo, Pop_j recebe a solução sol_v;
5. Estes passos são repetidos até que se esgote o número de gerações (G). A seguir, as soluções não dominadas de Pop são armazenadas no conjunto D, usando o método *addSolution* (descrito na Subseção 5.3). Ao final, é retornado o conjunto D.

O RVND usado no LA-ALNS possui uma busca local adicional para que possam ser analisadas trocas nos modos de operação. Esta nova busca local, chamada FI_{SS}, é muito semelhante a busca local FI_{SS}^{MO} (descrita na Subseção 6.2.2.1). A única diferença é no critério de aceitação. No FI_{SS} a avaliação da solução é feita pela função de decomposição TCH aplicada às funções objetivo do problema, assim como nas demais buscas locais usadas no MOEA/D.

6.3 Experimentos computacionais

Para realizar os experimentos computacionais foi utilizado um computador Core i7, 1,9 GHz, 6 GB de memória RAM e sistema operacional Ubuntu 16.04. Os algoritmos foram implementados na linguagem JAVA com a IDE Netbeans 8.0.2. Os resultados completos dos experimentos computacionais deste capítulo são disponibilizados em `http://www.decom.ufop.br/prof/marcone/projects/upmsp.html`

6.3.1 Validando os algoritmos

Nesta subseção é realizada a validação dos algoritmos MO-ALNS e MOEA/D + LA-ALNS. Para validá-los, eles foram executados para as instâncias de pequeno e médio porte, propostas no capítulo anterior (Tabela 5.1 da Seção 5.4). Os resultados dos algoritmos multiobjetivo são comparados aos resultados ótimos do método ϵ-restrito.

O critério de parada usado para o algoritmo MOEA/D + LA-ALNS é um total de 50 gerações ($S = 50$). Como o critério de parada do algoritmo MO-ALNS é o tempo de execução ($t_{\max}$), adota-se $t_{\max}$ igual ao tempo gasto pelo MOEA/D + LA-ALNS. O tempo máximo de execução com os algoritmos é de 3 horas e 28 minutos.

Para comparar os resultados dos algoritmos multiobjetivo e do método ϵ-restrito é utilizada a métrica hipervolume, que também foi utilizada no capítulo anterior. Nesta subseção, o ponto referência do hipervolume é a maior solução encontrada para cada um dos objetivos, considerando os limites superiores $f_b^{C_{max}}$ e f_b^{TEC} (gerados no capítulo anterior) e todas as soluções encontradas pelos dois algoritmos propostos. Os valores do hipervolume foram normalizados entre 0 e 1, dividindo o somatório dos hipercubos pela área máxima que contém a Pareto.

Devido a natureza estocástica dos algoritmos MO-ALNS e MOEA/D + LA-ALNS, eles foram executados cinco vezes para cada instância, e apenas o valor médio do hipervolume foi considerado. Os resultados do ϵ-restrito foram extraídos da Tabela 5.6 do capítulo anterior.

Na Tabela 6.1 são apresentados os resultados médios dos algoritmos e do método ϵ-restrito. Os valores estão agrupados pelas instâncias com o mesmo número de tarefas.

Os melhores resultados estão destacados em negrito e o desvio padrão de cada re-

Tabela 6.1: Resultados médios do indicador HV com instâncias de pequeno e médio porte.

Conjunto de instâncias	ϵ-restrito	MO-ALNS	MOEA/D + LA-ALNS
6	0,708 ($\pm$ 0,086)	**0,817** ($\pm$ 0,021)	0,755 ($\pm$ 0,046)
8	0,768 ($\pm$ 0,093)	**0,822** ($\pm$ 0,113)	0,791 ($\pm$ 0,139)
10	0,791 ($\pm$ 0,055)	**0,825** ($\pm$ 0,008)	0,810 ($\pm$ 0,013)
12	**0,863** ($\pm$ 0,093)	0,841 ($\pm$ 0,129)	0,831 ($\pm$ 0,131)
15	**0,833** ($\pm$ 0,088)	0,799 ($\pm$ 0,052)	0,832 ($\pm$ 0,133)

sultado é apresentado entre parênteses. Pode-se observar que os algoritmos propostos tiveram bons resultados, sugerindo que ambos têm uma boa convergência para a fronteira Pareto. O teste estatístico ANOVA [1] com 95% de confiança ($threshold = 0,05$) foi aplicado para verificar se existe diferença estatística entre os resultados do indicador HV para os algoritmos MO-ALNS e MO-ALNS/D. O teste encontrou *p-value* igual a $0,1479$, este valor de p é um indicador forte de que não existe diferença estatística entre os resultados. A seguir, é realizada uma análise gráfica da convergência dos algoritmos. Para isso, foram selecionadas duas instâncias aleatoriamente. As figuras 6.2 e 6.3 apresentam os resultados. Estes gráficos também sugerem uma boa convergência dos algoritmos propostos.

6.3.2 Geração de instâncias grandes

Na Subseção 5.4.1 do capítulo anterior foi proposto um conjunto de instâncias de pequeno e médio porte para o problema $R_M|S_{ijk}|(C_{\max}, TEC)$. Já nesta subseção é proposto um conjunto adicional de instâncias de grande porte para o problema em questão. Este conjunto possui 30 problemas, considerando combinações de 50, 100, 150, 200 e 250 tarefas com 10, 20 e 30 máquinas. Todas as combinações têm 5 modos de operação ($q = 5$). Na Tabela 6.2 são apresentadas as características das instâncias.

[1] A premissa de normalidade dos dados foi verificada usando o teste Shapiro-Wilk. O teste ANOVA foi aplicado com blocagem.

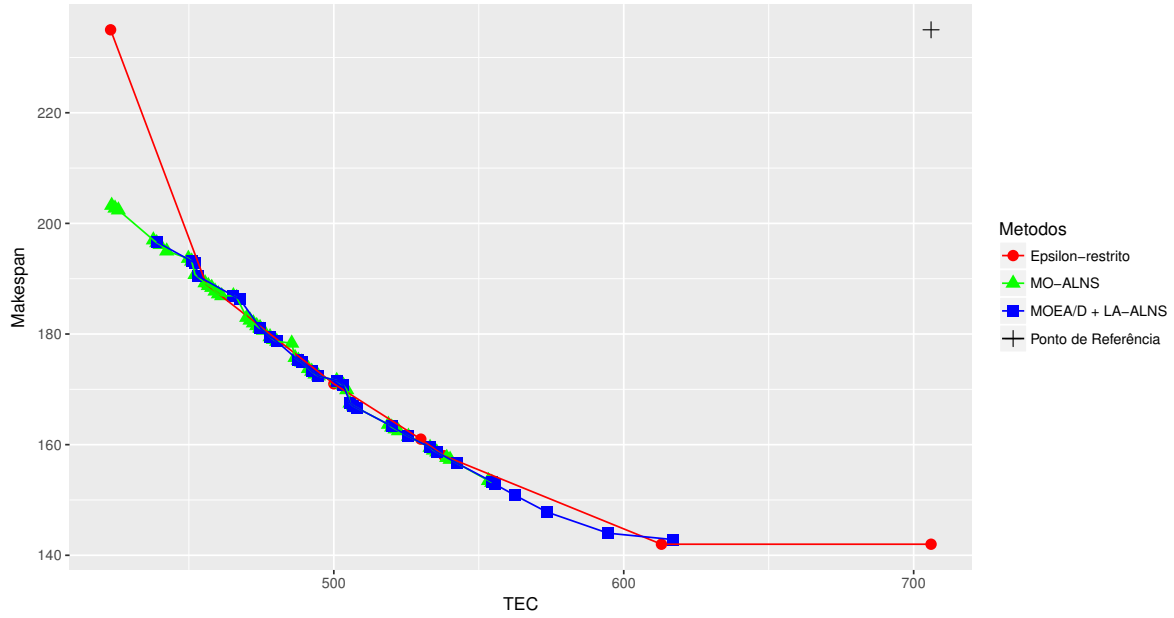


Figura 6.2: Fronteira Pareto obtida com os dois algoritmos e o método ϵ-restrito para uma instância com 8 tarefas e 2 máquinas.

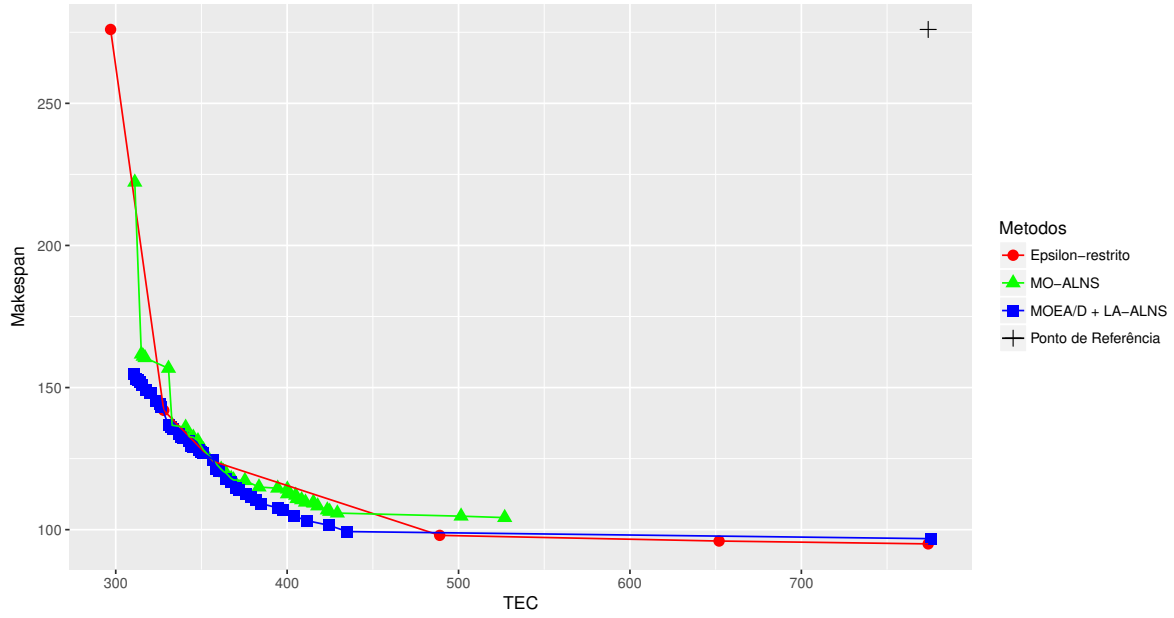


Figura 6.3: Fronteira Pareto obtida com os dois algoritmos e o método ϵ-restrito para uma instância com 12 tarefas e 4 máquinas.

Tabela 6.2: Características das instâncias de grande porte.

Parâmetros	Níveis	Baseado em
Número de tarefas (n):	50, 100, 150, 200, 250	Vallada e Ruiz (2011)
Número de máquinas (m):	10, 20, 30	Vallada e Ruiz (2011)
Número de modos de operação (q):	5	Mansouri et al. (2016a) e Ahilan et al. (2013)
Tempo de processamento (p_{ij}):	$U[1, 99]$	Vallada e Ruiz (2011)
Tempos de preparação dependentes da sequência (S_{ijk}):	$U[1, 9]$, $U[1, 124]$	Vallada e Ruiz (2011)
Potência da máquina (π_i):	$U[40, 200]$	-
Constante multiplicadora da velocidade (v_l):	1,2; 1,1; 1; 0,9; 0,8	Mansouri et al. (2016a) e Ahilan et al. (2013)
Constante multiplicadora da potência (λ_l):	1,5; 1,25; 1; 0,8; 0,6	Mansouri et al. (2016a) e Ahilan et al. (2013)

6.3.3 Comparando adaptações de meta-heurísticas mono-objetivo para resolver problemas multiobjetivo

Nesta subseção é realizada a comparação entre os resultados dos algoritmos MO-ALNS e MO-LNS. Estas meta-heurísticas são adaptações de algoritmos mono-objetivo baseados em busca local para resolver problemas multiobjetivo. O MO-LNS é uma versão multiobjetivo do método *Large Neighborhood Search*.

O MO-LNS implementado usa o método remoção máquina aleatória para remover tarefas e o método inserção gulosa para inserir tarefas. Estes métodos são descritos na Subseção 4.3.4. O procedimento de busca local usado no MO-LNS é o mesmo utilizado no MO-ALNS.

A comparação é realizada utilizando o conjunto de instâncias de grande porte. O critério de parada dos algoritmos é o mesmo usado na seção de validação (Subseção 6.3.1). O indicador hipervolume é utilizado para comparar os resultados dos algoritmos e o ponto de referência é a maior solução para cada objetivo. Todos os resultados dos algoritmos MO-ALNS e MO-LNS foram utilizados para identificar as maiores soluções. Os valores do indicador HV também foram normalizados entre 0 e 1, como na Subseção 6.3.1.

Como os algoritmos têm natureza estocástica, eles foram executados cinco vezes para cada instância e apenas o resultado médio foi considerado. A Tabela 6.3 apresenta os resultados médios dos algoritmos para o indicador HV. Estes valores estão agrupados pelas instâncias com o mesmo número de tarefas.

Tabela 6.3: Resultados do indicador HV para os algoritmos MO-ALNS e MO-LNS.

Conjunto de instâncias	MO-ALNS	MO-LNS
50	**0,752** ($\pm$ 0,038)	0,581 ($\pm$ 0,085)
100	**0,680** ($\pm$ 0,078)	0,490 ($\pm$ 0,074)
150	**0,747** ($\pm$ 0,093)	0,479 ($\pm$ 0,115)
200	**0,724** ($\pm$ 0,049)	0,551 ($\pm$ 0,084)
250	**0,638** ($\pm$ 0,131)	0,490 ($\pm$ 0,145)

Os melhores resultados da Tabela 6.3 estão destacados em negrito e entre parênteses é apresentado o desvio padrão de cada resultado. O algoritmo MO-ALNS encontrou os melhores resultados em todos os casos. Considerando os resultados completos, o MO-ALNS obteve o melhor desempenho em 93% dos casos. Na Figura 6.4 é mostrado o diagrama de caixa destes resultados.

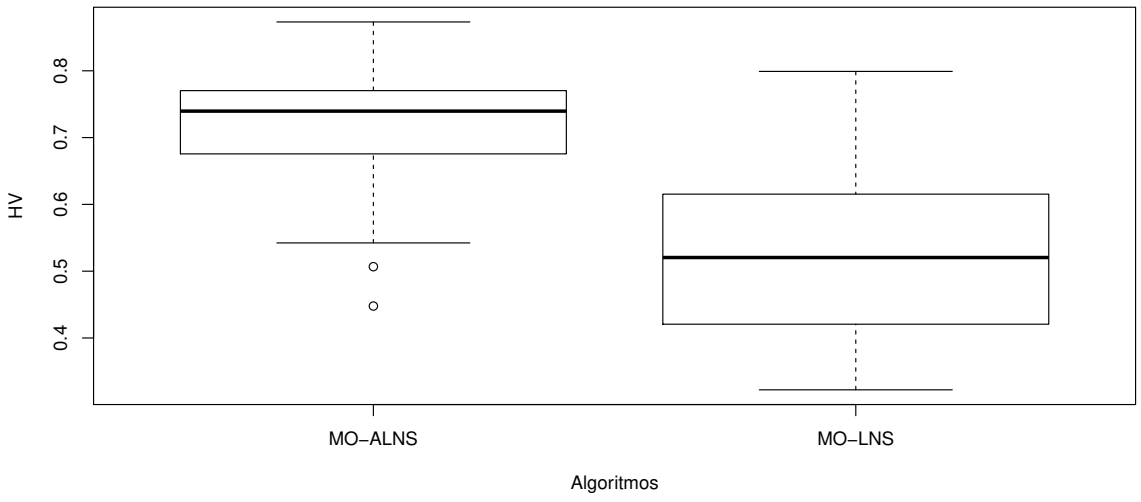


Figura 6.4: Diagrama de caixa dos resultados dos algoritmos MO-ALNS e MO-LNS para o indicador HV.

O diagrama de caixa também sugere um melhor desempenho do algoritmo MO-ALNS/D. Para verificar se existe diferença estatística entre esses resultados é aplicado o teste estatístico ANOVA [2] com 95% de confiança ($threshold = 0.05$). O *p-value*

[2] A premissa de normalidade dos dados foi verificada usando o teste Shapiro-Wilk. O teste ANOVA

encontrado foi 2.25×10^{-9}. Este resultado sugere que existe diferença estatística entre os resultados do indicador HV.

Os resultados desta subseção mostram que o algoritmo MO-ALNS é mais eficaz que o MO-LNS. O MO-ALNS é um algoritmo bem mais completo que contém seis métodos de remoções e seis métodos de inserções. Além de usar a técnica *Learning Automata* para aprender a melhor escolha momentânea dos métodos de inserção e remoção. Portanto, apenas o algoritmo MO-ALNS é usado na comparação com o algoritmo MOEA/D + LA-ALNS na próxima subseção.

6.3.4 Comparando os resultados dos algoritmos MO-ALNS e MOEA/D + LA-ALNS

Nesta subseção é realizada a comparação dos resultados dos algoritmos multiobjetivo MO-ALNS e MOEA/D + LA-ALNS usando as instâncias de grande porte (Tabela 6.2).

O critério de parada usado para os dois algoritmos é o mesmo da fase de validação, para o MOEA/D + LA-ALNS usa-se $S = 50$ e para o MO-ALNS usa-se $t_{\max}$ igual ao tempo gasto pelo MOEA/D + LA-ALNS.

Para comparar os resultados são utilizadas as métricas de convergência hipervolume e CS (descritas na Seção 5.4 do capítulo anterior). O ponto de referência utilizado no hipervolume é a maior solução encontrada para cada objetivo, considerando todos resultados encontrados pelos algoritmos MO-LNS, MO-ALNS e MOEA/D + LA-ALNS. Os valores da métrica hipervolume também foram normalizados entre 0 e 1, como na subseção anterior. Os algoritmos foram executados cinco vezes para cada instância e apenas os resultados médios das métricas são considerados.

Nas tabelas 6.4 e 6.5 são mostrados os resultados médios obtidos para ambos os indicadores. Os valores estão agrupados pelas instâncias com o mesmo número de tarefas.

Os melhores resultados das tabelas 6.4 e 6.5 estão destacados em negrito. O desvio padrão de cada resultado é apresentado entre parênteses. O algoritmo MOEA/D + LA-ALNS alcançou os melhores resultados na maioria dos casos para a métrica HV. Analisando os resultados completos, ele foi o melhor em 70% dos casos. Para a métrica

foi aplicado com blocagem.

Tabela 6.4: Resultados médios do indicador HV para os algoritmos MO-ALNS e MOEA/D + LA-ALNS.

Conjunto de instâncias	MO-ALNS	MOEA/D + LA-ALNS
50	0,791 (± 0,039)	**0,883** (± 0,026)
100	0,780 (± 0,052)	**0,833** (± 0,033)
150	0,782 (± 0,074)	**0,823** (± 0,050)
200	**0,742** (± 0,090)	0,724 (± 0,082)
250	0,665 (± 0,137)	**0,770** (± 0,097)

Tabela 6.5: Resultados médios do indicador CS para os algoritmo MO-ALNS e MOEA/D + LA-ALNS.

Conjunto de instâncias	MO-ALNS × MOEA/D + LA-ALNS	MOEA/D + LA-ALNS × MO-ALNS
50	0,049 (± 0,033)	**0,362** (± 0,101)
100	0,085 (± 0,049)	**0,267** (± 0,076)
150	0,060 (± 0,035)	**0,194** (± 0,092)
200	0,080 (± 0,053)	**0,181** (± 0,120)
250	0,148 (± 0,085)	**0,180** (± 0,082)

CS, o algoritmo MOEA/D + LA-ALNS também encontrou os melhores resultados na maioria dos casos. Considerando os resultados completos, ele foi o melhor em 76% dos casos. Pode-se observar que as soluções do MOEA/D + LA-ALNS cobrem mais partes da fronteira Pareto que não são cobertas pelas soluções do MO-ALNS. Nas figuras 6.5 e 6.6 são apresentados os diagramas de caixa destes resultados.

Os gráficos também sugerem um melhor desempenho do algoritmo MOEA/D + LA-ALNS para os experimentos realizados. Para melhor analisar os resultados da métrica hipervolume, foi aplicado um teste estatístico com o objetivo de verificar se existe diferença estatística entre os resultados. O teste aplicado foi o ANOVA [3] com 95% de confiança ($threshold = 0,05$). Foi encontrado como resultado $p\text{-}value = 0,0111$, como p é menor que o $threshold$, tem-se um indicativo forte que existe diferença estatística

[3] A premissa de normalidade dos dados foi verificada usando o teste Shapiro-Wilk. O teste ANOVA foi aplicado com blocagem.

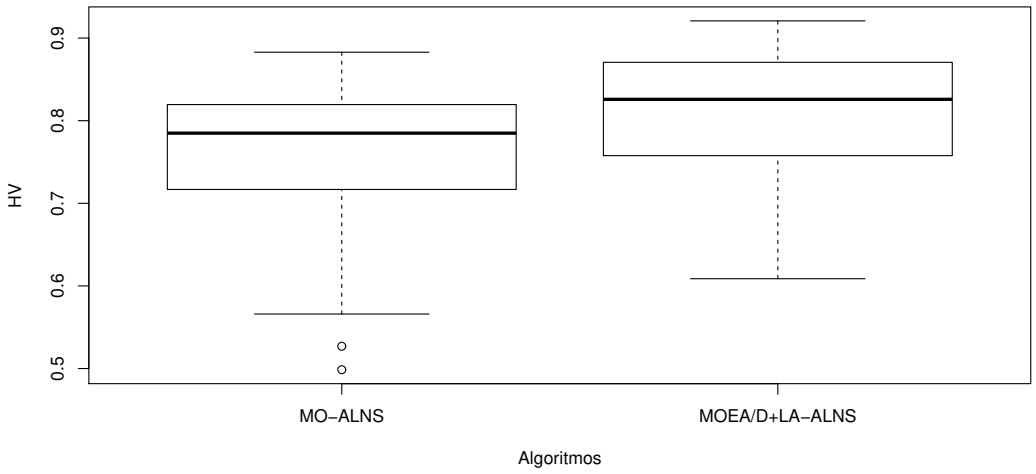


Figura 6.5: Diagrama de caixa dos resultados dos algoritmos MO-ALNS e MOEA/D + LA-ALNS para o indicador HV

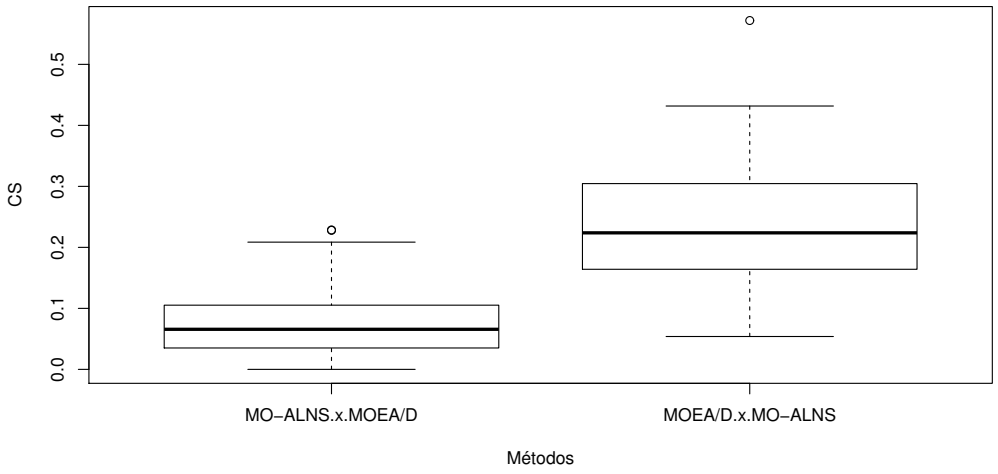


Figura 6.6: Diagrama de caixa dos resultados dos algoritmos MO-ALNS e MOEA/D + LA-ALNS para o indicador CS

entre os resultados do hipervolume.

Uma terceira métrica também é utilizada para medir a qualidade das aproximações da fronteira Pareto dos algoritmos. Esta métrica é a *hierarchical cluster counting* (HCC) (Guimarães et al., 2009). O HCC é capaz de medir tanto a uniformidade quanto a extensão de um conjunto estimado. Neste indicador as soluções não dominadas (ou pontos) do conjunto estimado são sequencialmente agrupadas em *clusters*. Na iteração inicial, o procedimento aglomerativo de *clusters* inicia com cada ponto sendo um *cluster*. Nas próximas iterações, os *clusters* mais próximos são unidos até que todos os dados sejam agrupados em apenas uma classe. O valor do HCC é dado pela integração das distâncias de fusão usadas em cada iteração do processo de aglomeração de *clusters*. O conjunto estimado que tem o maior valor do indicador HCC é a melhor descrição da fronteira Pareto.

Os resultados dos algoritmos para o indicador HCC são apresentados na Tabela 6.6. Estes valores estão agrupados pelas instâncias com o mesmo número de tarefas.

Tabela 6.6: Resultados do indicador HCC para os algoritmos MO-ALNS e MO-ALNS/D.

Conjunto de	MO-ALNS			MOEA/D + LA-ALNS		
instâncias	Min	Max	Média	Min	Max	Média
50	186,923	1778,783	818,824	260,784	2411,797	**1081,269**
100	351,529	3179,293	1185,179	500,012	5161,904	**1864,235**
150	273,688	2885,947	1599,840	621,842	6241,462	**2300,969**
200	420,987	2892,856	1442,437	961,406	4125,787	**3114,022**
250	592,521	5841,016	2590,140	880,586	10853,497	**5800,030**

Os melhores resultados médios da Tabela 6.6 estão destacados em negrito. O algoritmo MOEA/D + LA-ALNS obteve os melhores resultados em todos os casos. Analisando os resultados médios completos, verifica-se que o MOEA/D + LA-ALNS é o melhor em 100% dos casos. Um teste estatístico foi aplicado para verificar se existe diferença estatística entre os resultados médios do indicador HCC. O teste não-paramétrico [4] *Wilcoxon Signed Rank* (Zar, 1999) com 95% de confiança ($threshold = 0,05$) foi usado. O resultado encontrado foi *p-value* igual a $4,72 \times 10^{-7}$. Este valor de p é indicativo forte de que há diferença estatística entre os resultados médios do indicador HCC.

[4] Foi verificado que os dados não seguem uma distribuição normal usando o teste Shapiro-Wilk.

Uma análise gráfica também é realizada para avaliar o comportamento típico dos algoritmos em problemas de grande porte. Para isso, várias instâncias foram selecionadas aleatoriamente e uma destas tem os resultados ilustrados na Figura 6.7.

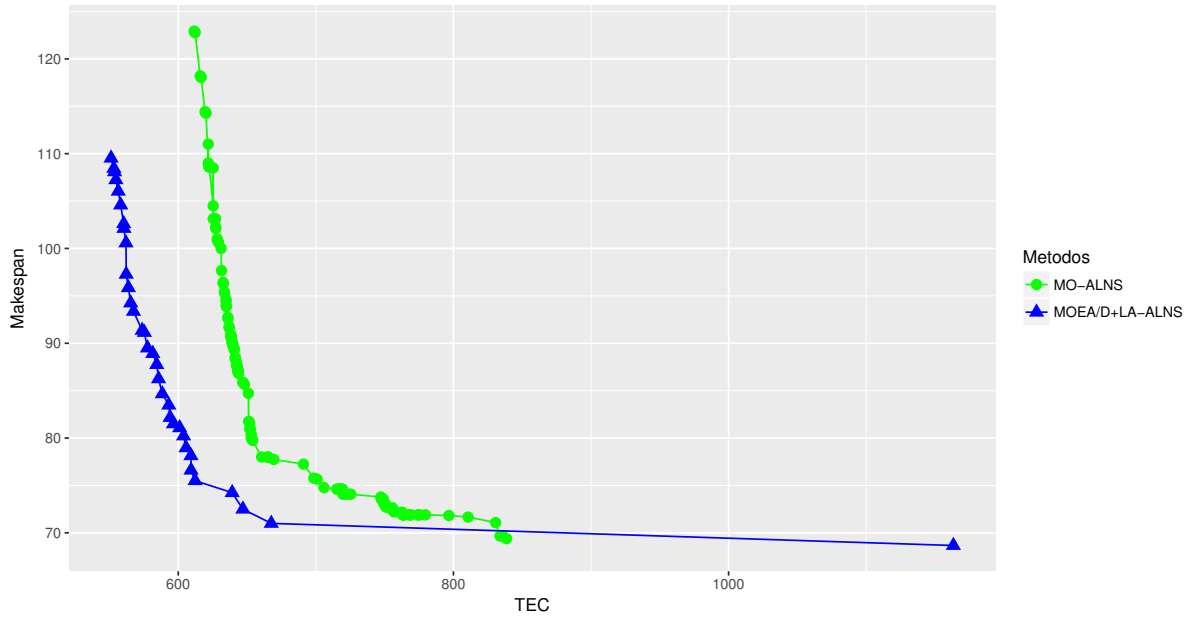


Figura 6.7: Aproximação da fronteira Pareto obtida com algoritmos MO-ALNS e MOEA/D + LA-ALNS para uma instância com 50 tarefas e 10 máquinas.

A Figura 6.7 ajuda a ilustrar como os algoritmos se comportam em instâncias de grande porte. Pode-se observar que a aproximação da fronteira Pareto do MOEA/D + LA-ALNS é muito melhor que a do MO-ALNS.

6.4 Conclusão

Neste capítulo foram tratadas versões de grande porte do problema $R_M|S_{ijk}|(C_{max}, TEC)$, que tem os objetivos de minimizar o consumo total de energia elétrica e o *makespan*. No capítulo anterior este problema também foi tratado, mas considerando apenas instâncias de pequeno e médio porte, o que justifica a utilização de métodos exatos.

Para a resolução de problemas de grande porte foram propostos dois algoritmos multiobjetivo, ambos baseados no algoritmo mono-objetivo LA-ALNS proposto no capítulo 4. Um algoritmo é chamado MO-ALNS, e se trata de uma versão multiobjetivo do LA-ALNS. A principal diferença em relação a versão mono-objetivo é o critério de aceitação,

na versão multiobjetivo usa-se a dominância Pareto. Foi adicionado também uma nova busca local que altera os modos de operação dos processamentos das tarefas. O outro algoritmo multiobjetivo proposto é uma combinação do algoritmo multiobjetivo MOEA/D e o algoritmo mono-objetivo LA-ALNS, sendo chamado MOEA/D + LA-ALNS. Neste algoritmo, o MOEA/D utiliza o LA-ALNS para explorar os subproblemas escalares ao invés dos operadores de reprodução, que são usados em sua versão original.

Nos experimentos computacionais os dois algoritmos multiobjetivo foram validados usando as instâncias de pequeno e médio porte do capítulo anterior, e os resultados ótimos do método ϵ-restrito. Posteriormente, foi proposto um conjunto de instâncias de grande porte e os dois algoritmos foram executados para estes problemas. As métricas hipervolume, *coverage of two sets* e *hierarchical cluster counting* foram usadas para medir a qualidade dos resultados. Observou-se que o algoritmo MOEA/D + LA-ALNS encontrou melhores resultados que o MO-ALNS na maioria dos casos para as três métricas. Posteriormente, foram aplicados testes estatísticos e verificou-se que existe diferença estatística entre os resultados da métrica hipervolume e da *hierarchical cluster counting*. Estes resultados mostram a eficiência do algoritmo MOEA/D + LA-ALNS.

O desempenho inferior do MO-ALNS pode ser explicado pelo fato do algoritmo não ter controle da diversificação na aproximação da fronteira Pareto, já que a cada passo a próxima solução atual é selecionada aleatoriamente. Por outro lado, uma das principais características do MOEA/D é a setorização da busca na aproximação da fronteira Pareto, já que divide o problema multiobjetivo em problemas mono-objetivo com base em agregações usando diferentes vetores de peso.

No geral, acredita-se que o algoritmo MOEA/D + LA-ALNS pode ser aplicado com sucesso a outros problemas de sequenciamento multiobjetivo. Ele combina dois algoritmos muito interessantes, o ALNS que tem sido aplicado com sucesso para resolver diversos problemas de sequenciamento mono-objetivo, e o MOEA/D que é amplamente usado na literatura e tem obtido ótimos resultados na resolução de diversos problemas multiobjetivo e *many-objective*.

Capítulo 7

Conclusões e trabalhos futuros

7.1 Considerações finais

Neste trabalho foram propostas abordagens exatas e heurísticas para a resolução de um problema de sequenciamento em máquinas de grande relevância. O problema específico é o problema de sequenciamento em máquinas paralelas não relacionadas com tempos de preparação dependentes da sequência (ou *unrelated parallel machine scheduling problem with setup times* no idioma inglês). A importância deste problema se deve ao fato de ter grande aplicabilidade prática, já que aparece em indústrias de diversos setores, como as químicas e as têxteis. Este problema também tem grande importância teórica porque pertence à classe de problemas $\mathcal{NP}$-difíceis, tornando desafiante o desenvolvimento de abordagens eficientes para a sua resolução.

Duas versões do problema em questão são tratadas, uma mono-objetivo formalmente definida por $R_M|S_{ijk}|C_{\max}$ e a outra multiobjetivo definida por $R_M|S_{ijk}|(C_{\max}, TEC)$. O $R_M|S_{ijk}|C_{\max}$ é uma versão clássica do problema com o objetivo de minimizar o *makespan*, esta versão é amplamente tratada na literatura. Para este problema existem dois grandes conjuntos de instâncias na literatura, apesar disso, a grande maioria dos trabalhos abordam apenas um deles, porque eles têm características bem distintas. O outro problema tratado é o $R_M|S_{ijk}|(C_{\max}, TEC)$, esta versão é uma nova abordagem proposta neste trabalho. Os objetivos neste problema são minimizar o *makespan* e o consumo total de energia elétrica. Um tema crescente nos últimos anos é o *green scheduling*, este tema trata do desenvolvimento de metodologias sustentáveis em problemas de sequenciamento. A partir deste tema, veio a ideia de tratar o problema $R_M|S_{ijk}|(C_{\max}, TEC)$,

já que abordagens de *green scheduling* ainda são pouco encontradas em problemas de sequenciamento de máquinas, principalmente no problema em questão.

As contribuições desta tese podem ser divididas em três partes, dadas a seguir: *i*) Desenvolvimento de um algoritmo adaptativo que aprende durante o processo de busca e que é capaz de resolver de maneira eficaz o problema $R_M|S_{ijk}|C_{\max}$, usando os dois conjuntos de instâncias da literatura; *ii*) Desenvolvimento de um modelo matemático para o problema $R_M|S_{ijk}|(C_{\max}, TEC)$, no contexto de *green scheduling*, e implementação de técnicas exatas para a sua resolução. *iii*) Desenvolvimento de algoritmos multiobjetivo adaptativos para a resolução de problemas de grande porte do $R_M|S_{ijk}|(C_{\max}, TEC)$.

Na primeira contribuição é proposto o algoritmo LA-ALNS que combina a metaheurística *Adaptive Large Neighborhood Search* (Ropke e Pisinger, 2006) com o uso de aprendizagem em autômatos (ou *Learning Automata* no idioma inglês) para ajustar as probabilidades de uso das heurísticas de remoção e inserção. Uma das principais heurísticas de inserção é baseada no método Húngaro, esta heurística consegue resolver subproblemas do problema principal de maneira ótima em tempo polinomial. Nos experimentos computacionais o algoritmo LA-ALNS foi executado para dois conjuntos de instâncias da literatura, disponíveis em SOA (2011) e Scheduling Research (2005). Os seus resultados foram comparados aos de seis algoritmos recentes da literatura, que são: AIPR (Cota et al., 2014a), ACO (Arnaout et al., 2010), ACOII (Arnaout et al., 2014), *Branch&Check* (Tran et al., 2016), GA2 (Vallada e Ruiz, 2011) e SA (Santos et al., 2016). A comparação dos resultados foi realizada de maneira separada porque grande parte dos algoritmos haviam sido executados para apenas um dos conjuntos de instâncias. O algoritmo LA-ALNS encontrou melhores resultados na maiorias do casos que cinco algoritmos da literatura, nos experimentos realizados. Em uma das comparações, o algoritmo LA-ALNS obteve piores resultados que o algoritmo SA. Mas este algoritmo teve seus parâmetros calibrados para um dos conjuntos de instâncias, o que pode ter levado a melhores resultados. Porém, esta característica não garante bons resultados em outros conjuntos de instâncias com características distintas e em instâncias reais. Dada a eficiência do LA-ALNS nos dois conjuntos de instâncias, verifica-se que o algoritmo é apropriado para resolver diferentes variações do problema tratado e tem grande aplicabilidade prática.

Na segunda contribuição é construído, implementado e analisado um modelo de programação linear inteira mista para o problema $R_M|S_{ijk}|(C_{\max}, TEC)$. Para implementar o modelo matemático é utilizado o método clássico ϵ-restrito e o método *Smart Pool* (Coelho et al., 2016a). O *Smart Pool* foi aperfeiçoado neste trabalho com o uso do

método de Scheffé (Scheffé, 1958) para a geração dos vetores de peso. Nos experimentos computacionais foi proposto um conjunto de instâncias de pequeno e médio porte para o problema. Inicialmente, foi analisado o *trade-off* entre os objetivos *makespan* e consumo total de energia elétrica, constatando-se que os objetivos são conflitantes e que o objetivo consumo total de energia elétrica tem grande importância. Posteriormente, foram comparados os resultados dos métodos ϵ-restrito e *Smart Pool*. O método *Smart Pool* encontrou os melhores resultados na maioria dos casos para as duas métricas de qualidade avaliadas, mostrando ser um método rápido e eficiente para encontrar boas aproximações da fronteira Pareto em problemas de pequeno e médio porte.

Na terceira contribuição foram propostos dois algoritmos multiobjetivo para a resolução de versões de grande porte do problema $R_M|S_{ijk}|(C_{\max}, TEC)$. O uso de técnicas heurísticas nestes casos se faz necessário, dada a incapacidade dos métodos exatos em resolver problemas de grande porte em um tempo restrito. O primeiro algoritmo proposto é chamado MO-ALNS e se trata de uma versão multiobjetivo do LA-ALNS, proposto anteriormente para resolver o $R_M|S_{ijk}|C_{\max}$. O segundo algoritmo é a combinação do algoritmo multiobjetivo MOEA/D (Zhang e Li, 2007) e o algoritmo mono-objetivo LA-ALNS, este algoritmo é chamado MOEA/D + LA-ALNS. No MOEA/D original os subproblemas gerados por meio de métodos de decomposição e vetores de pesos são explorados por meio de operadores de reprodução. Já no algoritmo MOEA/D + LA-ALNS, proposto neste trabalho, é utilizado o algoritmo mono-objetivo LA-ALNS para explorar os subproblemas do MOEA/D. Nos experimentos computacionais os dois algoritmos propostos foram validados usando o conjunto de instâncias de pequeno e médio porte, e os resultados ótimos do método ϵ-restrito. Posteriormente, foi criado um conjunto de instâncias de grande porte e os dois algoritmos foram executados para estes problemas. Para avaliar os algoritmos foram utilizadas três métricas de qualidade e o algoritmo MOEA/D + LA-ALNS encontrou os melhores resultados em todas as métricas. O algoritmo MO-ALNS não tem controle da diversificação na aproximação da fronteira Pareto, este fato pode explicar o pior desempenho. Já o algoritmo MOEA/D tem esta característica como uma das principais qualidades, uma vez que faz a setorização da busca na aproximação da fronteira Pareto.

7.2 Trabalhos futuros

Como trabalhos futuros são propostas as seguintes atividades:

- Aplicar o algoritmo mono-objetivo LA-ALNS a outros problemas de sequenciamento de máquinas e roteamento de veículos;
- Analisar novas características no contexto de *green scheduling* para o problema tratado e outros problemas relacionados, como variações no preço de energia por tempo de uso e emissões de carbono;
- Aplicar o método *Smart Pool* e o algoritmo multiobjetivo MOEA/D + LA-ALNS a outros problemas de sequenciamento multiobjetivo.

Apêndice A

Trabalhos gerados

Seguem abaixo os artigos gerados ao longo do desenvolvimento desta teste.

Artigos publicados:

- Cota, L. P., Coelho, V. N., Guimarães, F. G., Souza, M. J. F.: 2018, Bi-criteria formulation for green scheduling with unrelated parallel machines with sequence dependent setup times. International Transactions in Operations Research. doi: 10.1111/itor.12566.
- Cota, L. P., Guimarães, F. G., Oliveira, F. B. and Souza, M. J. F.: 2017a, An adaptive large neighborhood search with learning automata for the unrelated parallel machine scheduling problem, Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC 2017), Donóstia - San Sebastian, pp. 185-192. doi: 10.1109/CEC.2017.7969312.
- Cota, L. P., Guimarães, F. G., Oliveira, F. B. and Souza, M. J. F.: 2017b, Um método baseado em adaptive large neighborhood search para resolução de um problema de sequenciamento em máquinas paralelas, Proceedings of the XLIX Simpósio Brasileiro de Pesquisa Operacional (SBPO 2017), Blumenau/SC, pp. 1352-1363.
- Coelho, V. N., Coelho, I. M., Souza, M. J. F., Oliveira, T. A., Cota, L. P., Haddad, M. N., Mladenovic, N., Silva, R. C. P., Guimarães, F. G.: 2016, Hybrid self-adaptive evolution strategies guided by neighborhood structures for combinatorial optimization problems. Evolutionary Computation, v. 24, p. 637-666. doi: 10.1162/EVCO_a_00187.

- Rezende, J. C. V., Souza, M. J. F., Cota, L. P.: 2016, Um novo algoritmo híbrido para a resolução de problemas binários, Proceedings of the XLVIII Simpósio Brasileiro de Pesquisa Operacional (SBPO 2016), Vitória/ES, pp. 2515-2526.

Artigos submetidos que estão em análise:

- Cota, L. P., Guimarães, F. G., Oliveira, F. B., Souza, M. J. F., Siarry, P.: Integrating Q-Learning in Adaptive Large Neighborhood Search Metaheuristic for the Parallel Machine Scheduling Problem. Computers & Operations Research.
- Cota, L. P., Guimarães, F. G., Meneghini, I. R., Oliveira, F. B., Souza, M. J. F., Siarry, P.: An adaptive multi-objective algorithm based on decomposition and large neighborhood search for a machine green scheduling problem. Expert Systems with Applications.

Referências Bibliográficas

Ahilan, C.; Kumanan, S.; Sivakumaran, N. e Dhas, J. E. R. (2013). Modeling and prediction of machining quality in cnc turning process using intelligent hybrid decision making tools. *Applied Soft Computing*, v. 13, n. 3, p. 1543 – 1551. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2012.03.071. Hybrid evolutionary systems for manufacturing processes.

Aiex, R. M.; Resende, M. G. C. e Ribeiro, C. C. Sep(2007). Tttplots: a perl program to create time-to-target plots. *Optimization Letters*, v. 1, n. 4, p. 355–366. ISSN 1862-4480. doi: 10.1007/s11590-006-0031-4.

Al-Salem, A. (2004). Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, v. 17, n. 1, p. 177–187.

Alipour, M. M. (2012)a. A learning automata based algorithm for solving capacitated vehicle routing problem. *International Journal of Computer Science Issues*, v. 9, p. 138–145.

Alipour, Mir Mohammad. May(2012)b. Article: A learning automata based algorithm for solving traveling salesman problem improved by frequency-based pruning. *International Journal of Computer Applications*, v. 46, n. 17, p. 7–13. doi: 10.5120/7007-9328. Full text available.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, v. 246, n. 2, p. 345 – 378. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2015.04.004.

Arenales, M.; Armentano, F.; Morabito, R. e Yanasse, H. (2015). *Pesquisa Operacional*. Campus, 2nd edição. ISBN 978-85-352-5193-7.

Arnaout, J. P.; Musa, R. e Rabadi, G. (2014). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines - part ii: enhancements and experimentations. *Journal of Intelligent Manufacturing*, v. 25, p. 43–53. ISSN 1572-8145. doi: 10.1007/s10845-012-0672-3.

Arnaout, J.P.; Rabadi, G. e Musa, R. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, v. 21, n. 6, p. 693–701. ISSN 1572-8145. doi: 10.1007/s10845-009-0246-1.

Arroyo, J. E. C. e Armentano, V. A. (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.

Avalos-Rosales, O.; Alvarez, A. e Angel-Bello, F. (2013). A reformulation for the problem scheduling unrelated parallel machines with sequence and machine dependent setup times. *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, p. 278–282, Rome, Italy.

Avalos-Rosales, O.; Angel-Bello, F. e Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, v. 76, n. 9, p. 1705–1718. ISSN 1433-3015. doi: 10.1007/s00170-014-6390-6.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling.* John Wiley & Sons. ISBN 0471045551.

Baker, K. R. e Trietsch, D. (2009). *Principles of Sequencing and Scheduling.* John Wiley & Sons, Inc. ISBN 9780470451793. doi: 10.1002/9780470451793.

Bakkehaug, R.; Rakke, J. G.; Fagerholt, K. e Laporte, G. (2016). An adaptive large neighborhood search heuristic for fleet deployment problems with voyage separation requirements. *Transportation Research Part C: Emerging Technologies*, v. 70, n. Supplement C, p. 129 – 141. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2015.06.019.

Bampis, E.; Letsios, D. e Lucarelli, G. (2015). Green scheduling, flows and matchings. *Theoretical Computer Science*, v. 579, p. 126 – 136. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2015.02.020.

Beezão, Andreza Cristina; Cordeau, Jean-François; Laporte, Gilbert e Yanasse, Horacio Hideki. (2017). Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, v. 257, n. 3, p. 834–844. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2016.08.008.

Caniyilmaz, E.; Benli, B. e Ilkay, M. S. Apr(2015). An artificial bee colony algorithm approach for unrelated parallel machine scheduling with processing set restrictions, job sequence-dependent setup times, and due date. *The International Journal of Advanced Manufacturing Technology*, v. 77, n. 9, p. 2105–2115. ISSN 1433-3015. doi: 10.1007/s00170-014-6614-9.

Chang, P.C. e Chen, S.H. (2011). Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times. *Applied Soft*

Computing, v. 11, n. 1, p. 1263–1274. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2010.03.003.

Chen, J. Oct(2009). Scheduling on unrelated parallel machines with sequence- and machine-dependent setup times and due-date constraints. *The International Journal of Advanced Manufacturing Technology*, v. 44, n. 11, p. 1204–1212. ISSN 1433-3015. doi: 10.1007/s00170-008-1917-3.

Cheng, J.; Chu, F.; Liu, M.; Wu, P. e Xia, W. (2017). Bi-criteria single-machine batch scheduling with machine on/off switching under time-of-use tariffs. *Computers & Industrial Engineering*, v. . ISSN 0360-8352. doi: http://dx.doi.org/10.1016/j.cie.2017.04.026.

Coelho, I. M.; Munhoz, P. L. A.; Haddad, M. N.; Coelho, V. N.; Silva, M. M.; Souza, M.J. F. e Ochi, L. S. (2011). A computational framework for combinatorial optimization problems. *VII ALIO/EURO Workshop on Applied Combinatorial Optimization*, p. 51–54, Porto.

Coelho, V. N.; Coelho, I. M.; Coelho, B. N.; Cohen, M. W.; Reis, A. J. R.; Silva, S. M.; Souza, M. J. F.; Fleming, P. J. e Guimarães, F. G. (2016)a. Multi-objective energy storage power dispatching using plug-in vehicles in a smart-microgrid. *Renewable Energy*, v. 89, n. Supplement C, p. 730 – 742. ISSN 0960-1481. doi: http://dx.doi.org/10.1016/j.renene.2015.11.084.

Coelho, V. N.; Coelho, I. M.; Coelho, B. N.; deOliveira, G. C.; Barbosa, A. C.; Pereira, L.; deFreitas, A.; Santos, H. G.; Ochi, L. S. e Guimarães, F. G. (2017)a. A communitarian microgrid storage planning system inside the scope of a smart city. *Applied Energy*, v. 201, n. Supplement C, p. 371 – 381. ISSN 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2016.12.043.

Coelho, V. N.; Coelho, I. M.; Souza, M. J. F.; Oliveira, T. A.; Cota, L. P.; Haddad, M. N.; Mladenovic, N.; Silva, R. C. P. e Guimarães, F. G. Dec(2016)b. Hybrid self-adaptive evolution strategies guided by neighborhood structures for combinatorial optimization problems. *Evolutionary Computation*, v. 24, n. 4, p. 637–666. ISSN 1063-6560. doi: 10.1162/EVCO_a_00187.

Coelho, V. N.; Cohen, M. W.; Coelho, I. M.; Liu, N. e Guimarães, F. G. (2017)b. Multi-agent systems applied for energy systems integration: State-of-the-art applications and trends in microgrids. *Applied Energy*, v. 187, p. 820 – 832. ISSN 0306-2619. doi: http://dx.doi.org/10.1016/j.apenergy.2016.10.056.

Coello, C. A.; Lamont, G. B. e Veldhuizen, D. A. V. (2006). *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 0387332545.

Coffman, E. G. (1976). *Computer and Job-shop Scheduling Theory*. John Wiley & Sons, Inc.

Conway, R. W. e Maxwell, W. L. (1967). *Theory of Scheduling*. Addison-Wesley.

Cota, L. P.; Guimarães, F. G.; Oliveira, F. B. e Souza, M. J. F. (2017)a. An adaptive large neighborhood search with learning automata for the unrelated parallel machine scheduling problem. *Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC 2017)*, p. 185–192, Donóstia - San Sebastian. doi: 10.1109/CEC.2017.7969312.

Cota, L. P.; Guimarães, F. G.; Oliveira, F. B. e Souza, M. J. F. (2017)b. Um método baseado em adaptive large neighborhood search para resolução de um problema de sequenciamento em máquinas paralelas. *Proceedings of the XLIX Simpósio Brasileiro de Pesquisa Operacional (SBPO 2017)*, Blumenau/SC.

Cota, L. P.; Haddad, M. N.; Souza, M. J. F. e Coelho, V. N. (2014)a. AIRP: A heuristic algorithm for solving the unrelated parallel machine sheduling problem. *Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC 2014)*, p. 1855–1862, Beijing. doi: 10.1109/CEC.2014.6900245.

Cota, L. P.; Haddad, M. N.; Souza, M. J. F. e Martins, A. X. (2014)b. A heuristic algorithm for solving the unrelated parallel machine scheduling problem with sequence dependent setup time (in portugueses). *Proceedings of the 35th Brazilian Congress of Applied and Computational Mathematics*, Natal, Brazil. doi: 10.5540/03.2015.003.01.0412. Available at http://proceedings.sbmac.org.br/sbmac/article/view/724/730.

Dayarian, I.; Crainic, T. G.; Gendreau, M. e Rei, W. (2006). An adaptive large-neighborhood search heuristic for a multi-period vehicle routing problem. *Transportation Research Part E Logistics and Transportation*. doi: 10.1016/j.tre.2016.09.004.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. ISBN 978-0-471-87339-6.

Deb, K. (2009). *Multi-Objective Optimization Using Evolutionary Algorithms, Wileyinterscience series in systems and optimization*. Wiley. ISBN 978-0-470-74361-4.

Deb, K.; Pratap, A.; Agarwal, S. e Meyarivan, T. Apr(2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, v. 6, n. 2, p. 182–197. ISSN 1089-778X. doi: 10.1109/4235.996017.

Deb, Kalyanmoy e Jain, Himanshu. aug(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. ISSN 1089-778X. doi: 10.1109/tevc.2013.2281535.

Ding, J.; Song, S. e Wu, C. (2016). Carbon-efficient scheduling of flow shops by multi-objective optimization. *European Journal of Operational Research*, v. 248, p. 758–771. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.05.019.

Dooren, D. V. D.; Sys, T.; Toffolo, T. A. M.; Wauters, T. e Berghe, G. V. Vanden. (2017). Multi-machine energy-aware scheduling. *EURO Journal on Computational Optimization*, v. 5, n. 1, p. 285–307. ISSN 2192-4414. doi: 10.1007/s13675-016-0072-0.

Dorigo, M.; Maniezzo, V. e Colorni, A. Feb(1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, v. 26, n. 1, p. 29–41. ISSN 1083-4419. doi: 10.1109/3477.484436.

Feo, T. e Resende, M. (1995). Greedy randomized search procedures. *Journal of Global Optimization*, v. 6, n. 2, p. 109–133. ISSN 1573-2916. doi: 10.1007/BF01096763.

Fleszar, K.; Charalambous, C. e Hindi, K.S. (2011). A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-011-0522-8.

Fonseca, M. C. e Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. *Proceedings of the 5th International Conference on Genetic Algorithms*, p. 416–423, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. ISBN 1-55860-299-2.

Freitas, A. R. R. (2013). Redução de dimensionalidade em problemas com muitos objetivos: Uma aplicação em composição algorítmica. Tese de doutorado, Programa de Pós-Graduação em Engenharia Elétrica, UFMG, Belo Horizonte, Brasil.

French, S. (1982). *Sequencing and Scheduling*. Ellis Horwood, Ltd.

Garey, M. R. e Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-Completeness*, volume 174. ISBN 0716710455.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, v. 13, n. 5, p. 533 – 549. ISSN 0305-0548. doi: https://doi.org/10.1016/0305-0548(86)90048-1. Applications of Integer Programming.

Glover, F. W. e Kochenberger, G. A. (2003). *Metaheuristics*. Springer US, Verlag, USA, 1st edição. ISBN 0884-8289. doi: 10.1007/b101874.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edição. ISBN 0201157675.

Goldberger, J. e Tassa, T. (2008). A hierarchical clustering algorithm based on the hungarian method. *Pattern Recognition Letters*, v. 29, n. 11, p. 1632 – 1638. ISSN 0167-8655. doi: http://doi.org/10.1016/j.patrec.2008.04.003.

Graham, R.L.; Lawler, E.L.; Lenstra, J.K. e Kan, A.H.G.R. (1979). v. 5, n. Supplement C, p. 287 – 326. ISSN 0167-5060. doi: https://doi.org/10.1016/S0167-5060(08)70356-X.

Grimault, A.; Bostel, N. e Lehuédé, F. (2017). An adaptive large neighborhood search for the full truckload pickup and delivery problem with resource synchronization. *Computers & Operations Research*, v. 88, n. Supplement C, p. 1 – 14. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2017.06.012.

Guimarães, 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.

Haddad, M. N.; Cota, L. P.; Souza, M. J. F. e Maculan, N. (2014). AIV: A heuristic algorithm based on iterated local search and variable neighborhood descent for solving the unrelated parallel machine scheduling problem with setup times. *Proceedings of the 16th International Conference on Enterprise Information Systems (ICEIS 2014)*, p. 376–383, Lisbon, Portugal. DOI: 10.5220/0004884603760383.

Haddad, M. N.; Cota, L. P.; Souza, M. J. F. e Maculan, N. (2015). Solving the unrelated parallel machine scheduling problem with setup times by efficient algorithms based on iterated local search. *Lecture Notes in Enterprise Information Systems*, v. 227, p. 131–148. doi: DOI:10.1007/978-3-319-22348-38.

Hames, Y. Y.; Lasdon, L. S. e Wismer, D. A. July(1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, v. SMC-1, n. 3, p. 296–297. ISSN 0018-9472. doi: 10.1109/TSMC.1971.4308298.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. *Proceedings of the Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy.

Hansen, P.; Mladenovic, N. e Pérez, J. A. M. (2008). Variable neighborhood search: methods and applications. *4OR*, v. 6, p. 319–360. ISSN 1614-2411. doi: 10.1007/s10288-008-0089-1.

Helal, M.; Rabadi, G. e Al-Salem, A. (2006). A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, v. 3, n. 3, p. 182–192.

Iris, C.; Pacino, D. e Ropke, S. (2017). Improved formulations and an adaptive large neighborhood search heuristic for the integrated berth allocation and quay crane assignment problem. *Transportation Research Part E: Logistics and Transportation Review*, v. 105, n. Supplement C, p. 123 – 147. ISSN 1366-5545. doi: https://doi.org/10.1016/j.tre.2017.06.013.

Jungnickel, Dieter. (2008). *Graph Networks and Algorithms*, volume 5 of *Algorithms and Computation in Mathematics*. Springer-Verlag Berlin Heidelberg, New York, 3 edição. doi: 10.1007/978-3-540-72780-4.

Karp, Richard M. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, v. 40, n. 4, p. 85–103. doi: 10.1007/978-1-4684-2001-2_9.

Kim, D. W.; Kim, K. H.; Jang, W. e Frank Chen, F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, v. 18, p. 223–231. doi: 10.1016/S0736-5845(02)00013-3.

Kim, D. W.; Na, D. G. e Frank Chen, F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer-Integrated Manufacturing*, v. 19, p. 173–181. ISSN 0736-5845. doi: https://doi.org/10.1016/S0736-5845(02)00077-7.

Kirkpatrick, S.; Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680. ISSN 0036-8075. doi: 10.1126/science.220.4598.671.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, v. 2, p. 83–97.

Kuhn, H. W. (2010). *The Hungarian Method for the Assignment Problem*, Capítulo Chapter 2, p. 29–47. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-68279-0. doi: 10.1007/978-3-540-68279-0_2.

Lee, J.; Yu, J. e Lee, D. Dec(2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, v. 69, n. 9, p. 2081–2089. ISSN 1433-3015. doi: 10.1007/s00170-013-5192-6.

Lehmann, Ralph. (1984). Contributions to the hungarian method. *Mathematische Operationsforschung und Statistik. Series Optimization*, v. 15, n. 1, p. 91–97. doi: 10.1080/02331938408842911.

Lin, S. W. e Ying, K. C. (2014). ABC-based manufacturing scheduling for unrelated parallel machines with machine-dependent and job sequence-dependent setup times. *Computers & Operations Research*, v. 51, p. 172–181. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2014.05.013.

Lin, Y. e Hsieh, F. (2014). Unrelated parallel machine scheduling with setup times and ready times. *International Journal of Production Research*, v. 52, n. 4, p. 1200–1214. doi: 10.1080/00207543.2013.848305.

Liu, X.; Laporte, G.; Chen, Y. e He, R. (2017). An adaptive large neighborhood search metaheuristic for agile satellite scheduling with time-dependent transition time. *Computers & Operations Research*, v. 86, n. Supplement C, p. 41 – 53. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2017.04.006.

Liu, Y.; Dong, H.; Lohse, N. e Petrovic, S. (2016). A multi-objective genetic algorithm for optimisation of energy consumption and shop floor production performance. *International Journal of Production Economics*, v. 179, p. 259 – 272. ISSN 0925-5273. doi: http://dx.doi.org/10.1016/j.ijpe.2016.06.019.

Logendran, R.; McDonell, B. e Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations research*, v. 34, n. 11, p. 3420–3438. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2006.02.006.

López-Ibáñez, Manuel; Dubois-Lacoste, Jérémie; Stützle, Thomas e Birattari, Mauro. (2011). The irace package, iterated race for automatic algorithm configuration. Relatório Técnico TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium. URL `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`.

Lourenço, H. R.; Martin, O. e Stützle, T. (2003). Iterated local search. Glover, F. e Kochenberger, G., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, p. 321–353. Kluwer Academic Publishers, Norwell, MA.

Luenberger, D. G. e Ye, Y. (2008). *Linear and Nonlinear Programming*. Springer US, Verlag, USA, 3st edição. ISBN 978-0-387-74502-2. doi: 10.1007/978-0-387-74503-9.

Lust, T. e Teghem, J. (2010). Two-phase pareto local search for the biobjective traveling. *Journal of Heuristics*, v. 16, p. 475–510.

Majidi, S.; Hosseini-Motlagh, S. e Ignatius, J. Mar(2017). Adaptive large neighborhood search heuristic for pollution-routing problem with simultaneous pickup and delivery. *Soft Computing*. ISSN 1433-7479. doi: 10.1007/s00500-017-2535-5.

Mansouri, A. S. e Aktas, E. (2016). Minimizing energy consumption and makespan in a two-machine flowshop scheduling problem. *Journal of the Operational Research Society*, v. 67, n. 11, p. 1382–1394. ISSN 1476-9360. doi: 10.1057/jors.2016.4.

Mansouri, S. A.; Aktas, E. e Besikci, U. (2016)a. Green scheduling of a two-machine flow shop: Trade-off between makespan and energy consumption. *European Journal of Operation Research*, v. 248, p. 772–788. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.08.064.

Mansouri, S. A.; Aktas, E. e Besikci, U. (2016)b. Minimizing energy consumption and makespan in a two-machine flowshop scheduling problem. *Journal of the Operational Research Society*. ISSN 1476-9360. doi: 10.1057/jors.2016.4.

Mattos Ribeiro, Glaydston e Laporte, Gilbert. mar(2012). An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Comput. Oper. Res.*, v. 39, n. 3, p. 728–735. ISSN 0305-0548. doi: 10.1016/j.cor.2011.05.005.

Meneghini, I. R. e Guimarães, F. G. June(2017). Evolutionary method for weight vector generation in multi-objective evolutionary algorithms based on decomposition and aggregation. *2017 IEEE Congress on Evolutionary Computation (CEC)*, p. 1900–1907, June(2017). doi: 10.1109/CEC.2017.7969533.

Montgomery, D. (2007). *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, 5th edição.

Morton, T. E. e Pentico, D. W. (1993). *Heuristic Scheduling Systems*. John Wiley & Sons, Inc.

Muth, J. F. e Thompson, G. L. (1963). *Industrial scheduling*. Englewood Cliffs, N.J. : Prentice-Hall.

Narendra, K. S. e Thathachar, K. S. (1989). *Learning Automata: An Introduction*. Prentice-Hall, New York.

Narendra, K. S. e Thathachar, M. A. (2012). *Learning automata: an introduction*. Courier Corporation.

Narendra, K. S. e Thathachar, M. A. L. July(1974). Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, v. SMC-4, n. 4, p. 323–334. ISSN 0018-9472. doi: 10.1109/TSMC.1974.5408453.

Papadimitriou, Christos H. e Steiglitz, Kenneth. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., N.J. ISBN 0-13-152462-3.

Pareto, V. (1896). *Cours d´économie politique*. Institut Coppet.

PassMark,. Cpu benchmarks, (2017). accessed 1 February 2017.

Paula, M. C.; Mateus, G. R e Ravetti, M. G. May(2010). A non-delayed relax-and-cut algorithm for scheduling problems with parallel machines, due dates and sequence-dependent setup times. *Comput. Oper. Res.*, v. 37, n. 5, p. 938–949. ISSN 0305-0548. doi: 10.1016/j.cor.2009.07.006.

Pereira Lopes, M. J. e de Carvalho, J. M. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*, v. 176, p. 1508–1527. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2005.11.001.

Pinedo, M. L. (2001). *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 2rd edição. ISBN 0130281387, 9780130281388.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edição. ISBN 0387789340, 9780387789347.

Rabadi, G.; Moraga, R. J. e Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, v. 17, n. 1, p. 85–97. ISSN 1572-8145. doi: 10.1007/s10845-005-5514-0.

Randall, P. A. e Kurz, M. E. (2007). Effectiveness of Adaptive Crossover Procedures for a Genetic Algorithm to Schedule Unrelated Parallel Machines with Setups. *International Journal of Operational Research*, v. 4, p. 1–10.

Rifai, A. P.; Nguyen, H. e Dawal, S. Z. M. (2016). Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing*, v. 40, n. Supplement C, p. 42 – 57. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2015.11.034.

Rocha, L. P.; Ravetti, M. G.; Matheus, G. R. e Pardalos, P. M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, v. 35, p. 1250–1264. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2006.07.015.

Ropke, S. e Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, v. 40, n. 4, p. 455–472. doi: 10.1287/trsc.1050.0135.

Ruiz, R. e Stützle, T. (2005). An iterated greedy algorithm for the flowshop problem with sequence dependent setup times. *Proceedings of the 6th Metaheuristics International Conference*, p. 817–826, Vienna, Austria.

Santos, Haroldo G.; Toffolo, Túlio A.M.; Silva, Cristiano L.T.F. e Vanden Berghe, Greet. (2016). Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem. *International Transactions in Operational Research*, p. n/a–n/a. ISSN 1475-3995. doi: 10.1111/itor.12316.

Sauer, Ildo L.; Tatizawa, Hedio; Salotti, Francisco A.M. e Mercedes, Sonia S. (2015). A comparative assessment of brazilian electric motors performance with minimum efficiency standards. *Renewable and Sustainable Energy Reviews*, v. 41, n. Supplement C, p. 308 – 318. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2014.08.053.

Scheduling Research,. Scheduling research virtual center, (2005). A web site that includes benchmark problem data sets and solutions for scheduling problems. Disponível em `http://www.schedulingresearch.com`. Acesso em 01 de julho de 2013.

Scheffé, H. (1958). Experiments with mixtures. *Journal of the Royal Statistical Society*, v. 20, n. 2, p. 344–360.

Sen, P. C. (2013). *Principles of Electric Machines and Power Electronics*. Wiley, 3rd edição. ISBN 9781118078877.

Shapiro, S. S. e Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, v. 52, p. 591–611.

Shaw, P. (1998). *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems*, p. 417–431. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-49481-2.

Siarry, P. (2016). *Metaheuristics*. Springer International Publishing, Switzerland, 1st edição. ISBN 978-3-319-45401-6. doi: 10.1007/978-3-319-45403-0.

SOA,, (2011). Sistemas de Optimizacion Aplicada. A web site that includes benchmark problem data sets and solutions for scheduling problems. Disponível em http://soa.iti.es/problem-instances.

Souza, M. J. F.; Coelho, I.M.; Ribas, S.; Santos, H.G. e Merschmann, L.H.C. (2010). A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research*, v. 207, n. 2, p. 1041–1051. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2010.05.031.

Storn, R. e Price, K. Dec(1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, v. 11, n. 4, p. 341–359. ISSN 1573-2916. doi: 10.1023/A:1008202821328.

Toro, E. M.; Franco, J. F.; Echeverri, M. G. e Guimarães, F. G. (2017). A multi-objective model for the green capacitated location-routing problem considering environmental impact. *Computers & Industrial Engineering*, v. 110, p. 114 – 125. ISSN 0360-8352. doi: http://dx.doi.org/10.1016/j.cie.2017.05.013.

Tran, Tony T.; Araujo, Arthur e Beck, J. Christopher. (2016). Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing*, v. 28, n. 1, p. 83–95. doi: 10.1287/ijoc.2015.0666.

Trivedi, A.; Srinivasan, D.; Sanyal, K. e Ghosh, A. June(2017). A survey of multiobjective evolutionary algorithms based on decomposition. *IEEE Transactions on Evolutionary Computation*, v. 21, n. 3, p. 440–462. ISSN 1089-778X. doi: 10.1109/TEVC.2016.2608507.

Vadlamani, S. e Hosseini, S. (2014). A novel heuristic approach for solving aircraft landing problem with single runway. *Journal of Air Transport Management*, v. 40, n. Supplement C, p. 144 – 148. ISSN 0969-6997. doi: https://doi.org/10.1016/j.jairtraman.2014.06.009.

Vafashoar, R. e Meybodi, M. R. (2016). Multi swarm bare bones particle swarm optimization with distribution adaption. *Applied Soft Computing*, v. 47, p. 534 – 552. ISSN 1568-4946. doi: http://dx.doi.org/10.1016/j.asoc.2016.06.028.

Vallada, E. e Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, v. 211, n. 3, p. 612–622. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2011.01.011.

Wang, S.; Liu, M.; Chu, F. e Chu, C. (2016). Bi-objective optimization of a single machine batch scheduling problem with energy cost consideration. *Journal of Cleaner Production*, v. 137, p. 1205 – 1215. ISSN 0959-6526. doi: http://dx.doi.org/10.1016/j.jclepro.2016.07.206.

Wen, M.; Linde, E.; Ropke, S.; Mirchandani, P. e Larsen, A. (2016). An adaptive large neighborhood search heuristic for the electric vehicle scheduling problem. *Computers & Operations Research*, v. 76, n. Supplement C, p. 73 – 83. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2016.06.013.

Weng, M. X.; Lu, J. e Ren, H. (2001). Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. *International Journal of Production Economics*, v. 70, n. 3, p. 215–226. ISSN 0925-5273. doi: https://doi.org/10.1016/S0925-5273(00)00066-9.

Wildi, T. (2013). *Electrical Machines, Drives and Power Systems*. Pearson, 6th edição. ISBN 9781292024585.

Winston, W. L. (2004). *Operations Research: Applications and Algorithms*. thomson brooks/cole, 4th edição. ISBN 978-0534380588.

Ying, Kuo-Ching; Lee, Zne-Jung e Lin, Shih-Wei. (2010). Makespan minimisation for scheduling unrelated parallel machines with setup times. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-010-0483-3.

Yu, C.; Zhang, D. e Lau, H. Y. K. (2017). An adaptive large neighborhood search heuristic for solving a robust gate assignment problem. *Expert Systems with Applications*, v. 84, n. Supplement C, p. 143 – 154. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2017.04.050.

Zar, J. H. (1999). *Biostatistical analysis*. Prentice Hall, Upper Saddle River, NJ, fourth edição.

Zeidi, J. R. e MohammadHosseini, S. Dec(2015). Scheduling unrelated parallel machines with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, v. 81, n. 9, p. 1487–1496. ISSN 1433-3015. doi: 10.1007/s00170-015-7215-y.

Zhang, H.; Zhao, F.; Fang, K. e Sutherland, J. W. (2014). Energy-conscious flow shop scheduling under time-of-use electricity tariffs. *CIRP Annals*, v. 63, p. 37–40. ISSN 0007-8506. doi: https://doi.org/10.1016/j.cirp.2014.03.011.

Zhang, Q. e Li, H. Dec(2007). Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, v. 11, n. 6, p. 712–731. ISSN 1089-778X. doi: 10.1109/TEVC.2007.892759.

Zhu, X. e Wilhelm, W. E. (2006). Scheduling and lot sizing with sequence-dependent setup: A literature review. *IIE Transactions*, v. 38, p. 987–1007. doi: 10.1080/07408170600559706.

Zitzler, E.; Laumanns, M. e Thiele, L. (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, E. e Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, v. 3, p. 257–271. ISSN 1089-778X. doi: 10.1109/4235.797969.