MICHELE BERNARDINO FIDELIS

# META-HEURÍSTICAS PARA O PROBLEMA DE SEQUENCIAMENTO DE LOTES DE TAREFAS EM MÁQUINAS PARALELAS

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

VIÇOSA
MINAS GERAIS – BRASIL
2017

**Ficha catalográfica preparada pela Biblioteca Central da Universidade Federal de Viçosa - Câmpus Viçosa**

T
F451m
2017

Fidelis, Michele Bernardino, 1991-
Meta-heurísticas para o problema de sequenciamento de lotes de tarefas em máquinas paralelas / Michele Bernardino Fidelis. – Viçosa, MG, 2017.
xi, 66f. : il. (algumas color.) ; 29 cm.

Orientador: José Elias Claudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Referências bibliográficas: f. 62-66.

1. Heurística. 2. Algorítimos. 3. Máquinas. 4. Solução de problemas. 5. Otimização combinatória. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-Graduação em Ciência da Computação. II. Título.

CDD 22. ed. 518.1

MICHELE BERNARDINO FIDELIS

# META-HEURÍSTICAS PARA O PROBLEMA DE SEQUENCIAMENTO DE LOTES DE TAREFAS EM MÁQUINAS PARALELAS

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

APROVADA: 14 de dezembro de 2017.

Mauro Nacif Rocha

Sabrina de Azevedo Silveira

Mayron César de Oliveira Moreira

José Elias Cláudio Arroyo
(Orientador)

*Dedico este trabalho a minha família e ao meu namorado Junior.*

# Agradecimentos

Agradeço primeiramente a Deus por me dar forças para concluir essa etapa. Agradeço aos meus pais João e Carminha pelo apoio e amor incondicional. Agradeço também ao meu orientador José Elias Arroyo e aos demais professores do Departamento de Informática da UFV. Agradeço também, à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo financiamento da minha bolsa durante o mestrado, e à Divisão de Apoio ao Desenvolvimento Científico e Tecnológico da Universidade Federal de Viçosa. Agradecimento em especial ao meu namorado Junior pelo companheirismo, paciência e por ser uma pessoa maravilhosa que amo muito.

# Sumário

**Lista de Figuras** **vi**

**Lista de Tabelas** **viii**

**Resumo** **ix**

**Abstract** **xi**

**1 Introdução** **1**

1.1 O Problema e sua Importância . . . . . 2

1.2 Objetivos . . . . . 5

1.2.1 Objetivos Específicos . . . . . 5

1.3 Estrutura do Trabalho . . . . . 6

**2 Sequenciamento de Lotes de Tarefas em Máquinas Paralelas** **7**

2.1 Definição do Problema Abordado . . . . . 7

2.1.1 Exemplo Numérico . . . . . 9

2.2 Modelo Matemático . . . . . 11

2.3 Revisão Bibliográfica . . . . . 13

**3 Meta-Heurísticas Propostas para o Problema $P|rj, dj, batch, incompatible family|\sum w_j T_j$** **16**

3.1 Solução Inicial . . . . . 17

3.2 Busca Local . . . . . 18

3.3 Adaptive Large Neighborhood Search . . . . . 19

3.3.1 Métodos de Remoção . . . . . 21

3.3.2 Métodos de Inserção . . . . . 23

3.3.3 Escolhendo um Método de Remoção e Inserção . . . . . 24

3.3.4 Aspectos da Heurística ALNS . . . . . 25

3.3.5 Estrutura da Heurística ALNS . . . . . 26
3.4 Iterated Greedy . . . . . 28
3.4.1 Etapa de Destruição_Construção . . . . . 29
3.4.2 Estrutura do Iterated Greedy . . . . . 30
3.5 Simulated Annealing . . . . . 31
3.5.1 Processo de Pertubação . . . . . 33

**4 Experimentos Computacionais** **35**
4.1 Instâncias do Problema . . . . . 36
4.1.1 Instâncias de Pequeno Porte . . . . . 36
4.1.2 Instâncias da Literatura . . . . . 36
4.2 Métricas para Avaliação das Heurísticas . . . . . 37
4.3 Calibração dos Parâmetros das Heurísticas . . . . . 38
4.3.1 Calibração dos Parâmetros do ALNS . . . . . 38
4.3.2 Calibração dos Parâmetros do IG . . . . . 40
4.3.3 Calibração dos Parâmetros do SA . . . . . 41
4.4 Implementação das Heurísticas da Literatura . . . . . 42
4.5 Experimento 1: Teste com Instâncias de Pequeno Porte no *solver ILOG/CPLEX* . . . . . 45
4.6 Experimento 2: Comparações entre métodos heurísticos . . . . . 48
4.7 Análise de Convergência das Heurísticas . . . . . 56
4.8 Conclusão . . . . . 57

**5 Conclusão** **60**

**Referências Bibliográficas** **62**

# Lista de Figuras

2.1 Exemplo de sequenciamento com 10 tarefas e 2 máquinas . . . . . 10

4.1 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para calibração do ALNS . . . . . 40
4.2 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para calibração do IG . . . . . 41
4.3 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para calibração do SA . . . . . 42
4.4 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para os algoritmos MA, $MA^*$, $MA_1$, $MA_2$, $MA_3$ . . . . . 45
4.5 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para os algoritmos MA, $MA^*$, $MA_1$, $MA_2$, $MA_3$ . . . . . 46
4.6 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação de RPDs (%) entre as heurísticas e o modelo matemático . . . . . 47
4.7 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do RPI (%) das heurísticas para o critério de parada $= 0.05 \times n$ . . . . . 51
4.8 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do RPI (%) das heurísticas para o critério de parada $= 0.1 \times n$ . . . . . 52
4.9 Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do RPI (%) das heurísticas para o critério de parada $= 0.2 \times n$ . . . . . 52
4.10 Gráfico de médias e itervalos HSD de Tukey com nível de confiança de 95% para comparação da heurísticas ALNS, IG e SA para o critério de parada $= 0.05 \times n$ . . . . . 55

4.11 Gráfico de médias e itervalos HSD de Tukey com nível de confiança de 95% para comparação da heurísticas ALNS, IG e SA para o critério de parada = $0.1 \times n$ . . . . . 55
4.12 Gráfico de médias e itervalos HSD de Tukey com nível de confiança de 95% para comparação da heurísticas ALNS, IG e SA para o critério de parada = $0.2 \times n$ . . . . . 55
4.13 Convergência das soluções ao decorrer do tempo usando os algoritmos ALNS, IG e SA . . . . . 57
4.14 Convergência das soluções ao decorrer do tempo usando os algoritmos ALNS, IG e SA . . . . . 57
4.15 Convergência das soluções ao decorrer do tempo usando os algoritmos ALNS, IG e SA . . . . . 57

# Lista de Tabelas


2.1 Características da instância com 10 tarefas e 2 máquinas . . . . . 10
2.2 Cálculo do Atraso de cada Tarefa . . . . . 11

4.1 Características das instâncias de pequeno porte . . . . . 36
4.2 Características das instâncias de grande porte . . . . . 37
4.3 Conjunto de valores de parâmetros testados para a calibração do ALNS . 39
4.4 Conjunto de valores de parâmetros testados para a calibração do ALNS . 41
4.5 Conjunto de valores de parâmetros testados para a calibração do SA . . . 42
4.6 Valores ótimos da execução do modelo matemático para o conjunto de 288 instâncias . . . . . 47
4.7 Desvio médio entre o limitante superior fornecido ao CPLEX em relação a melhor solução encontrada pelo CPLEX . . . . . 48
4.8 Porcentagem de melhoria das heurísticas em relação ao algoritmo MA considerando o critério de parada = $0.05 \times n$ . . . . . 49
4.9 Porcentagem de melhoria das heurísticas em relação ao a algoritmo MA considerando o critério de parada = $0.1 \times n$ . . . . . 49
4.10 Porcentagem de melhoria das heurísticas em relação ao algoritmo MA considerando o critério de parada = $0.2 \times n$ . . . . . 50
4.11 Valores médios de RPDs para as heurísticas ALNS, IG e SA para o critério de parada = $0.05 \times n$ . . . . . 53
4.12 Valores médios de RPDs para as heurísticas ALNS, IG e SA para o critério de parada = $0.1 \times n$ . . . . . 53
4.13 Valores médios de RPDs para as heurísticas ALNS, IG e SA para o critério de parada = $0.2 \times n$ . . . . . 54
4.14 Características das instâncias . . . . . 56

# Resumo

FIDELIS, Michele Bernardino, M.Sc., Universidade Federal de Viçosa, dezembro de 2017. **Meta-heurísticas para o problema de sequenciamento de lotes de tarefas em máquinas paralelas**. Orientador: José Elias Claudio Arroyo.

Este trabalho aborda um problema de sequenciamento (scheduling) onde as tarefas são processadas em lotes em máquinas paralelas idênticas. Neste problema uma máquina pode executar um conjunto (lote) de tarefas simultaneamente. Além disso, as tarefas são classificadas em famílias, onde uma família agrupa tarefas que possuam alguma característica em comum. Assim, os lotes devem conter somente tarefas de uma mesma família. O problema também considera tarefas com diferentes tempos de chegada (release times) e tempos de processamento. As tarefas possuem ainda uma data de entrega e uma prioridade. O problema consiste em determinar os lotes (grupos) de tarefas para serem sequenciados nas máquinas de tal maneira que o atraso total ponderado das tarefas seja minimizado. O problema envolvendo sequenciamento de lotes, que é uma extensão do sequenciamento de tarefas clássico (onde uma máquina processa somente uma tarefa por vez), possui muitas aplicações reais, como em indústrias de fundição, de fabricação de móveis, de processamento de metais, de processamento de alimentos, farmacêuticas e de semicondutores. Para resolver o problema abordado, três algoritmos baseados em meta-heurísticas foram desenvolvidos: Adaptive Large Neighborhood Search (ALNS), Iterated Greedy (IG) e Simulated Annealing (SA). Todos estes algoritmos utilizam técnicas de busca em vizinhança para melhorar a qualidade de uma solução. As meta-heurísticas ALNS, IG e SA possuem estruturas simples e elas têm sido aplicadas satisfatoriamente para resolver diferentes problemas de otimização combinatória, especialmente problemas de sequenciamento da produção, o que justifica a utilização para o problema em estudo. Experimentos computacionais, utilizando dados da literatura foram realizados a fim de avaliar o desempenho dos algoritmos. Os resultados são comparados com os resultados gerados por dois algoritmos da literatura (Memetic Algorithm e Variable

Neighborhood Search) e com os resultados da resolução do modelo matemático do problema. Os experimentos e testes realizados demonstram que os algoritmos desenvolvidos neste trabalho geram soluções válidas de excelente qualidade superando as melhores soluções apresentadas na literatura.

# Abstract

FIDELIS, Michele Bernardino, M.Sc., Universidade Federal de Viçosa, December, 2017. **Meta-Heuristics for the problem parallel batch processing machines**. Adviser: José Elias Claudio Arroyo.

This work addresses a scheduling problem where the jobs are processed in batches on a identical parallel machines. In this problem a machine can process a set (batch) of jobs simultaneously. In addition, jobs are classified into families, where a family groups jobs that have some characteristic in common. Thus, the batches must contain only jobs of a same family. Also, the problem considers jobs with different release times and processing times. The jobs also have a due date and a priority. The objective of the problem is to group the job set into batches and assign the batches to the parallel machines in order to minimize the total weighted tardiness of the jobs. The parallel batch processing machines scheduling problem, that is an extension of classic job sequencing(where a machine processes only one task at a time) has many real application, such as in the foundry industry manufacturing, food processing industries, pharmaceutical industries and semiconductor industries. To solve this problem, three algorithms based on meta-heuristics were developed: Adaptive Large Neighborhood Search (ALNS), Iterated Greedy (IG) and Simulated Annealing (SA). All of these algorithms use neighborhood search techniques to improve the quality of solutions. In addition, the meta-heuristics ALNS, IG and SA have been applied satisfactorily to solve different combinatorial optimization problems. Computational experiments using literature data were performed to evaluate the performance of the algorithms. The obtained results are compared with the results generated by two algorithms from the literature (Memetic Algorithm and Variable Neighborhood Search) and with the mathematical model of the problem. The computational experiments demonstrate that the algorithms developed in this work generate valid solutions of excellent quality, outperforming the best solutions presented in the literature.

# Capítulo 1

# Introdução

O processo de globalização está em constante evolução e transformação. É notório que as transformações tecnológicas causem um impacto em muitas indústrias, além disso, com o crescimento econômico mundial criou-se um cenário competitivo entre as indústrias, no qual se faz necessário a utilização dos recursos da forma mais eficiente possível a fim de competir no mercado global. No contexto atual, a indústria de eletrônicos tem evoluído e tornado-se uma das maiores indústrias do mundo. Como exposto em [Mönch et al., 2011], na fabricação de semicondutores é de suma importância a capacidade de atender datas de vencimentos para satisfação dos clientes. Assim, a capacidade de cumprir com as datas e simultaneamente obter uma redução do tempo do ciclo de tarefas tornou-se um fator determinante para o sucesso na competição acirrada do mercado global. Logo, o investimento em estratégias competitivas que possam aperfeiçoar e viabilizar sistemas de produção são essenciais no mercado industrial atual. Neste contexto, a utilização da programação da produção como ferramenta para auxiliar na tomada de decisão torna-se essencial para a competitividade das empresas.

A programação da produção consiste em um complexo processo de tomada de decisão, que está relacionada a um sistema de coleta de informações, planejamento e programação [Herrmann et al., 2005]. Este processo pode ser definido como a alocação de um conjunto de recursos, disponíveis ao longo do tempo, que possam satisfazer da melhor maneira possível algum critério de otimalidade [Graves, 1981]. Portanto, problemas da programação da produção geralmente consistem em problemas de otimização combinatória.

Sistemas de produção consistem em complexos sistemas estocásticos e dinâmicos presentes em muitas indústrias. A utilização da programação da produção tem como finalidade aumentar a produtividade minimizando os custos de operações,

sendo capaz de identificar recursos em conflitos, determinar a sequência em que certas tarefas serão realizadas, alocar recursos, determinar a melhor forma de entrega dos produtos, identificar períodos ociosos nas instalações, entre outras atividades [Herrmann et al., 2005].

A programação da produção possui um importante papel em muitas indústrias e sistemas de produção, assim como em alguns ambientes de processamento de informação. Entre os processos presentes na programação da produção encontra-se o sequenciamento (*scheduling*). Segundo [Wight, 1984], *scheduling* pode ser definido como o estabelecimento do tempo para executar uma tarefa. O sequenciamento consiste em um processo de tomada de decisão, possuindo um importante papel em muitas indústrias de manufaturas e serviços. Problemas envolvendo sequenciamento geralmente consistem em designar um conjunto de tarefas a um conjunto de máquinas com a finalidade de minimizar algum critério individual ou um conjunto de critérios. Geralmente o critério de otimização pode estar relacionado ao tempo total de processamento das tarefas, ao atraso gerado pelas tarefas, aproveitamento de recursos físicos, atendimento de prazos, entre outros.

Na literatura há uma grande quantidade de trabalhos que integram sequenciamento com técnicas de loteamento *batching*. Segundo [Potts & Kovalyov, 2000], nesse tipo de técnica tarefas podem ser agrupadas quando compartilham o mesmo tempo de preparação (*setup*) em uma máquina ou quando a máquina pode processar várias tarefas simultaneamente. A utilização de lotes de tarefas em um processamento é motivada pelo ganho de eficiência, isto é, processar um conjunto de tarefas simultaneamente pode ser mais barato e mais rápido do que o processamento de tarefas individuais. Portanto, o estudo de problemas de sequenciamento que envolvam loteamento possuem um interesse prático, uma vez que muitos processos de fabricação necessitam utilizar de forma eficiente os recursos disponíveis com a finalidade de obter lucros e vantagens competitivas no mercado global.

## 1.1 O Problema e sua Importância

O sequenciamento de lotes de tarefas é um processo muito utilizado em indústrias, uma vez que facilita o processamento das tarefas que possuem alguma característica em comum. Em trabalhos encontrados na literatura o sequenciamento de lotes ocorre em duas versões: *serial-batching* e *parallel-batching* [Mathirajan & Sivakumar, 2006].

Em máquinas com *serial-batching* as tarefas podem ser processadas em lotes se

possuírem o mesmo valor de tempo de preparação, *setup*, em uma máquina. Nesse tipo de configuração um lote consiste em um conjunto de tarefas que podem ser processadas sequencialmente em uma mesma máquina. As tarefas possuem tempos de conclusão diferentes e o tempo de processamento do lote é a soma do tempo de processamento de todas as tarefas do lote. Segundo [Potts & Kovalyov, 2000], uma situação na qual este modelo apresenta benefícios consiste em situações nas quais as máquinas requerem um tempo de preparação para o processamento de tarefas que possuem diferentes características. O tempo de preparação da máquina pode consistir na necessidade de realizar uma mudança em alguma ferramenta ou na limpeza da máquina antes que uma tarefa inicie o seu processamento. O modelo no qual as tarefas possuem diferentes características é conhecido como modelo de sequenciamento de famílias. Nesse modelo, as tarefas são classificadas em famílias de acordo com a sua similaridade.

No *parallel-batching* as tarefas são processadas simultaneamente em um lote. Nessa situação, um lote consiste em um conjunto de tarefas que podem ser agrupadas e processadas simultaneamente em uma mesma máquina. Nesse tipo de configuração, todas as tarefas pertencentes ao mesmo lote, iniciam e terminam o processamento simultaneamente. O tempo de processamento do lote consiste no maior tempo de processamento entre as tarefas pertencentes ao lote. Segundo [Potts & Kovalyov, 2000], operações de processamento em lotes são comuns em indústrias de fabricação de semicondutores nas quais operações para fabricação de placas de circuito são realizadas em fornos que podem agrupar várias tarefas simultaneamente. Também podem ser encontradas em indústrias químicas nas quais algumas operações são realizadas em tanques ou fornos. Na literatura, o processamento de lotes paralelos é conhecido como o sequenciamento de máquinas paralelas de processamento em lotes (*scheduling of batch processing machine* (SBP)) [Mathirajan & Sivakumar, 2006].

Um outro tipo de classificação utilizada em problemas envolvendo sequenciamento de lotes consiste em classificar as tarefas em famílias. Uma família agrupa tarefas que tenham alguma característica em comum. Segundo [Cheng et al., 2008], problemas de loteamento podem ser classificados em função da compatibilidade das tarefas. A primeira categoria engloba problemas de loteamento com tarefas de famílias compatíveis, na qual, as tarefas podem ser agrupadas e processadas simultaneamente e o tempo de processamento do lote é determinado pelo maior tempo de processamento entre as tarefas pertencentes ao lote. A outra categoria refere-se a problemas com tarefas de famílias incompatíveis. Encontra-se com frequência em operações de difusão/oxidação em fábricas de semicondutores. Nessas operações, as tarefas são separadas e classificadas em famílias, de acordo com propriedades

químicas semelhantes, e somente tarefas da mesma família podem ser processadas simultaneamente.

[Potts & Kovalyov, 2000] apresenta também os modelos baseados em família que podem ser classificados em função da disponibilidade das tarefas após o seu processamento. A primeira variante é denominada disponibilidade de lote, *batch availability*. Neste tipo de situação, uma tarefa somente torna-se disponível para ser entregue ao cliente após todas as outras do mesmo lote terem sido processadas. Enquanto na disponibilidade de tarefas, *job availability*, as tarefas estão disponíveis imediatamente após o seu processamento.

Problemas envolvendo sequenciamento de lotes são encontrados também com frequência em indústrias de fabricação de semicondutores, especificamente na produção de circuitos integrados. A utilização de lotes, segundo [Mönch et al., 2011], pode ser encontrada principalmente em operações de oxidação/deposição/difusão utilizadas para produção de chips. Operações em fábricas desse tipo são operações de longa duração, muitas dessas operações envolve o processamento de lotes, isto é, o processamento simultâneo de um conjunto de tarefas em uma mesma máquina. O sequenciamento de lotes pode resultar na formação de uma longa fila de lotes não processados em frente a uma máquina, resultando em um tempo de maior duração para o processo. Assim, a necessidade de estratégias de sequenciamento para alcançar melhores resultados reduzindo os custos de produção são necessárias [Mönch et al., 2011].

Um outro fator que em muitos problemas é considerado refere-se ao tempo de chegada das tarefas. Em muitas operações reais, as tarefas não estão sempre disponíveis no início do sequenciamento. As tarefas podem estar disponíveis para o processamento em diferentes instantes de tempo. Isso reflete condições práticas em muitos sistemas de produção de multi-estados [Cheng et al., 2008].

Um exemplo real da utilização de lotes pode ser encontrado em [Damodaran & Vélez-Gallego, 2012], no qual os autores apresentam o problema que foi observado em uma pequena instalação de fabricação para eletrônicos, responsável pela montagem e testes das placas de circuito impresso. Nessa instalação, máquinas de processamento em lotes são usadas em larga escala para testar conjuntos de eletrônicos e detectar falhas prematuras antes desses equipamentos serem comercializados. Nesse tipo de indústria as placas são agrupadas em lotes e submetidas a câmaras de ciclos térmicos. Essas câmaras são máquinas de processamento em lote que podem testar várias placas simultaneamente. Essas máquinas possuem elevado custo de funcionamento, assim se faz necessário um planejamento para melhor utilização das máquinas visando minimizar o tempo de conclusão dos lotes de tarefas.

Este trabalho, aborda o problema de sequenciamento de máquinas paralelas de processamento em lote considerando o modelo baseado em famílias incompatíveis com disponibilidade de lotes. Nesse problema, tarefas da mesma família possuem o mesmo tempo de processamento, assim apenas tarefas da mesma família podem ser processadas simultaneamente em um lote. Este problema ocorre em algumas situações práticas, como em máquinas que realizam operações de difusão em fábricas de produção de chips. Devido à natureza química dessas operações, apenas tarefas que possuem características químicas semelhantes podem ser processadas simultaneamente [Chiang et al., 2010].

## 1.2 Objetivos

Verificar a viabilidade da aplicação de heurísticas ao problema $P|rj, dj, batch, incompatible family|\sum w_j T_j$, que possam melhorar os resultados dos algoritmos encontrados na literatura. Deseja-se desenvolver heurísticas que sejam eficientes para resolução do problema abordado, produzindo soluções de alta qualidade em tempo computacional aceitável.

### 1.2.1 Objetivos Específicos

Especificamente, pretende-se realizar os seguintes passos:

1 - Realizar um estudo sobre trabalhos presentes na literatura que abordam problemas relacionados ao sequenciamento de máquinas paralelas de processamento de lotes, a fim de identificar os principais conceitos e estratégias utilizadas para resolução de problemas semelhantes;

3 - Implementar algoritmos encontrados na literatura do problema com o objetivo de comparação dos resultados;

4 - Propor o uso de meta-heurísticas definindo as estruturas, a representação da solução e o procedimento de busca local utilizado;

5 - Realização de experimentos computacionais: As heurísticas propostas serão comparadas utilizando um conjunto de instâncias disponíveis na literatura considerando diferentes métricas(ex:função objetivo, tempo computacional);

6 - Comparar os resultados utilizando métodos estatísticos. Espera-se ao final deste trabalho implementar heurísticas de alta qualidade que melhorem os resultados de algoritmos da literatura.

## 1.3 Estrutura do Trabalho

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 apresenta as características do problema em estudo neste trabalho, assim como uma revisão da literatura sobre os trabalhos que abordam problemas em máquinas paralelas de sequenciamento de lotes. Também é apresentado a formulação do modelo matemático encontrado na literatura para o problema. No Capítulo 3, as meta-heurísticas propostas para solucionar o problema são apresentadas, enquanto no Capítulo 4, os experimentos realizados neste trabalho são apresentados. Por fim, o Capítulo 5 apresenta as conclusões deste trabalho.

# Capítulo 2

# Sequenciamento de Lotes de Tarefas em Máquinas Paralelas

## 2.1 Definição do Problema Abordado

Em geral, três decisões são necessárias para a resolução do problema: a primeira consiste na formação dos lotes seguida de sua alocação nas máquinas e, por fim, o sequenciamento dos lotes nas máquinas. O problema em estudo consiste em processar lotes de tarefas em máquinas paralelas levando em conta que cada tarefa pode possuir um atraso gerado pela espera ao iniciar seu processamento. Esse atraso causa uma ineficiência ao processamento, logo, se faz necessário o agrupamento e o sequenciamento dos lotes para que o atraso seja minimizado.

O problema abordado neste trabalho é definido como em [Chiang et al., 2010] e é descrito a seguir:

1. Seja um conjunto $N = \{1, ..., j, ..., n\}$ de $n$ tarefas e $M = \{1, ..., k, ...m\}$ um conjunto de $m$ máquinas paralelas idênticas de processamento em lote.

2. Cada tarefa $j \in N$ deve ser processada em apenas uma máquina $k \in M$. As tarefas possuem tamanhos unitários e as máquinas são de processamento em lote, ou seja, as tarefas podem ser agrupadas e processadas simultaneamente em uma máquina.

3. Cada tarefa pertence a uma família $f$ do conjunto $F = \{1, ..f, ..., f_{max}\}$. Cada família $f$ tem $n_f$ tarefas, assim, o número total de tarefas pode ser calculado como:

$$n = \sum_{f=1}^{f_{max}} n_f \tag{2.1}$$

4. Todas as tarefas da mesma família $f$ possuem o mesmo tempo de processamento $p_f$. Portanto, somente tarefas da mesma família podem ser processadas juntas.

5. Toda tarefa $j$ possui um tempo de chegada $r_j$, de forma que, o processamento da tarefa somente pode começar após este instante.

6. O tempo de processamento de um lote é igual ao maior tempo de processamento entre as tarefas pertencentes ao lote. Portanto, um lote $b$ com tarefas da família $f$ estará disponível para ser processado no tempo:

$$r_b = max\left\{r_j | j \in b\right\} \tag{2.2}$$

7. O tamanho máximo do lote é denotado por $B$, de forma que cada máquina pode processar no máximo $B$ tarefas ao mesmo tempo.

8. Cada tarefa $j$ possui também um peso ou prioridade $w_j$, e uma data de entrega $d_j$.

9. Quando uma máquina começa a processar um lote $b$ de tarefas, nenhuma interrupção é permitida. Assim, não é possível adicionar ou retirar tarefas do lote até que o processamento finalize.

10. O tempo de conclusão de uma tarefa $j$ é definido como $C_j$. Se o tempo de conclusão de uma tarefa ocorrer antes da sua data de entrega ($d_j$), não há atraso ($T_j = 0$). Caso contrário, o atraso da tarefa $j$ é calculado por $T_j = (C_j - d_j)$. Portanto, o atraso de uma tarefa $j$ é definido como $T_j = max(0, C_j - d_j)$. O atraso total ponderado (*Total Weighted Tardiness (TWT)*) é determinado como:

$$TWT = \sum_{j=1}^{n} w_j T_j. \tag{2.3}$$

11. O problema consiste em agrupar as tarefas em lotes e, em seguida, sequenciá-los nas máquinas a fim de minimizar o atraso total ponderado

O problema em estudo pode ser representado usando a notação $\alpha|\beta|\gamma$, introduzida por [Graham et al., 1979], como: $P|rj, dj, batch, incompatible family|\sum w_jT_j$, onde $P$ representa máquinas paralelas idênticas, $r_j$ indica o tempo de chegada, $d_j$ representa a data de entrega e $\sum w_jT_j$ representa a função objetivo que calcula o atraso total ponderado.

Como o problema de sequenciamento, $P|rj, dj, batch, incompatible family|\sum w_jT_j$, apresentado neste trabalho consiste em uma generalização do problema de sequenciamento com uma máquina, que foi provado ser NP-difícil [Pinedo, 2012], então o problema de sequenciamento de tarefas em máquinas paralelas de processamento em lote, apresentado neste trabalho, é considerado também NP-difícil. Para melhor entedimento do problema, a seguir um exemplo numérico é apresentado.

### 2.1.1 Exemplo Numérico

Uma solução $S$ para o problema é composta por um conjunto de lotes, nos quais tarefas da mesma família são agrupadas. Esses lotes são sequenciados em um conjunto de máquinas idênticas de processamento em lote. Neste trabalho, uma solução $S$ é representada como um vetor de $m$ listas no qual cada lista representa a sequência de lotes em cada máquina. Cada lista contém os lotes formados com as tarefas organizadas de acordo com a ordem do tempo de chegada. Um exemplo com 2 máquinas, 8 tarefas e 4 lotes pode ser representado por $M_1 = \{1, 3\}$ e $M_2 = \{4, 2\}$, no qual 1, 2, 3 e 4 representam os índices dos lotes que podem ser representados por: $B_1 = \{2, 4\}$, $B_2 = \{1, 3\}$, $B_3 = \{5, 6\}$ e $B_4 = \{7, 8\}$.

Para um maior entendimento das características do problema, apresentamos um exemplo considerando uma instância com $n = 10$ tarefas e $m = 2$ máquinas. As tarefas são classificadas em famílias, no exemplo há duas famílias, cada família contém 5 tarefas. Essas tarefas são agrupadas em lotes e os lotes sequenciados nas máquinas. Cada lote pode conter no máximo duas tarefas. A Tabela 2.1 apresenta os dados de cada tarefa, a família à qual a tarefa pertence ($f_j$), os valores do peso ($w_j$), tempo de chegada ($r_j$), tempo de processamento ($p_j$) e da data de entrega ($d_j$) de cada tarefa $j$.

A Figura 2.1 ilustra uma possível solução para o problema. Os lotes da família 1 estão representados pela cor rosa enquanto os lotes formados com tarefas da família 2 estão representados na cor azul. Os lotes da família 2, {6, 7} e {8, 9} foram sequenciados na máquina M0, enquanto os lotes da família 1, {3}, {1, 2} e {4, 5}, e o lote da família 2, {10}, foram sequenciados na máquina M1.

Como exposto anteriormente, um lote apenas pode conter tarefas pertencentes

Tabela 2.1: Características da instância com 10 tarefas e 2 máquinas

| *Tarefa* | $f_j$ | $w_j$ | $r_j$ | $p_j$ | $d_j$ |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 2 | 4 |
| 2 | 0 | 2 | 3 | 2 | 5 |
| 3 | 0 | 1 | 0 | 2 | 6 |
| 4 | 0 | 3 | 4 | 2 | 8 |
| 5 | 0 | 2 | 5 | 2 | 9 |
| 6 | 1 | 5 | 0 | 4 | 6 |
| 7 | 1 | 1 | 2 | 4 | 7 |
| 8 | 1 | 3 | 5 | 4 | 9 |
| 9 | 1 | 4 | 2 | 4 | 10 |
| 10 | 1 | 5 | 6 | 4 | 11 |

à mesma família. Embora o número máximo de tarefas no lote, tamanho do lote, no exemplo seja de 2 tarefas, é possível a formação de lotes com apenas 1 tarefa. Isto acontece em algumas instâncias e geralmente ocorre devido ao valor do tempo de chegada próximo a zero e/ou ao baixo tempo de processamento da tarefa. Um outro fator a ser observado é o tempo de início de processamento do lote. No exemplo, o lote formado pelas tarefas {6, 7} apenas começa o processamento no tempo igual a 2, embora a tarefa 6 já esteja disponível no tempo 0 para processamento. Assim, um lote somente inicia o seu processamento no maior tempo de chegada entre todas as tarefas pertencentes àquele lote. O peso da tarefa, representado por $w_j$, está relacionado com a prioridade da tarefa em ser realizada. Portanto, quanto maior o valor do peso associado a uma tarefa maior será sua prioridade em ser realizada.

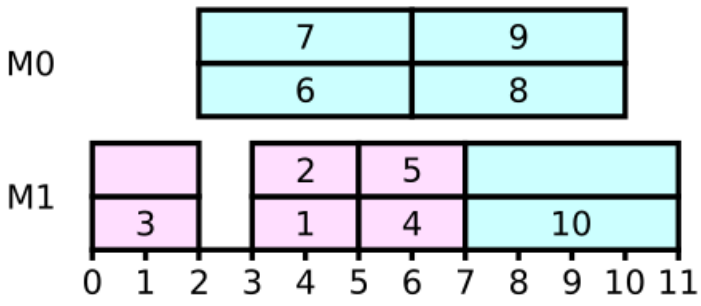



Figura 2.1: Exemplo de sequenciamento com 10 tarefas e 2 máquinas

A Tabela 2.2 apresenta o cálculo do atraso de cada tarefa. Cada linha da tabela representa os valores dos parâmetros utilizados para o cálculo do atraso. Os parâmetros utilizados são o tempo de processamento ($C_j$), a data de entrega ($d_j$) e o peso ($w_j$) de cada tarefa. A última coluna da tabela representa o cálculo do atraso

Tabela 2.2: Cálculo do Atraso de cada Tarefa

| *Tarefa* | $C_j$ | $d_j$ | $w_j$ | $w_jT_j$ |
|---|---|---|---|---|
| 1 | 5 | 4 | 1 | 0 |
| 2 | 5 | 5 | 2 | 0 |
| 3 | 2 | 6 | 1 | 0 |
| 4 | 7 | 8 | 3 | 0 |
| 5 | 7 | 9 | 2 | 0 |
| 6 | 6 | 6 | 5 | 0 |
| 7 | 6 | 7 | 1 | 0 |
| 8 | 10 | 9 | 3 | 3 |
| 9 | 10 | 10 | 4 | 0 |
| 10 | 11 | 11 | 5 | 0 |
| Atraso Total | | | | 3 |

ponderado para cada tarefa. Pode-se notar que a única tarefa que obteve atraso foi a tarefa 8, $T_8 = max\{0, 10 - 9\} = 1$, a qual possui um valor de peso igual 3, assim o atraso ponderado da tarefa $w_8T_8 = 3$.

## 2.2 Modelo Matemático

Nesta seção será apresentado o modelo de programação linear inteira (PLI) proposto por [Klemmt et al., 2009] para o problema em estudo. Esse trabalho utiliza a mesma notação introduzida em [Bilyk et al., 2014].

- $i, j \in \{1, ...n\}$ índice da tarefa;
- $k \in \{1, ..., m\}$ índice da máquina;
- $b \in \{1, ..., b_{max}\}$ índice para lotes;
- $f \in \{1, ..., f_{max}\}$: índice para famílias.

Os seguintes parâmetros são utilizados:

- $p_j$: tempo de processamento;
- $d_j$: data de entrega;
- $r_j$: tempo de chegada;
- $w_j$: peso da tarefa;
- $B$ : número máximo de tarefas em um lote;

- $G$ : número positivo.

As seguintes variáveis de decisão são utilizadas:

- $s_{bk}$ : tempo de início do lote $b_{esimo}$ na máquina $k$;
- $C_j$ : tempo de finalização do processamento da tarefa $j$;
- $T_j$ : atraso da tarefa $j$ pertencente a família $f$;
- $x_{jbk}$: é igual a 1 se a tarefa $j$ é processada no $b_{esimo}$ lote na máquina $k$. Caso contrário, é igual a zero;
- $y_{bkf(j)}$ : é igual a 1, se todas as tarefas $j$ no $b_{esimo}$ lote na máquina $k$ pertence a família $f$, caso contrário, igual a zero.

O modelo matemático é apresentado a seguir:

$$\text{Min} \sum_{j=1}^{n} w_j T_j \tag{2.4}$$

$$\text{Restrições: } \sum_{k=1}^{m} \sum_{b=1}^{b_{max}} x_{jbk} = 1 \ \forall j \tag{2.5}$$

$$\sum_{j=1}^{n} x_{jbk} \leq B \ \forall b, k \tag{2.6}$$

$$\sum_{f=1}^{f_{max}} y_{bkf(j)} = 1 \ \forall f, b, k \tag{2.7}$$

$$y_{bkf(j)} - x_{jbk} \geq 0 \ \forall j, b, k \tag{2.8}$$

$$r_j x_{jbk} \leq s_{bk} \ \forall j, b, k \tag{2.9}$$

$$s_{bk} + p_j x_{jbk} \leq s_{b+1,k} \ \forall j, b, k \tag{2.10}$$

$$G(1 - x_{jbk}) + C_j \geq s_{bk} + p_j \ \forall j, b, k \tag{2.11}$$

$$C_j - T_j \leq d_j \ \forall j \tag{2.12}$$

$$s_{bk} \geq 0; x_{jbk}, y_{bkf} \in \{0, 1\}; C_j, T_j \geq 0, \forall j, b, k, f \tag{2.13}$$

A equação (2.4) representa a função objetivo que é a minimização do $TWT$. A restrição (2.5) garante que cada tarefa pode somente ser processada em um lote e em uma máquina. Na restrição (2.6) o tamanho máximo do lote é restringindo. Restrição (2.7) e (2.8) garantem que somente tarefas da mesma família podem ser

processadas juntas. Restrição (2.9) garante que cada tarefa pode ser processada somente após seu tempo de chegada, enquanto na restrição (2.10) cada lote pode iniciar seu processamento somente após o termino do processamento do último lote presente na máquina. Restrição (2.11) restringe o tempo de processamento de cada tarefa. Restrição (2.12) expressa o atraso para cada tarefa, e finalmente, restrição (2.13) define que as variáveis sejam binárias e não negativas.

## 2.3 Revisão Bibliográfica

Problemas da programação da produção que utilizam o conceito de sequenciamento de lotes de tarefas são de grande importância em indústrias, como exposto anteriormente, e têm sido estudados extensivamente nos últimos anos [Potts & Kovalyov, 2000; Mathirajan & Sivakumar, 2006; Mönch et al., 2011].

O problema de sequenciamento de tarefas em máquinas paralelas com minimização do atraso ponderado, $P|rj, dj, batch, incompatible family|\sum w_jT_j$, tem sido abordado em vários trabalhos na literatura. Os trabalhos iniciais sobre sequenciamento de lotes de tarefas abordam um modelo mais simples, considerando máquinas de processamento serial de lote, conhecido como: *single batch machines*. Pode-se citar o trabalho de [Uzsoy, 1995] no qual propõe três algoritmos eficientes para a minimização do *makespan*, *maximum lateness* e *total weighted completion time*.

Com o objetivo de minimizar o atraso total em máquinas de processamento de lotes serial com tarefas de famílias incompatíveis ($1|batch, incompatible|\sum T_j$), [Mehta & Uzsoy, 1998] propõe um algoritmo de programação dinâmica que possui complexidade de tempo polinomial quanto ao número de famílias e a capacidade das máquinas são fixadas. O problema é decomposto em duas partes: agrupar as tarefas em lotes e sequenciá-los as máquinas. [Perez et al., 2005], amplia o trabalho para o problema $1|batch, incompatible|\sum w_jT_j$ no qual o objetivo é o de minimizar o atraso total ponderado, *Total Weighted Tardiness* (TWT).

Sistemas de produção reais necessitam da utilização de máquinas paralelas para evitar que o sistema seja bloqueado pela indisponibilidade de uma única máquina [Chiang et al., 2010]. Em [Balasubramanian et al., 2004], o problema $1|batch, incompatible|\sum w_jT_j$ é ampliando para o problema $P_m|batch, incompatible|\sum w_jT_j$. Os autores propõem duas versões do *Genetic Algorithm* (GA) para o problema. Na primeira versão do GA, a primeira etapa consiste em formar lotes de tarefas, em seguida designar os lotes formados para as máquinas utilizando o GA e por fim sequenciá-los nas máquinas. A segunda versão consiste

em designar tarefas para as máquinas utilizando o GA, e então formar os lotes em cada máquina e finalmente sequenciá-los.

O estudo apresentado por [Mönch et al., 2005] é uma extensão do artigo de [Balasubramanian et al., 2004]. Neste trabalho os autores abordam o problema $P_m|r_j, d_j, batch, incompatible|\sum w_j T_j$ no qual consideram tempo de chegada dinâmico para as tarefas, ou seja, as tarefas nem sempre estão disponíveis no início do processamento. O uso de tempo de chegada dinâmico torna o problema mais realístico, uma vez que em algumas indústrias exitem tarefas a serem realizadas que nem sempre encontram-se disponíveis no início do sequenciamento. O problema também torna-se mais complexo, visto que o processo de formação e sequenciamento dos lotes torna-se dependente do tempo de chegada das tarefas. O tempo de chegada dinâmico também é considerado no trabalho de [Damodaran & Velez-Gallego, 2010], no qual é proposta uma heurística para minimizar o makespan em máquinas paralelas de processamento em lotes com tarefas que possuem diferentes tempos de chegada. Um algoritmo *Ant Colony Optimization* (ACO) é proposto pelos autores apresentando bons resultados para o problema. [Herr & Goel, 2016] estudam o problema de sequenciamento de tarefas em uma máquina com o objetivo de minimizar o atraso total. Cada tarefa neste problema possui um tempo de processamento, uma data de chegada, pertencem a uma família e requerem uma certa quantidade de recurso antes do início do seu processamento.

O problema $P_m|r_j, d_j, batch, incompatible|\sum w_j T_j$ também é abordado em [Chiang et al., 2010]. Neste trabalho um *memetic algorithm* (MA) é proposto e os resultados apresentados no trabalho são comparados aos resultados do trabalho de [Mönch et al., 2005]. Experimentos demonstram que o algoritmo MA proposto supera os algoritmos GAs propostos no trabalho de [Mönch et al., 2005], obtendo maior eficiência e qualidade em relação a soluções.

[Bilyk et al., 2014] acrescenta restrições de precedência ao problema $P_m|r_j, d_j, batch, incompatible|\sum w_j T_j$ e propõem os algoritmos *Variable Neighborhood Search* (VNS) e *Greedy Randomized Adaptive Search Procedure* (GRASP) para o problema $P_m|r_j, d_j, p-batching, incompatible, prec|TWT$. Embora o problema principal do trabalho envolva precedência entre as tarefas, os autores realizam uma comparação entre os resultados obtidos pelo algoritmo VNS e o algoritmo MA proposto por [Chiang et al., 2010] para o problema sem restrições de precedência. Os resultados obtidos demonstram que o VNS é capaz de alcançar uma melhoria de 10,28% em relação ao MA quando o critério de parada é de um minuto por instância.

Alguns trabalhos na literatura abordam problemas de sequenciamento de máquinas paralelas de processamento em lotes considerando tarefas de tamanho

diferentes. [Cheng et al., 2013], [Jia et al., 2016] e [Jia et al., 2016] propõem uma adaptação do algoritmo *Ant Colony Optimization* (ACO) para o problema $P_m|B, s_j, batch|C_{max}$, no qual as tarefas possuem diferentes tamanhos $s_j$ e as máquinas possuem capacidades e velocidades de processamento idênticas. Esses estudos consideram o problema de sequenciar $n$ tarefas de diferentes tamanhos e de famílias incompatíveis em um conjunto de máquinas paralelas de processamento em lotes com o objetivo de minimizar o makespan. [Jia et al., 2015] consideram tarefas com tamanhos diferentes e máquinas de diferentes capacidades. Para minimizar o makespan, os autores propõem uma heurística construtiva baseada na regra *First-Fit-Decreasing* (FFD), assim como uma meta-heurística baseada em *Max-Min Ant System* (MMAS). [Kashan et al., 2008] propõem uma versão do algoritmo *Hybrid Genetic Heuristic* para o problema de máquinas paralelas de sequenciamento de lotes no qual as tarefas possuem diferentes tamanhos. O objetivo do trabalho consiste em minimizar o *makespan*. Neste tipo de problema, o tempo de processamento do lote é definido pelo maior tempo de processamento entre todas as tarefas pertencentes ao lote. Problemas de sequenciamento envolvendo *setup* também são abordados na literatura. O *setup* consiste em um tempo de preparação da máquina necessário para que a tarefa possa ser processada. Em [Allahverdi et al., 2008] e [Allahverdi, 2015] é apresentada uma extensa revisão de problemas de sequenciamento de tarefas nos quais o *setup* é considerado.

Este trabalho realiza o estudo do problema $P|r_j, d_j, batch, incompatible family|\sum w_j T_j$. Ao nosso conhecimento, os melhores resultados para o problema são encontrados nos trabalhos de [Chiang et al., 2010] e [Bilyk et al., 2014], nos quais os autores implementaram um algoritmo MA e um algoritmo VNS, respectivamente.

# Capítulo 3

# Meta-Heurísticas Propostas para o Problema $P|rj, dj, batch, incompatible family|\sum w_j T_j$

Neste capítulo estão descritas as meta-heurísticas propostas para a resolução do problema $P|rj, dj, batch, incompatible family|\sum w_j T_j$. A primeira heurística é baseada no algoritmo *Adaptive Large Neighborhood Search* (ALNS), que consiste em uma extensão do algoritmo *Large Neighborhood Search*. Embora a heurística ALNS seja muito utilizada em problemas de roteamento de veículos [Pisinger & Ropke, 2007], em trabalhos recentes ela tem apresentado soluções de alta qualidade para problemas de sequenciamento de tarefas [Muller, 2009], [Kovacs et al., 2012], [Iris et al., 2017] e [Palomo-Martínez et al., 2017]. A segunda heurística proposta é baseada na meta-heurística *Iterated Greedy* (IG), que é de simples implementação e que tem provado ser eficiente para resolver um grande número de problemas de sequenciamento de tarefas [Ruiz & Stützle, 2007]. A terceira heurística implementada é baseada na meta-heurística *Simulated Annealing* (SA) cuja ideia é alicerçada no processo de resfriamento de um material e que, experimentalmente, produz boas soluções nas primeiras iterações [Kirkpatrick et al., 1983].

A revisão da literatura realizada mostra que as meta-heurísticas ALNS, IG e SA ainda não foram aplicadas para o problema em estudo. Todas as heurísticas apresentadas neste trabalho utilizam a mesma heurística construtiva para a obtenção de uma solução inicial (Seção 3.1). Um procedimento de busca local também é utilizado pelas heurísticas para melhorar a solução inicial. A heurística IG utiliza a busca local também durante o processo de busca. O procedimento de busca local é descrito na Seção 3.2. As heurísticas ALNS, IG e SA são descritas nas Seções 3.3,

3.4, 3.5, respectivamente.

## 3.1 Solução Inicial

A solução inicial viável para o problema é obtida através da heurística construtiva *Time Window Decomposition* (TWD), proposta por [Bilyk et al., 2014]. Essa heurística utiliza duas regras de prioridade. A primeira regra, denominada *Apparent Tardiness Cost* (ATC), proposta por [Vepsalainen & Morton, 1987], determina a prioridade de uma tarefa $j$ ser inserida em um lote. O índice de prioridade para uma tarefa $j$ é calculado da seguinte forma:

$$I_{ATC,j}(t) = \frac{w_j}{p_j} exp \frac{(-(d_j - p_j - t + (r_j - t)^+)^+)}{k\bar{p}} \tag{3.1}$$

onde $k$ é um parâmetro escalar, $t$ é o tempo no qual a próxima máquina estará disponível e $\bar{p}$ é a média do tempo de processamento das tarefas que ainda não foram processadas. A notação $x^+$ é definida como $x^+ = max(x, 0)$. Esta regra calcula a prioridade de cada tarefa ser processada baseada nos parâmetros das tarefas.

A segunda regra é utilizada para escolher o lote que será sequenciado em uma máquina. A prioridade de um lote é calculada usando a regra BATC-II [Bilyk et al., 2014]. De acordo com essa regra, o valor do índice de um lote $b$ com $n_b$ tarefas é calculado:

$$I_{BATC-II,b}(t) = \frac{n_b}{B} w_j p_j \sum_{j \in J(b)} exp \frac{(-(d_j - p_j - t + (r_b - t)^+)^+)}{k\bar{p}} \tag{3.2}$$

onde $J(b)$ é o conjunto de tarefas presentes no lote $b$ e $r_b$ é o maior tempo de chegada das tarefas desse lote. A prioridade do lote é calculada em função dos principais valores dos parâmetros das tarefas. A heurística TWD é apresentada nos passos seguintes:

1. Quando uma máquina tornar-se disponível no tempo $t$, considere a janela de tempo $(t, t + \Delta t)$, onde $\Delta$ representa um intervalo de tempo pré-definido. Defina o conjunto de tarefas $M(f, t, \Delta t) = \{j | r_j \leq t + \Delta t, f(j) = f\}$ para cada família, isto é, tarefas da família $f$ com tempo de chegada menor do que $t + \Delta t$.

2. Ordene as tarefas no conjunto $M(f, t, \Delta t)$ em ordem decrescente pelo valor do índice ATC e derive o conjunto de tarefas $M^*(f, t, \Delta t) = \{j | j \in M(f, t, \Delta t), pos(j) \leq thres\}$, onde $thres$ representa o número predefinido de tarefas selecionadas, com as primeiras $thres$ tarefas de cada família. A

função $pos(j)$ é a posição da tarefa $j$ na sequência de tarefas que são incluídas no conjunto $M(f, t, \Delta t)$.

3. Considere todas as possíveis formações de lotes das tarefas do conjunto $M^*(f, t, \Delta t)$. Atribua a cada lote um valor de índice usando BATC-II. O lote com o maior valor de índice entre os lotes das diferentes famílias é escolhido e inserido na máquina disponível.

4. Defina uma nova janela de tempo e volte ao Passo 1, se houver tarefas que não foram processadas.

A heurística TWD, para avaliar todas as formações de lotes, depende dos valores dos parâmetros $k$, $\Delta t$ e $thres$. Assim como em [Bilyk et al., 2014] os valores dos parâmetros utilizados foram $k = 0.1 * h$ $(h = 1, ..., 50)$, $\Delta t = \bar{p}/2$ e $thres = 15$.

## 3.2 Busca Local

Heurísticas de busca local são desenvolvidas inserindo pequenas mudanças estruturais na solução atual. Ao utilizar um procedimento de busca local, uma heurística é capaz de investigar um número significativo de soluções em curto espaço de tempo. O processo de busca local pode ser utilizado para melhorar a solução inicial $S$ e as soluções resultantes do processo de perturbação das heurísticas.

A busca local utilizada neste trabalho é baseada em inserção de lotes, ou seja, um lote é removido da sua posição original e inserido em todas as outras possíveis posições do sequenciamento. Um lote pode ser inserido na mesma máquina de origem ou em outras máquinas.

O Algoritmo 1 apresenta o pseudocódigo da heurística de busca local proposta. O processo de BuscaLocal recebe como parâmetros de entrada uma solução $S$ e o número máximo de lotes $batchLimit$, para serem removidos aleatoriamente da solução. A melhor solução conhecida é armazenada na variável $S_b$ (linha 2). Na linha 3, a função $removeRandom$ remove aleatoriamente $batchLimit$ lotes da solução $S$ retornando a solução $S_p$. Na linha 4, a função $loteInsertion$ insere $batchLimit$ a solução parcial $S_p$. Neste processo é utilizado a estratégia de *best improvement*, assim, para cada lote removido, todas as possíveis inserções são testadas e a melhor solução da vizinhança $S_b$ é escolhida. O processo de BuscaLocal irá retornar a melhor solução obtida.

**Algoritmo 1:** BuscaLocal($S$, $batchLimit$)

```
1 inicio
2 |  S_b = S;
3 |  S_p = removeRandom(S, batchLimit);
4 |  S* = loteInsertion(S_p, batchLimit);
5 |  se TWT(S*) < TWT(S_b) então
6 |  |  S_b = S*;
7 |  fim
8 |  return S_b;
9 fim
```

## 3.3 Adaptive Large Neighborhood Search

A metaheurística *Adaptive Large Neighborhood Search* (ALNS), introduzida por [Ropke & Pisinger, 2006a] e [Ropke & Pisinger, 2006b], consiste em uma extensão da heurística *Large Neighborhood Search* (LNS) proposta por [Shaw, 1997] para resolver o problema de roteamento de veículos com janelas de tempo. Heurísticas baseadas em LNS pertencem a uma classe de algoritmos conhecidos como heurísticas *Very Large Scale Neighborhood Search* (VLNS) [Pisinger & Ropke, 2010].

O pseudocódigo da heurística LNS, segundo [Ropke & Pisinger, 2006a], é apresentado no Algoritmo 2. O parâmetro utilizado pela heurística LNS é o $q$, este parâmetro determina o número de movimentos realizados durante a remoção e a inserção, ou seja, no problema de sequenciamento o valor da variável $q$ determina o número de tarefas que são removidas e inseridas. Durante as iterações, realizadas entre as linhas 4 a 14, a solução inicial $S$ é gradualmente melhorada por alternar processos de remoção (linha 6) e reinserção (linha 7) na solução atual $S^*$. O critério de aceitação é verificado na linha 8, enquanto na linha 10 testamos se a solução atual $S^*$ melhora a melhor solução $S_{best}$. O algoritmo retorna a melhor solução $S_{best}$ encontrada, linha 15.

Na heurística LNS, a vizinhança é implicitamente definida por métodos, geralmente heurísticas, que são utilizadas para destruir e reparar uma solução [Pisinger & Ropke, 2010]. Segundo [Ropke & Pisinger, 2006a], algumas heurísticas podem ter dificuldades em mover de uma área de boas soluções para outro espaço de soluções quando o problema a ser resolvido possui restrições complexas. Ao contrário de uma simples busca local, os autores propõem a utilização de movimentos que podem rearranjar entre 30% - 40% de todas tarefas em uma única iteração. Embora o tempo computacional gasto para realizar a pesquisa em uma vizinhança extensa é consideravelmente alto, heurísticas LNS são capazes de escapar de ótimos locais e

encontrar soluções de alta qualidade explorando vizinhanças com tais características [Pisinger & Ropke, 2010].

**Algoritmo 2:** LNS($q$)

```
1  inicio
2      S = Solucao_Inicial();
3      S_best := S;
4      enquanto Critério_de_Parada não Satisfeito faça
5          S* := S;
6          remove q tarefas de S*;
7          reinsere q tarefas em S*;
8          se Critério_de_aceitacao(S*, S) então
9              S := S*;
10             se f(S*) < f(S_best) então
11                 S_best := S*;
12             fim
13         fim
14     fim
15     Retorna S_best;
16 fim
```

ALNS estende o LNS por permitir que múltiplas vizinhanças de remoção e inserção da solução sejam utilizadas durante o mesmo processo de busca. Heurísticas baseadas em LNS tem apresentado bons resultados para resolução de vários problemas de transportes [Shaw, 1997], [Ropke & Pisinger, 2006b], [Ropke & Pisinger, 2006a], [Pisinger & Ropke, 2010], [Ribeiro & Laporte, 2012], [Demir et al., 2012] e [Hemmelmayr et al., 2012]. Na literatura, há poucos trabalhos que utilizam a ALNS para resolver problemas de sequenciamento. [Lausch & Mönch, 2016] utilizam a ALNS para o problema de sequenciamento de lotes de tarefas no qual todas as tarefas estão disponíveis no tempo igual a zero. Por outro lado, [Kovacs et al., 2012] e [Pillac et al., 2013] estudam *service technician routing* e *scheduling problems and the technician routing*, respectivamente. O primeiro problema consiste em minimizar a soma dos custos totais de roteamento, no qual cada técnico possui um número de habilidades em diferentes níveis e cada tarefa demanda técnicos que possuem as habilidades apropriadas. O segundo problema consiste em designar equipes para atender a solicitações de serviço, levando em consideração as janelas de tempo.

Segundo [Pisinger & Ropke, 2007] a estrutura do ALNS apresenta várias vantagens, uma vez que a heurística ALNS além de ser uma heurística capaz de se adaptar a várias características das instâncias, produz soluções de alta qualidade.

Além disso, ALNS é capaz de explorar grandes partes do espaço de soluções de forma estruturada.

ALNS consiste em uma heurística de busca local composta de um número de sub-heurísticas concorrentes que são utilizadas para modificar a solução atual. Em cada iteração escolhe-se uma sub-heurística para destruir e reparar a solução atual, respectivamente. ALNS escolhe entre um número de heurísticas de inserção e remoção para intensificar e diversificar a pesquisa. A escolha dos métodos para remoção e inserção é de suma importância para a performance e robustez do algoritmo. Na implementação apresentada neste trabalho, foram escolhidas remoções baseadas na retirada de tarefas dos lotes, ocorrendo em algumas situações a remoção de lotes de tarefas. Embora esse processo de remoção seja simples e possa levar a soluções pobres em qualidades, o desempenho da heurística ALNS demonstrou ser elevado, uma vez que, os métodos utilizados são rápidos e um movimento ruim gerado por métodos de inserção diversificam o espaço de busca, podendo assim escapar de ótimos locais. Os métodos de remoção e inserção utilizados são descritos nas Subseções 3.3.1, 3.3.2, respectivamente.

Como exposto anteriormente, neste trabalho os métodos de remoção e inserção são realizados baseados em movimentos de tarefas. O parâmetro $taskLimit \in \{0, ..., n\}$ determina o número máximo de tarefas que serão removidas e inseridas durante as iterações. A escolha do valor do parâmetro influencia diretamente o desempenho da heurística ALNS. Em geral, ao utilizar valores baixos de $taskLimit$, tornar-se mais fácil a movimentação através do espaço de busca. Por outro lado, valores de $taskLimit$ altos tornam o processo de inserção mais lento.

A frequência que uma heurística é escolhida para modificar uma solução é baseada na análise estatística do histórico de seu desempenho. O método para selecionar heurísticas de remoção e inserção utilizadas durante o processo é descrito na Subseção 3.3.1, enquanto a Subseção 3.3.2 descreve o mecanismo de seleção baseado em métodos estatísticos utilizados durante a busca. A solução gerada após aplicação dos métodos de remoção e inserção é aceita se satisfazer algum critério definido previamente. Neste trabalho, o critério de aceitação é o da heurística *Simulated Annealing*. Enquanto o critério de parada utilizado na heurística é o tempo de CPU em segundos.

### 3.3.1 Métodos de Remoção

A heurística ALNS, implementada para o problema em estudo, utiliza diferentes heurísticas de remoção, constituindo o conjunto $N^-$ de métodos de remoção. Este

deve conter heurísticas que possam intensificar e diversificar o espaço de busca do problema. Como exposto em [Pisinger & Ropke, 2007], para diversificar a pesquisa, as heurísticas podem utilizar uma vizinhança de remoção aleatória. Neste problema, considera-se remover aleatoriamente tarefas de lotes aleatórios. A intensificação do espaço de busca utiliza remover vizinhanças baseadas na remoção de tarefas considerando o tempo de chegada, data de entrega ou tarefas que contenham o valor de $TWT$ elevado, ou seja, dentro de um lote encontra-se a tarefa que possui o maior valor de atraso ponderado e a remove. Essa seção descreve sete heurísticas de remoção. As heurísticas recebem como dados de entrada uma solução atual $S$ e o número de tarefas $taskLimit$ a serem removidas da solução. A saída das heurísticas consistem em uma solução parcial na qual $taskLimit$ tarefas são removidas. Os métodos consistem em uma adaptação da perturbação utilizada em [Kim et al., 2002]. As seguintes heurísticas são listadas abaixo:

- jobRemoveRandom($S, taskLimit$): Este método simples e rápido seleciona aleatoriamente $taskLimit$ tarefas para remover da solução atual $S$. Ao selecionar uma tarefa verifica-se a qual lote essa tarefa pertence para assim retira-la da solução. Esse processo é repetido até que $taskLimit$ tarefas sejam removidas.

- jobRemoveWorst:($S, taskLimit$): Todos os lotes pertencentes a um sequenciamento possuem um valor de atraso, se existir. Esse valor é dado pela soma do atraso das tarefas pertencentes a esse lote. A ideia desse método é baseada na remoção em função do atraso das tarefas, com o intuito de verificar a influência que a tarefa com o maior valor de atraso na solução pode causar. Assim, verifica-se o que ocorre com um lote após a retirada dessa tarefa. O algoritmo consiste em escolher aleatoriamente um lote e remover a tarefa que possuir maior valor de atraso ponderado. Esse processo é repetido até que $taskLimit$ tarefas sejam removidas.

- jobRemoveReady($S, taskLimit$): Neste problema, as tarefas possuem diferentes tempos de chegada, não estando sempre disponível no início do sequenciamento. Um lote consiste em um grupo de tarefas que possuem o mesmo tempo de processamento podendo ter diferentes tempo de chegada e datas de entrega. Assim, o tempo de chegada do lote é dado pelo maior valor de tempo de chegada entre as tarefas pertencentes ao lote. Através de testes computacionais, observou-se que em algumas situações, as tarefas pertencentes a um lote, mesmo já estando disponíveis, devem esperar para iniciar o processamento

devido a essa restrição do problema. Essa situação pode levar ao surgimento de atraso. Portanto, o objetivo desse método é verificar o impacto do tempo de chegada das tarefas nos lotes. O algoritmo escolhe aleatoriamente um lote e remove entre as tarefas aquela com menor valor de tempo de chegada. Esse processo é repetido até que $taskLimit$ tarefas sejam removidas.

- jobRemoveDueDate($S, taskLimit$): Este método possui a mesma ideia utilizada no método jobRemoveReady. No entanto, nosso foco consiste em remover tarefas que possuem valor de data de entrega $d_j$ baixo. O algoritmo escolhe aleatoriamente um lote e remove entre as tarefas aquela com menor valor de $d_j$. Esse processo é repetido até que $taskLimit$ tarefas sejam removidas.

- jobRemoveIndex ($S, taskLimit$): A ideia deste processo consiste em verificar a influência de dois parâmetros do problema: o tempo de processamento ($p_j$) e o peso ($w_j$) da tarefa. Assim, utiliza-se o resultado da razão $w_j/p_j$ para remover tarefas dos lotes. O processo consiste em selecionar um lote aleatório e remover a tarefa com maior valor de $w_j/p_j$ do lote. Este processo é repetido até que $taskLimit$ tenham sido eliminadas.

- batchRemove($S, taskLimit$): Escolhe um lote e remove todas as tarefas pertencentes ao mesmo, com a probabilidade de seleção do lote é igual ao atraso do lote dividido pelo atraso total dos lotes [Kim et al., 2002]. Esse processo é repetido até que $taskLimit$ tarefas tenham sido removidas.

- batchRemoveWorst:($S, taskLimit$): Neste processo os lotes são organizados em ordem decrescente dos valores do atraso. Em uma lista $L$ são inseridos os primeiros 30% dos lotes. A partir desta lista, um lote é selecionado de forma aleatória e retirado do sequenciamento. Este processo é executado até que $taskLimit$ tarefas tenham sido removidas.

### 3.3.2 Métodos de Inserção

Nesta seção, apresentamos o conjunto $N^+$ de heurísticas construtivas de inserção. Estes algoritmos possuem um caráter guloso, ao priorizar a inserção de uma tarefa no melhor lote da família correspondente. As buscas serão guiadas pela estratégia *best improvement*, que consiste na escolha do primeiro vizinho com melhor valor de função objetivo para a continuidade do percurso no espaço de busca. Esta seção descreve quatro heurísticas de inserção utilizadas neste trabalho. As heurísticas recebem como dados de entrada uma solução parcial $S$ e o número de tarefas $taskLimit$

a serem inseridas na solução. A saída consiste em uma nova solução gerada pela inserção de *taskLimit* tarefas. Os métodos consistem em uma adaptação da perturbação utilizada em [Kim et al., 2002]. As seguintes heurísticas são listadas abaixo:

- jobInsertRandom($S, taskLimit$): Neste método uma tarefa é inserida aleatoriamente em um dos lotes da mesma família. Caso não exista um lote disponível da mesma família, forma-se um novo lote com a tarefa e ele é inserido de forma aleatória em uma máquina escolhida também ao acaso. O processo é repetido até que *taskLimit* tarefas tenham sido inseridas.

- jobInsertOrd($S, taskLimit$): Neste método, as tarefas são ordenadas em ordem decrescente do valor do índice ATC (Equação 3.1). Este índice determina a prioridade de uma tarefa $j$ ser inserida em um lote. O processo é repetido até que *taskLimit* tarefas tenham sido inseridas.

- jobInsertBest($S, taskLimit$): A ideia dessa heurística consiste em verificar qual o melhor lote para inserir uma tarefa. Assim, testa-se todas as possíveis inserções para a tarefa e escolhe-se o lote que gere o menor valor de $TWT$ para a solução. Se não houver lotes da mesma família disponíveis, um novo lote é formado com a tarefa e é inserido de forma aleatória em uma máquina também escolhida ao acaso. O processo é repetido até que *taskLimit* tarefas tenham sido inseridas.

- jobInsert($S, taskLimit$): Este processo verifica se é melhor inserir uma tarefa em um lote ou formar um novo lote com a tarefa. Para isto, testa-se todas as inserções possível para a tarefa e a formação de um novo lote também é verificada. Deste modo, testamos, se o atraso total ponderado gerado é menor ao se inserir a tarefa ou ao formar um novo lote. O processo é repetido até que *taskLimit* tarefas tenham sido inseridas.

### 3.3.3 Escolhendo um Método de Remoção e Inserção

Todos os métodos de remoção e inserção descritos anteriormente são utilizados durante as iterações na heurística ALNS. Para controlar quais vizinhanças serão escolhidas, um valor de peso é designado para cada heurística. A escolha dos métodos para realizar a remoção e inserção da solução em cada iteração é realizada através do *roulette wheel selection principle*, um algoritmo estocástico no qual a ideia básica consiste em selecionar o método mais apto a ser aplicado utilizando um vetor de probabilidades.

Os valores das probabilidades dos métodos utilizados no ALNS são mapeados para uma linha de segmento contínua, na qual, o tamanho do segmento correspondente a um método é igual ao valor da probabilidade da ocorrência daquele método. Para selecionar o operador mais apropriado, um número aleatório é gerado e a heurística cujo segmento abrange o número aleatório gerado é selecionada. Como em [Pisinger & Ropke, 2007], um peso $w_i$ é designado para cada método de remoção e inserção. Inicialmente, todos os métodos possuem o mesmo valor de peso igual a 1. No decorrer das iterações do ALNS, quanto mais uma vizinhança $N_i$ de um conjunto $N$ tenha contribuído para a melhoria da solução, um valor de peso maior é obtido e portanto, maior será a probabilidade da vizinhança ser escolhida.

Portanto, para $N \in \{N^-, N^+\}$ métodos de remoção e inserção com peso $w_i$, $i \in \{1, 2, ..., N\}$, a probabilidade para escolher um método de remoção/inserção $j \in N$ é dada pela seguinte equação:

$$\frac{w_j}{\sum_{i=1}^{N} w_i} \tag{3.3}$$

onde $w_i$ corresponde ao peso atribuído ao método $i$ e $N$ representa os conjuntos de métodos de remoção e inserção. Vale ressaltar que os métodos de remoção e inserção são escolhidos independentemente.

### 3.3.4 Aspectos da Heurística ALNS

O ALNS utiliza um ajuste automático para atualizar o valor dos pesos $w_i$ dos métodos de remoção e inserção. O peso representa uma pontuação para cada método utilizado, que representa a performance do operador durante as iterações ocorridas.

A pesquisa na heurística ALNS é dividida em um número $L$ de segmentos. Um segmento corresponde a um número $nMax$ de iterações da ALNS. Após um número $nMax$ de iterações, os pesos $w$ dos operadores de remoção e inserção são atualizados. Utiliza-se um vetor de resultados para essa atualização. Os operadores de inserção utilizam o vetor presentado por $\pi^-$ e os operadores de inserção o vetor $\pi^+$. Cada posição do vetor armazena um valor de resultado correspondente a um operador. No início de cada segmento $L$, o valor armazenado em cada posição do vetor $\pi$ é inicialmente zero. Em cada iteração o valor armazenado pode ser incrementado utilizando valores pré estabelidos, esses valores representam o quão boa foi a solução envontrada após a aplicação do operador. Três parâmetros são utilizados para esse cálculo: $\alpha_1$, $\alpha_2$ e $\alpha_3$ [Ropke & Pisinger, 2006a].

Portanto, após aplicação de um método, se a solução resultante da iteração

for a melhor global, o valor do método é incrementado por $\sigma_1$. Caso contrário, se a solução obtida na iteração melhora a solução atual e não havia sido ainda encontrada, o valor do método no vetor $\pi$ é incrementando em $\sigma_2$. Caso contrário, se a iteração resulta em uma solução que ainda não havia sido aceita e o valor da solução encontrada é pior que o valor da solução atual, mas a solução é aceita, o valor do vetor é incrementado por $\sigma_3$.

Essa estratégia faz com que os métodos possam ser capazes de explorar novas partes do espaço da solução, uma vez que apenas soluções ainda não visitadas serão recompensadas. Assim como em [Ropke & Pisinger, 2006a], as soluções visitadas são acompanhadas atribuindo uma chave de *hash* a cada solução. A utilização de uma tabela *hash* tem como objetivo realizar uma busca rápida obtendo o valor desejado a partir de uma chave simples de pesquisa. Como a pontuação para um método é atribuída após a remoção e inserção, não é possível afirmar qual etapa foi responsável pela pontuação do método.

Ao completar $nMax$ iterações, os pesos dos métodos são atualizados utilizando o vetor $\pi$ que armazena a pontuação do método durante $nMax$ iterações. Como em [Ropke & Pisinger, 2006a], vamos denotar por $w_{ij}$ o peso do método $i$ no segmento $j$. No primeiro segmento o valor do peso de todas os métodos é igual a 1. Após o término do segmento $j$, o valor do peso de todos os métodos é atualizado pela seguinte equação:

$$w_{i,j+1} = w_{ij}(1-\rho) + \rho\frac{\pi_i}{a_i} \tag{3.4}$$

onde $w_{i,j+1}$ representa o novo peso do método $i$ que será utilizado no segmento $j+1$, $a_i$ é o número de vezes que o método $i$ foi utilizado durante o segmento $j$, $\pi_i$ a pontuação do método $i$ acumulada durante $nMax$ iterações e $\rho \in [0,1]$ é o fator de reação que controla o impacto dos pesos.

### 3.3.5 Estrutura da Heurística ALNS

O Algoritmo 3 apresenta o pseudocódigo da heurística ALNS [Pisinger & Ropke, 2007]. O algoritmo ALNS possui nove parâmetros de entrada: *Critério_ de_ Parada* (tempo de CPU), $nMax$ (número máximo de iterações), $taskLimit$, $batchLimit$, $x$, $\rho$, $\sigma 1$, $\sigma 2$ e $\sigma 3$. Denotamos por $S_{best}$ a melhor solução encontrada durante a pesquisa, enquanto $S$ a solução atual. Na linha 2, uma solução inicial é encontrada utilizando a heurística construtiva $TWT$, seguida da execução da busca local e a inicialização da solução $S_{best}$ nas linhas 3 e 4, respectivamente. As iterações do

algoritmo são realizadas entre as linhas 5 a 25, até que o critério de parada seja satisfeito. Durante cada iteração, iteração das linhas 7 a 23, a solução atual $S$ é submetida a um método de remoção e um de inserção escolhidos de acordo com sua performance em iterações passadas (linha 9 a 11). O conjunto de métodos de remoção e inserção são denotados por $N^-$ e $N^+$, respectivamente. Após a aplicação dos métodos de remoção e inserção, uma nova solução $S^*$ é gerada a partir da solução $S$ (linha 11). Entre as linhas 12 a 21 ocorre a verificação do critério de aceitação. A heurística ALNS implementada utiliza o critério de aceitação conhecido como regra de Metropolis utilizado no algoritmo Simulated Annealing [Ruiz & Stützle, 2007]. Uma solução pior é aceita com uma probabilidade calculada por $exp(-\Delta/T)$, onde $\Delta = TWT(S^*) - TWT(S)$ (diferença de custo entre a nova solução e a solução atual), e $T$ é a constante denominada temperatura que depende de um instante particular. A temperatura $T$ é calculado por $T = \sum_{j=1}^{n} p_j/(n * m * x)$, onde $p_j$ é o tempo de processamento da tarefa $j$, $n$ é o número de tarefas, $m$ o número de máquinas e $x$ é um valor inteiro. Na linha 24, os pesos dos métodos são atualizados. A linha 26 retorna a melhor solução encontrada.

**Algoritmo 3:** ALNS(*Critério_de_Parada*,$nMax$,$taskLimit$, $batchLimit$, $x$,$\rho$, $\sigma_1, \sigma_2, \sigma_3$)

```
1  inicio
2      S := Solucao_Inicial();
3      S := BuscaLocal(S, batchLimit);
4      S_best := S;
5      enquanto Critério_de_Parada não Satisfeito faça
6          Iter := 0;
7          enquanto Iter < nMax faça
8              Seleciona usando roulette wheel N_i ∈ N^-;
9              S* := Destroi(S, N_i);
10             Seleciona usando roulette wheel N_j ∈ N^+;
11             S* := Repara(S*, N_j);
12             Δ = TWT(S*)−TWT(S);
13             se TWT(S*) < TWT(S) então
14                 S := S*;
15                 se TWT(S*) < TWT(S_best) então
16                     S_best := S*;
17                 fim
18             senão
19                 se rand(0,1) ≤ exp(−Δ)/T então
20                     S := S*
21                 fim
22             fim
23             Atualiza os vetores de resultados π^- e π^+;
24             Iter := Iter + 1;
25         fim
26         Atualiza os pesos w de N^- e N^+;
27     fim
28     Retorna S_best
29 fim
```

## 3.4 Iterated Greedy

*Iterated Greedy* (IG) é uma heurística simples e eficiente proposta por [Ruiz & Stützle, 2007], utilizada com muito sucesso para resolver diferentes problemas de sequenciamento de tarefas, como em [Rodriguez et al., 2013], [Rodriguez et al., 2013] e [Fanjul-Peyro & Ruiz, 2010].

Neste trabalho, a fim de resolver o problema em estudo, é desenvolvida uma heurística baseada em IG na qual o tamanho da destruição é variável. Inicialmente

uma solução inicial é gerada usando a heurística TWD [Bilyk et al., 2014] e em seguida, a solução é melhorada pela busca local. Em cada iteração do IG, enquanto o critério de parada não é satisfeito, a solução atual $S$ é submetida ao processo de Destruição_Construção. Na primeira etapa a solução $S$ é parcialmente destruída, retirando-se aleatoriamente tarefas dos lotes sequenciados nas máquinas. O número de tarefas retiradas ($jobLimit$) varia em cada iteração do IG. Ao final do processo de destruição, obtém-se uma solução parcial ($S_p$). Na etapa de Reconstrução, as tarefas são inseridas na solução parcial ($S_p$) de forma gulosa. Em seguida, a solução obtida $S^*$ é melhorada pelo processo de busca local gerando a solução $S^{**}$ e então, verifica-se o critério de aceitação. Ao término do processo a melhor solução encontrada é retornada.

Nas próximas seções estão descritos as etapas de Destruição_Construção e buscaLocal utilizadas pela heurística IG adaptada do trabalho [Ruiz & Stützle, 2007] para o problema em estudo.

### 3.4.1 Etapa de Destruição_Construção

A fase de Destruição_Construção é baseada na retirada e inserção de tarefas consistindo de duas etapas. Na primeira etapa, chamada de destruição, remove-se aleatoriamente $t$ diferentes tarefas da solução atual. Resumidamente, um lote $b$ é selecionado de forma aleatória e então uma tarefa $i$ presente neste lote é aleatoriamente selecionada e removida para ser inserida no conjunto $JR$. Este processo é repetido $t$ vezes obtendo uma solução parcial $S_p$. A fim de controlar o número de tarefas a serem retiradas da solução durante a etapa de destruição, o parâmetro $t$ é utilizado. O valor do parâmetro inicialmente igual a 1 é incrementado em um a cada iteração se não ocorrer melhora da solução após aplicados os processos de Destruição_Construção e buscaLocal. O valor de $t$ é novamente um quando ocorre melhora da solução ou quando atinge um valor de limite máximo, denotado por $jobLimit$.

A próxima etapa é a construção. Nesta etapa uma tarefa $i$ é aleatoriamente selecionada do conjunto $JR$ e reinserida em todas os lotes da mesma família que não estejam cheios. A tarefa $i$ é inserida na posição que resulte no menor valor de atraso ponderado total para a solução parcial $S_p$. Esse processo é repetido até que o conjunto $JR$ fique vazio, ou seja, até que todas as tarefas do conjunto $JR$ tenham sido reinseridas em $S_p$, obtendo-se uma nova solução completa $S$.

### 3.4.2 Estrutura do Iterated Greedy

O pseudocódigo da heurística IG é apresentado no Algoritmo 4. Os parâmetros de entrada da heurística IG são: o $Criterio\_de\_parada$, o parâmetro $jobLimit$, que representa o número de tarefas que serão retiradas da solução $S$ para formar a solução parcial $S_p$, e o parâmetro $batchLimit$ denotando o número de lotes retirados durante o processo de BuscaLocal.

Na linha 2 uma solução inicial $S$ é obtida utilizando a heurística construtiva $TWD$ e em seguida, linha 3, melhorada pelo processo de BuscaLocal. Nas linhas 4 e 5, o tamanho da destruição $t$ (número de tarefas a serem retiradas da solução na etapa de destruição) e a melhor solução $S_b$ são inicializados, respectivamente. Entre as linhas 6 a 21 as iterações do IG são realizadas até que o critério de parada seja satisfeito. Em cada iteração a solução atual $S$ é submetida ao procedimento de Destruição-Construção (linha 7) e a solução obtida é melhorada pelo processo de BuscaLocal, obtendo-se a solução $S^*$ (linha 8). Entre as linhas 9 a 14, se a solução $S^*$ for melhor que a solução atual $S$, o tamanho da destruição $t$ é reinicializado em 1 e a melhor solução $S_b$ obtida é atualizada. Se a solução atual não é melhorada, o tamanho da destruição $t$ é incrementado em um (linha 15). O valor máximo que $t$ pode assumir é limitado por $jobLimit$ se $t$ excede $jobLimit$, o valor de $t$ é novamente 1 (linhas 16 e 17). Se a solução $S^*$ é admitida pelo critério de aceitação (linha 18 e 19), a solução atual $S$ é substituída pela solução $S^*$, mesmo sendo uma solução pior. A heurística IG utiliza a regra de Metrópolis como critério de aceitação, definida previamente na Seção 3.3.5.

**Algoritmo 4:** IG($Criterio\_de\_parada$,$d_{max}$, $jobLimit$,$batchLimit$)

```
1  inicio
2      S := Solucao_Inicial();
3      S := BuscaLocal(S,batchLimit);
4      t := 1;
5      S_b := S;
6      enquanto Critério_ de_ parada não satisfeito faça
7          S* := Destruição_Construção(S, t);
8          S* := BuscaLocal(S*,batchLimit);
9          Δ := TWT(S*)−TWT(S);
10         se TWT(S*) < TWT(S) então
11             t := 1;;
12             S := S*;
13             se TWT(S*) < TWT(S_b) então
14                 S_b := S*;
15         senão
16             t := t + 1;
17             se t > jobLimit então
18                 t := 1;
19             se rand(0, 1) ≤ exp(−Δ)/T) então
20                 S := S*
21         fim
22     fim
23 fim
```

## 3.5 Simulated Annealing

A metaheurística Simulated (SA) proposta por [Kirkpatrick et al., 1983] tem sido utilizado em muitos problemas de sequenciamento [Kim et al., 2002; Li et al., 2011; Damodaran & Vélez-Gallego, 2012]. A heurística SA utilizada neste trabalho é baseada no estudo de [Damodaran & Vélez-Gallego, 2012], aplicado no contexto de um grupo de máquinas idênticas de processamento de lotes. O SA proposto por [Damodaran & Vélez-Gallego, 2012] consiste em inicializar o problema com uma solução com número de lotes reduzidos e pesquisar por soluções melhores sem que o número de lotes seja alterado. Essa pesquisa é realizada até que não ocorrra melhora em um número fixo de iterações. Após essas iterações um novo lote é criado e a pesquisa continua. A ideia proposta pelos autores consiste em controlar o número de lotes criados.

Este trabalho propõe uma heurística SA na qual a vizinhança de soluções é

gerada por sete mecanismos de perturbação. A heurística utiliza múltiplos reinícios, isto é, quando a temperatura tem um valor baixo (muito próximo de zero), o algoritmo é reiniciado da melhor solução obtida. O pseudocódigo da heurística SA é apresentado no Algoritmo 5. Como parâmetros de entrada, temos: *Critério_de_Parada* (tempo de CPU), $T_0$ (temperatura inicial), $\alpha$ (fator de resfriamento), $nMax$ (número máximo de iterações) e o parâmetro *batchLimit*, utilizado na busca local.

Na linha 2, uma solução inicial é determinada utilizando a heurística TWD. $S_{best}$ é a melhor solução global, $S^*$ a melhor solução para um dado número de iterações e $S$ é a solução atual. As iterações do algoritmo são realizadas entre as linhas 5 a 29 até que o critério de parada seja satisfeito. Na linha 6, a temperatura $T$ é inicializada. Durante cada iteração a solução atual $S$ é submetida a iterações entre as linhas 7 a 28, até que o valor da temperatura $T$ satisfaça a condição de parada. Para um dado valor de temperatura $T$, o algoritmo realiza um número fixo de iterações $nMax$ (linha 9) nas quais a solução atual $S$ é submetida ao processo de perturbação gerando a solução $S'$ (linha 11). Entre as linhas 12 a 20, o critério de aceitação é verificado. Se a solução $S'$, gerada pelo processo de pertubação, melhora a solução atual $S$ e o valor da solução de $S^*$, o valor das soluções é atualizado, linha 13 e 15, respctivamente. No SA, uma solução pior é aceita com uma dada probabilidade dada por $exp^{-((TWT(S')-TWT(S))/T)}$, onde $TWT(S')-TWT(S)$ é a diferença de custo entre uma nova solução e a solução atual e $T$ é a constante de temperatura que depende de um instante particular. Na linha 22, a temperatura $T$ é decrementada por uma fator $\alpha$. O SA proposto neste trabalho utiliza dois métodos de pertubação denominados por adiciona_NovoLote e Deleta_Lote. Esses métodos são utilizados para controlar o número de lotes da solução. No primeiro método um lote é criado e adicionado ao processamento. Neste método, uma tarefa é selecionada de forma aleatória de um lote e um novo lote é formado e adicionado ao processamento. O novo lote formado pela tarefa é sequenciado em uma máquina aleatória. No segundo método, verificamos se é possível excluir um lote do processamento. A retirada de um lote consiste em verificar se é possível retirar as tarefas pertencentes ao lote e se a retirada do lote levará a um valor de $TWT$ menor que o da solução atual. Se o valor $TWT$ da solução $S^*$ for menor que o valor da melhor solução global $S_{best}$, o valor de $S_{best}$ é atualizado (linha 23) e o método *Adiciona_NovoLote($S^*$)* (linha 24) é aplicado. Caso não ocorra uma melhora, então aplica-se o método *Delete_Lote($S^*$)* (linha 27) para verificar se é possível remover um lote. Na linha 30 o SA retorna a melhor solução encontrada.

**Algoritmo 5:** SA($Criterio\_de\_Parada, T_0, \alpha, nMax$)

```
1  inicio
2      S = Solucao_Inicial();
3      S* = BuscaLocal(S, batchLimit);
4      S_best = S*;
5      enquanto Criterio_de_Parada não satisfeito faça
6          T = T0;
7          enquanto T > 0.0001 faça
8              Iter = 0;
9              enquanto Iter < nMax faça
10                 S'= Processo_de_Pertubação(S);
11                 Δ = TWT(S')−TWT(S);
12                 se TWT(S') < TWT(S) então
13                     S = S';
14                     se TWT(S') < TWT(S*) então
15                         S* = S';
16                 senão
17                     se random(0,1) < exp^{−Δ/T)} então
18                         S = S';
19                 fim
20                 Iter = Iter + 1;
21             fim
22             T = α × T ;
23             se TWT(S*) < TWT(S_best) então
24                 S_best = S*;
25                 S= Adiciona_NovoLote(S*);
26             senão
27                 S= Delete_Lote(S*);
28         fim
29     fim
30     Retorna S_best;
31 fim
```

### 3.5.1 Processo de Pertubação

O Processo de Perturbação utilizado na heurística SA consiste em uma adaptação da perturbação utilizada em [Kim et al., 2002]. Esta etapa possui um conjunto de vizinhanças na qual em cada iteração um vizinho de cada vizinhança é gerado. O vizinho que gerar o menor valor de $TWT$ é escolhido. As seguintes vizinhanças são utilizadas neste trabalho:

- swapBatch: Seleciona dois lotes da mesma família de forma aleatória e troca suas posições. O primeiro lote é selecionado baseado na taxa de atraso [Kim et al., 2002], ou seja, a probabilidade de um lote ser selecionado é igual a soma do atraso das tarefas pertences ao lote dividido pelo atraso total do processamento.

- batchInsert: Um lote é escolhido de forma aleatória e é inserido em uma nova posição em uma máquina de forma aleatória.

- batchMerge: Seleciona aleatoriamente dois lotes da mesma família. Retira-se as tarefas de ambos os lotes e as adiciona ao conjunto $N$. As tarefas são escolhidas de forma aleatória do conjunto $N$ e então são inseridas novamente aos lotes.

- batchDiv: Um lote é escolhido de forma aleatória e então ocorre a divisão do lote em dois novos lotes. Primeiramente verifica-se o número de tarefas do lote para então aleatoriamente escolher o número de tarefas em cada novo lote. Após a criação dos lotes esses são inseridos de forma aleatória em máquinas também escolhidas ao acaso.

- swapJob: Primeiramente escolhe dois lotes da mesma família de forma aleatória, para então selecionar de forma também aleatória duas tarefas para realizar a troca entre os lotes.

- jobInsertReadyTime: Seleciona aleatoriamente um lote e nesse lote seleciona a tarefa com menor valor de tempo de chegada,$r_j$. Retira-se essa tarefa e aleatoriamente adiciona a outro lote da mesma família que não esteja completo ou um novo lote é formado com essa tarefa.

- jobInsertDueDate: Similar ao método jobInsertReadyTime, no entanto, esse método seleciona a tarefa com menor valor da data de entrega $d_j$.

# Capítulo 4

# Experimentos Computacionais

Neste capítulo são descritos os detalhes dos experimentos computacionais realizados neste trabalho, assim como as métricas utilizadas para a análise dos resultados. Na Seção 4.1 as características do conjunto de instâncias utilizadas nos testes são apresentadas. A Seção 4.2 apresenta as métricas utilizadas para avaliar as heurísticas. Na Seção 4.3 a calibração de parâmetros dos métodos heurísticos implementados neste trabalho é realizada. Na Seção 4.4 é explicado detalhes da reimplementação do *Memetic Algorithm* (MA) proposto por [Chiang et al., 2010], e pelo *Variable Neighborhood Search* (VNS), proposto por [Bilyk et al., 2014]. Ambos algoritmos foram propostos para resolver o mesmo problema abordado neste trabalho.

Na Seção 4.5 são apresentados os testes realizados com o modelo matemático apresentado na Seção 2.2. As Seções 4.6 e 4.7 apresentam os resultados obtidos realizando uma comparação entre as heurísticas propostas com os algoritmos do estado-da-arte da literatura. Na Seção 4.6 os resultados obtidos pelas heurísticas são comparados em função do valor da função objetivo, enquanto na Seção 4.7 a comparação entre as heurísticas é realizada em função do tempo de execução. Por fim, na Seção 4.8 é realizada uma comparação entre as heurísticas analisando as vantagens e desvantagens de cada método proposto para o problema em estudo.

Todos os algoritmos foram implementados em C++ e executados em um Cluster no qual cada nó possui 2 processadores Intel(R) Xeon(R) CPU X5650(12M Cache, 2.66 GHz, 6.40 GT/s Intel(R) QPI, 6 cores, 12 threads) e 24 GB de RAM DDR3 1333 MHz. No entanto, os experimentos apresentados neste trabalho foram executados utilizando 1 processador, 4 threads e 4GB de RAM.

Tabela 4.1: Características das instâncias de pequeno porte

| **Parâmetros** | **Valores** |
|---|---|
| Número de Famílias ($f$) | 2, 4 |
| Número de Máquinas ($m$) | 2, 3 |
| Número de Tarefas ($n$) | 20, 40, 60, 80 |
| Tamanho Máximo do Lote ($B$) | 3, 4 |
| Tempo de Processamento da Família ($p_j$) | 2, 4, 10, 16 e 20 com Probabilidade de distribuição dos tempos 0.2, 0.2, 0.3, 0.2 e 0.1, respectivamente |
| Peso da Tarefa ($w_j$) | $\sim$U(0,1) |
| Tempo de Chegada ($r_j$) | $r_j \sim$ U(0, $\alpha \sum p_j/(m * B)$) $\alpha \in \{0.25, 0.5, 0.75\}$ |
| Data de Entrega ($d_j$) | $d_j - r_j \sim$ U(0, $\beta \sum p_j/(m * B)$) $\beta \in \{0.25, 0.5, 0.75\}$ |

## 4.1 Instâncias do Problema

Os experimentos computacionais realizados neste trabalho utilizam dois grupos de instâncias, a saber: instâncias de pequeno porte, geradas neste trabalho, e instâncias de grande porte da literatura. As subseções seguintes apresentam as características desses dois grupos de instâncias.

### 4.1.1 Instâncias de Pequeno Porte

Um conjunto de 288 instâncias de pequeno porte foi gerado com a finalidade de verificar se o modelo matemático consegue encontrar o valor ótimo para instâncias menores. Os valores do tempo de processamento das famílias ($p_j$), da prioridade ($w_j$), tempo de chegada ($r_j$) e data de entrega ($d_j$) das tarefas foram gerados da mesma maneira que [Chiang et al., 2010]. A Tabela 4.1 apresenta as características das instâncias de pequeno porte geradas. Essas instâncias foram utilizadas nos testes do modelo matemático apresentado na Seção 2.2.

### 4.1.2 Instâncias da Literatura

Os experimentos computacionais realizados utilizam um total de 4860 instâncias de grande porte produzidas e disponibilizadas por [Chiang et al., 2010] para o problema $P|rj, dj, batch, incompatible family|\sum w_j T_j$. As características das instâncias são apresentadas na Tabela 4.2. Esse grupo de instâncias foi utilizado para verificar a eficiência das heurísticas propostas neste trabalho em relação ao MA e ao VNS propostos por [Chiang et al., 2010] e [Bilyk et al., 2014], respectivamente.

Tabela 4.2: Características das instâncias de grande porte

| **Parâmetros** | **Valores** |
|---|---|
| Número de Famílias ($f$) | 3, 6, 12 |
| Número de Máquinas ($m$) | 3, 4, 5 |
| Número de Tarefas ($n$) | 180, 240, 300 |
| Tamanho Máximo do Lote ($B$) | 4, 8 |
| Tempo de Processamento da Família ($p_j$) | 2, 4, 10, 16 e 20 com Probabilidade de distribuição dos tempos 0.2, 0.2, 0.3, 0.2 e 0.1, respectivamente |
| Peso da Tarefa ($w_j$) | $\sim$U(0,1) |
| Tempo de Chegada ($r_j$) | $r_j \sim \mathrm{U}(0,\ \alpha \sum p_j/(m * B))$<br>$\alpha \in \{0.25, 0.5, 0.75\}$ |
| Data de Entrega ($d_j$) | $d_j - r_j \sim \mathrm{U}(0,\ \beta \sum p_j/(m * B))$<br>$\beta \in \{0.25, 0.5, 0.75\}$ |

## 4.2 Métricas para Avaliação das Heurísticas

Para avaliar o desempenho das heurísticas propostas, utiliza-se a métrica de Desvio Percentual Relativo (em inglês *Relative Percentage Deviantion*) (RPD). O RPD é definido pela seguinte equação:

$$RPD\% = \frac{f_{\text{Heuristica}} - f_{\text{best}}}{f_{\text{best}}} \times 100, \tag{4.1}$$

onde $f_{\text{Heuristica}}$ corresponde ao valor do TWT obtido por uma heurística proposta e $f_{best}$ é o menor valor de TWT para uma instância encontrada entre todas as heurísticas comparadas. Quanto menor o valor do RPD, melhor será a qualidade da solução.

Utiliza-se também a métrica RPI (*Relative Percentage Improve*) para avaliar a porcentagem de melhoria provocada por uma heurística em relação ao algoritmo MA proposto por [Chiang et al., 2010]. A métrica RPI é definida como:

$$RPI\% = \frac{f_{\text{MA}} - f_{\text{Heuristica}}}{f_{\text{MA}}} \times 100, \tag{4.2}$$

onde $f_{\text{MA}}$ corresponde ao valor do TWT obtido pelo algoritmo MA proposto por [Chiang et al., 2010] para uma determinada instância e $f_{Heuristica}$ é o menor valor de TWT encontrado após 10 execuções de determinada heurística implementada. Quanto maior o valor da métrica RPI, maior será a melhoria da heurística em relação ao algoritmo MA da literatura.

Apesar das métricas citadas acima utilizarem a melhor solução conhecida para comparação, na Seção 4.5 uma análise considerando algumas soluções ótimas são

realizadas utilizando a métrica RPD. Essas soluções foram obtidas pela resolução do modelo matemático, através do software IBM CPLEX 12.6.

O critério de parada utilizado pelas heurísticas nos experimentos é baseado na quantidade de tempo de CPU. Os seguintes valores de tempos são utilizados: $0.05 \times n$, $0.1 \times n$, $0.2 \times n$ segundos, onde $n$ é o número de tarefas da instância.

## 4.3 Calibração dos Parâmetros das Heurísticas

Nas seções seguintes a calibração dos parâmetros dos algoritmos ALNS, IG e SA é realizada. Para tanto, um novo conjunto de 486 instâncias foi gerado de acordo com os parâmetros adotados na Tabela 4.2.

As heurísticas foram executadas 10 vezes para cada instância. O critério de parada utilizado foi fixado no tempo de CPU de $0.1 \times n$ segundos, onde $n$ representa o número de tarefas da instância. Os resultados obtidos são analisados utilizando a métrica RPD, apresentada na seção 4.2. Uma Análise de Variância (ANOVA) é realizada para validar os resultados obtidos, assim como verificar se há diferenças significativas entre os valores testados com a finalidade de determinar a melhor configuração de parâmetros para cada heurística. Os testes estatísticos realizados neste trabalho consideram a ANOVA multi-factor, uma vez que as heurísticas propostas apresentam diversos fatores que podem influenciar no resultado da calibração. Assim, o uso da ANOVA multi-factor permite avaliar o impacto que os diferentes fatores provocam na calibração. É importante destacar também que os dados utilizados para os testes da ANOVA são gerados utilizando o método Full Factorial DOE (Design Of Experiment DOE), que consiste em gerar todas as combinações de parâmetros (fatores) utilizados na calibração de uma heurística [Eriksson et al., 2000].

Nas subseções seguintes os detalhes e resultados obtidos para a calibração dos parâmetros específicos de cada heurística são apresentados.

### 4.3.1 Calibração dos Parâmetros do ALNS

A fim de determinar a melhor configuração de valores para os parâmetros da heurística ALNS, são realizados testes computacionais preliminares a fim de calibrar os seguintes dados de entrada:

- $nMax$: Tamanho máximo do segmento, ou seja, número de iterações antes da atualização dos valores dos peso;

- *taskLimit*: Número de tarefas que serão removidas e inseridas nos processos de remoção e inserção;
- $\alpha_1$, $\alpha_2$, $\alpha_3$: Parâmetros utilizados para incrementar os valores do vetor de resultados em cada iteração do ALNS. São utilizados para atualização dos pesos dos métodos;
- $\rho$: Fator de reação que controla a rapidez com a qual o algoritmo de ajuste de peso reage a mudanças dos resultados, neste trabalho foi utilizado o valor de $\rho = 0.1$ como em [Pisinger & Ropke, 2010];
- $x$: utilizado para o cálculo da temperatura
- *batchLimit*: Número de lotes que serão retirados durante a etapa da busca local.

Os valores testados para cada parâmetro são apresentados na Tabela 4.3. Observa-se que há um total de 729 combinações de parâmetros da heurística ALNS geradas utilizando o método *full factorial DOE*.

Tabela 4.3: Conjunto de valores de parâmetros testados para a calibração do ALNS

| **Parâmetros** | **Valores Testados** | **Quantidade** |
|---|---|---|
| $nMax$ | {50, 100, 150} | 3 |
| $taskLimit$ | {15%, 25%, 35%} | 3 |
| $\rho$ | {0.1} | 1 |
| $\alpha_1$ | {20, 30, 40} | 3 |
| $\alpha_2$ | {10, 15, 20} | 3 |
| $\alpha_3$ | {5, 10, 15} | 3 |
| $x$ | {10, 100, 1000} | 3 |
| $batchLimit$ | {30%} | 1 |
| Total | | 729 |

Para os testes, as três principais hipóteses da ANOVA, normalidade, igualdade de variância e independência dos resíduos, foram verificadas. A Figura 4.1 mostra as médias e intervalos resultantes do teste *Tukey* da diferença Honestamente Significativa (HSD) com nível de confiança de 95% obtidas na execução de um teste de Análise de Variância (ANOVA) para as diferentes combinações dos parâmetros da heurística ALNS. Para facilitar a visualização dos resultados são apresentados os resultados de 49 configurações. Pode-se observar que existe diferença significativa entre as configurações, pois há intervalos que não se sobrepõem.

Na Figura, 4.1, apesar das melhores configurações não apresentarem diferenças estatisticamente significativas, a configuração que apresentou menor média de

RPD foi encontrada pela configuração de número 21. Os valores dos parâmetros da configuração 21 são: $nMax = 50$, $taskLimit = 25\%$, $batchLimit = 30\%$, $\rho = 0.1$, $\alpha_1 = 30$, $\alpha_2 = 10$, $\alpha_3 = 15$ e $x = 100$.

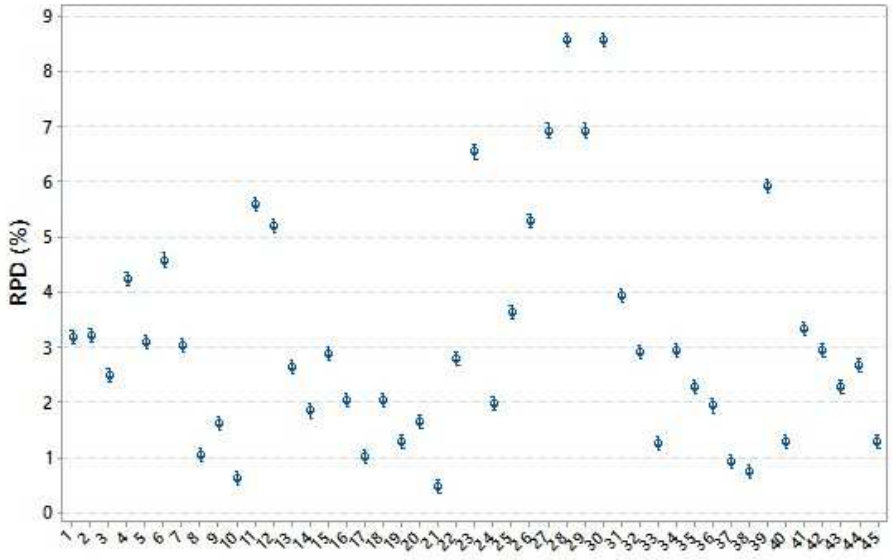



Figura 4.1: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para calibração do ALNS

### 4.3.2 Calibração dos Parâmetros do IG

Foram realizados experimentos para determinar valores adequados para os seguintes parâmetros da heurística IG apresentada neste trabalho:

- *jobLimit*: Nível máximo de destruição utilizado na fase de Destruição-Construção;
- $x$: Utilizado no cálculo do valor da temperatura para verificar o critério de aceitação para soluções de pior qualidade;
- *batchLimit*: Parâmetro que representa o número de lotes que serão retirados da solução na etapa de busca local.

Os parâmetros foram combinados gerando um total de 27 configurações possíveis para o algoritmo IG. Os valores testados para cada parâmetro são apresentados a seguir:

A Figura 4.2 ilustra as médias e os intervalos HSD de Tukey com nível de confiança de 95% obtidas na execução de um teste de Análise de Variância (ANOVA)

Tabela 4.4: Conjunto de valores de parâmetros testados para a calibração do ALNS

| **Parâmetros** | **Valores Testados** | **Quantidade** |
|---|---|---|
| *jobLimit* | {20%, 25%, 30%} | 3 |
| $x$ | {10, 100, 1000} | 3 |
| *batchLimit* | {15%, 20%, 25%} | 3 |
| Total | | 27 |

para as diferentes combinações dos parâmetros $d_{max}$, $T$ e *batchLimit*. Pode-se observar que existe diferença significativa entre as configurações, pois há intervalos que não se sobrepõem.

Na Figura 4.2, apesar das melhores configurações não apresentarem diferenças estatisticamente significativas, a configuração que apresentou menor média de RPD foi encontrada pela configuração de número 4, com os seguintes valores de configuração $jobLimit = 25\%$, $x = 1000$ e $batchLimit = 25\%$ .

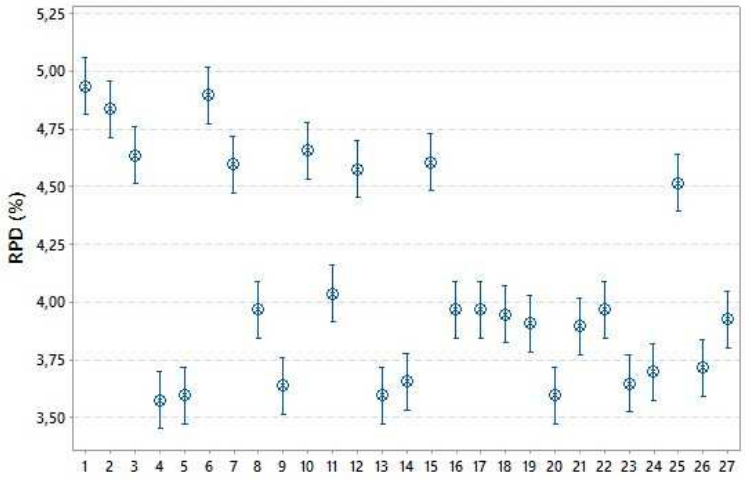



Figura 4.2: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para calibração do IG

### 4.3.3 Calibração dos Parâmetros do SA

A metaheurística SA utiliza os seguintes parâmetros:

- $T_0$: Valor da temperatura;
- $\alpha$: Fator de resfriamento;
- $nMax$: Número de iterações;
- *batchLimit*: utilizado na busca local.

Esses parâmetros foram combinados para realizar a calibração de parâmetros. No total 81 configurações foram geradas. Os valores testados para cada parâmetro são mostrados na Tabela 4.5:

Tabela 4.5: Conjunto de valores de parâmetros testados para a calibração do SA

| **Parâmetros** | **Valores Testados** | **Quantidade** |
|---|---|---|
| $T_0$ | {10 000, 20 000, 30 000} | 3 |
| $\alpha$ | {0.3, 0.4, 0.5} | 3 |
| $nMax$ | { n/10, n/20, n/30} | 3 |
| $batchLimit$ | {25%, 25%, 30%} | 3 |
| Total | | 81 |

A Figura 4.3 ilustra as médias e os intervalos HSD de Tukey com nível de confiança de 95% obtidas na execução de um teste de Análise de Variância (ANOVA) para as diferentes combinações dos parâmetros $T_0$, $\alpha$, $nMax$ e $batchLimit$. Pode-se observar na Figura 4.3 que não existe diferenças significativas entre as menores médias dos parâmetros, no entanto, a combinação que gerou o menor RPD médio, foi a combinação de número 27. Após a calibração os seguintes parâmetros são utilizados no algoritmo SA: $T_0 = 20000$, $\alpha = 0.4$, $nMax = \text{n}/30$ e $batchLimit = 30\%$.

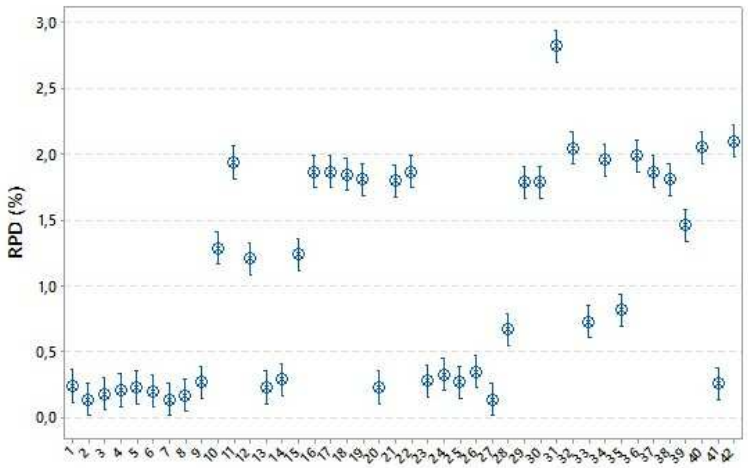



Figura 4.3: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para calibração do SA

## 4.4 Implementação das Heurísticas da Literatura

Algoritmos Meméticos (MAs) consistem em métodos de busca heurística aplicados a problemas de otimização, gerados pela combinação de algoritmos populacionais

(ex.: Algoritmos Genéticos), com algoritmos de busca local, como *Hill Climbling*.

Os GAs constituem uma classe particular de algoritmos que utilizam técnicas inspiradas na biologia evolutiva, como por exemplo, mutação, seleção natural e recombinação. A fase inicial de um GA gera uma população inicial de cromossomos. A representação de cada um destes é geralmente referenciada como um indivíduo, enquanto um grupo de indivíduos representa uma população. Após a geração da população inicial, avalia-se a aptidão dos indivíduos, isto é, a qualidade de cada solução. Essa avaliação ocorre durante o processo evolutivo. Alguns indivíduos são selecionados e através de operadores como mutação e crossover geram-se novos indivíduos para a próxima geração. Em geral, os indivíduos mais aptos são selecionados e os menos aptos descartados. Espera-se que após um número de gerações, uma solução satisfatória seja encontrada [de Lacerda & de Carvalho, 1999].

O algoritmo MA proposto por [Chiang et al., 2010] utiliza uma codificação de cromossomos capaz de considerar e representar a formação de lotes e o sequenciamento de lotes simultaneamente, isto é, um cromossomo representa uma sequência de lotes, na qual, em cada lote, há no mínimo uma tarefa e no máximo $b_{max}$ tarefas. Assim, para representar um sequenciamento, os lotes são colocados no cromossomo da esquerda para direita em ordem crescente do tempo de conclusão do lote anterior a ele no sequenciamento. Distintos cromossomos representam diferentes sequências de lotes e/ou diferentes formações de lotes. Esse esquema de representação utiliza operadores de mutação de codificação eficazes [Chiang et al., 2010].

No trabalho de [Chiang et al., 2010] o tamanho da população foi definido como 100 e o número de gerações como 1000. O critério de parada utilizado pelo MA é o máximo de gerações sem melhora, esse valor foi fixado em 50 gerações sem melhora. A média de tempo computacional gasto foi de 3.9 segundos.

VNS consiste em uma meta-heurística baseada em busca local que explora a ideia de mudança da estrutura da vizinhança, ou seja, tem por base a pesquisa em vizinhanças diferentes. A principal ideia da meta-heurística VNS é escapar de ótimos locais, isto é realizado através da exploração do espaço de soluções através de trocas sistemáticas de estruturas de vizinhança [Mladenović & Hansen, 1997]. Para escapar de ótimos locais a busca é reiniciada de uma vizinhança escolhida aleatoriamente. Este passo de reinício é chamado de *shaking* e é realizado utilizando diferentes vizinhanças de tamanhos crescente.

O algoritmo VNS proposto por [Bilyk et al., 2014] utiliza o algoritmo TWD para encontrar uma solução inicial. Durante as iterações do VNS as instâncias são submetidas a um processo de perturbação, no qual há um conjunto de cinco vizinhanças a serem percorridas. Estas vizinhanças consistem basicamente na mo-

vimentação de lotes. Após o processo de perturbação uma busca local é realizada. A busca local é composta por vizinhanças geradas pela troca de tarefas e lotes e também pela inserção de lotes. No critério de aceitação se a solução corrente é melhorada após o processo de perturbação e busca local, altera-se a solução corrente e a busca na vizinhança é inicializada, isto é, o processo de busca pelas vizinhanças retorna a vizinhança inicial. Caso contrário a busca continua na próxima vizinhança, caso todas as vizinhanças já tenham sido visitadas, a busca inicia-se da primeira vizinhança. O critério de parada utilizado pelo VNS foi baseado no tempo em segundos.

Os autores compararam o desempenho do algoritmo VNS proposto com os melhores resultados obtidos pelo MA proposto em [Chiang et al., 2010]. O algoritmo VNS é executado utilizando as 4860 instâncias utilizadas pelo MA. No algoritmo VNS cada instância é executada 10 vezes e o menor valor da função objetivo é utilizado para determinar o desempenho da heurística. Resultados apresentados demonstram que para o critério de parada de 60 segundos o algoritmo VNS supera o algoritmo MA em 10.28%.

Como um dos objetivos deste trabalho é verificar a utilização de diferentes heurísticas que possam obter melhores resultados para o problema em estudo, se faz necessária a comparação com os melhores resultados obtidos na literatura. Para que essa comparação seja realizada de forma justa todos os algoritmos comparados devem ser executados na mesma máquina, portanto, foi realizada a reimplementação dos algoritmos MA e VNS seguindo os detalhes fornecidos nos artigos de [Chiang et al., 2010] e [Bilyk et al., 2014], respectivamente.

Apenas os resultados obtidos pelo algoritmo MA foram disponibilizados, por esse motivo, embora ambas heurísticas tenham sido reimplementadas, nesta seção apenas será feita a comparação entre MA proposto por [Chiang et al., 2010] e o algoritmo memético reimplementado neste trabalho que será denotado por $MA^*$. O algoritmo $MA^*$ foi reimplementado seguindo os passos do algoritmo original descrito no artigo [Chiang et al., 2010]. Este algoritmo foi implementado e executado na mesma máquina utilizada para os testes deste trabalho utilizando diferentes critérios de parada. O primeiro critério é o mesmo utilizado no artigo [Chiang et al., 2010], o número de gerações sem melhora. Utiliza-se também os seguintes critérios de paradas baseados em tempo de CPU, são eles:$0.05 \times n$, $0.1 \times n$ e $0.2 \times n$ segundos, onde $n$ representa o número de tarefas. Para esses critérios denota-se o algoritmo $MA^*$ por $MA_1$, $MA_2$ e $MA_3$ respectivamente.

Para verificar a performance dos algoritmos reimplementados uma comparação entre os algoritmos MA, $MA^*$, $MA_1$, $MA_2$ e $MA_3$ é realizada. A Figura 4.4 mostra

o gráfico de médias resultantes do teste HSD de *Tukey* com nível de confiança de 95%. Na Figura 4.4 é possível observar que há diferença significativa entre o $MA$ e o $MA^*$ reimplementado. É possível observar ainda que os resultados obtidos pelas implementações dos algoritmos $MA_1$, $MA_2$ e $MA_3$ são estatisticamente melhores que o resultado apresentado pelo algoritmo MA. Embora o algoritmo $MA_3$ apresenta a menor média, não há diferença significativa entre esses três algoritmos.

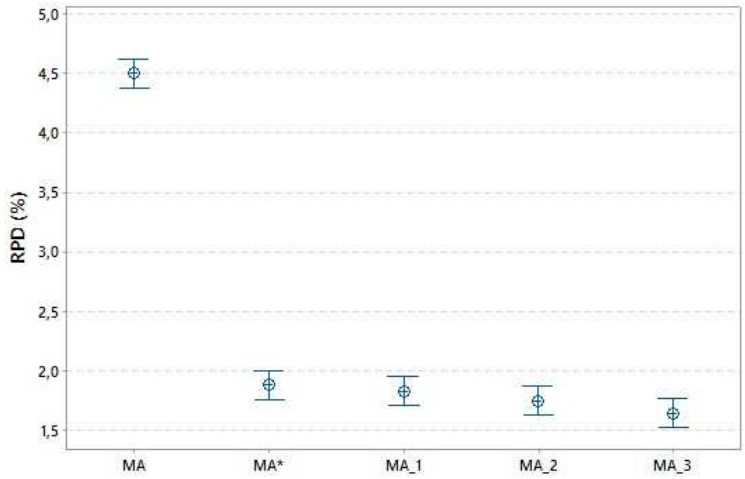



Figura 4.4: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para os algoritmos MA, $MA^*$, $MA_1$, $MA_2$, $MA_3$

A Figura 4.5 mostra o gráfico de médias resultantes do teste HSD de *Tukey* com nível de confiança de 95% realizado para comparar os resultados obtidos pelo $MA^*$ reimplementado com as versões implementadas dos algoritmos $MA_1$, $MA_2$ e $MA_3$. Pode se observar que o algoritmo $MA_3$ apresenta a menor média entre os algoritmos.

É importante salientar que apesar da reimplementação do algoritmo $MA^*$ ter seguido os passos originais do algoritmo proposto por [Chiang et al., 2010], existe a diferença de hardware utilizado para realização dos testes e portanto a diferença entre o desempenho dos algoritmos é compreensível.

## 4.5 Experimento 1: Teste com Instâncias de Pequeno Porte no *solver ILOG/CPLEX*

O primeiro experimento realizado utiliza um conjunto de 288 instâncias de pequeno porte, geradas conforme a Tabela 4.1. Esse experimento tem como intuito a execução das instâncias no *solver* ILOG/CPLEX a fim de encontrar a solução ótima de

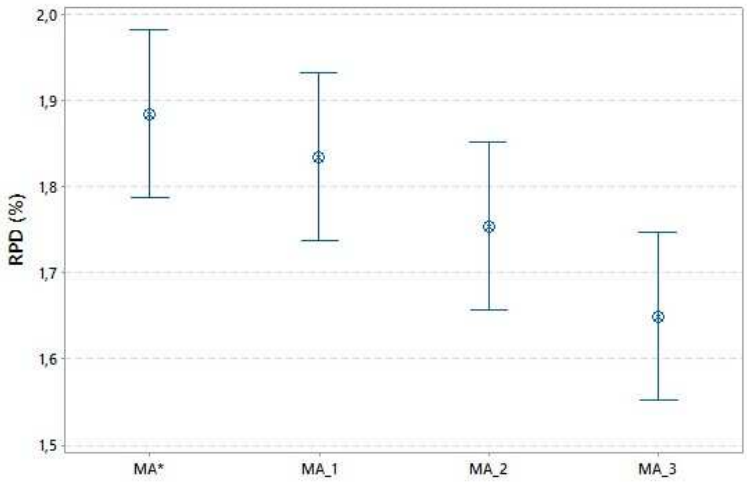



Figura 4.5: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para os algoritmos MA, $MA^*$, $MA_1$, $MA_2$, $MA_3$

algumas instâncias para posterior comparação com os algoritmos propostos: ALNS, IG e SA, como os algoritmos reimplementados VNS e $MA^*$. O modelo matemático foi implementado utilizando a biblioteca ILOG Concert, por meio da linguagem C++. A resolução do modelo ocorreu através do software IBM CPLEX versão 12.6. O modelo foi executado uma única vez para cada instância, com tempo limite de 3600 segundos.

A Figura 4.6 ilustra as médias e os intervalos HSD de Tukey com nível de confiança de 95%, obtidas na execução de um teste de Análise de Variância (ANOVA) para a comparação entre os resultados da média de RPD(%) das heurísticas e do modelo matemático. Pode-se observar na Figura 4.6 que existe diferenças significativas entre as heurísticas propostas e os resultados obtidos pelo modelo matemático executado no *solver* CPLEX. No entanto, entre as heurísticas executadas não houve diferenças significativas sendo que a heurística SA obteve a menor média de RPD.

A Tabela 4.6 apresenta as instâncias, para as quais o modelo matemático conseguiu encontrar soluções ótimas. Pode-se observar que do conjunto de 288 instâncias, apenas para 6 instâncias foi possível obter o valor ótimo. As instâncias que contêm um número de tarefas elevado são mais difíceis de serem resolvidas, uma vez que o *solver* apenas encontrou o valor ótimo para algumas instâncias com número de tarefas igual 20.

O segundo experimento tem como objetivo verificar se a utilização de soluções iniciais viáveis ajudaria o modelo a encontrar soluções em um tempo computacional menor ou até mesmo encontrar soluções ótimas. Os limitantes superiores do modelo são obtidos pelo algoritmo SA proposto, uma vez que foi a heurística que apresentou

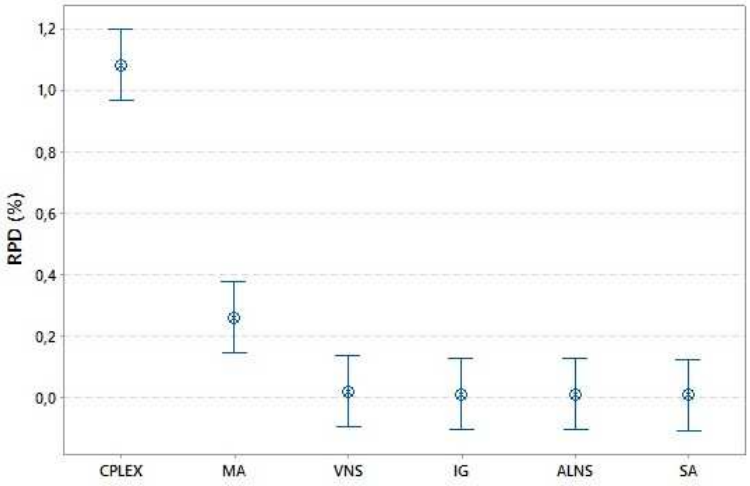



Figura 4.6: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação de RPDs (%) entre as heurísticas e o modelo matemático

Tabela 4.6: Valores ótimos da execução do modelo matemático para o conjunto de 288 instâncias

| **Número de Tarefas** | **Ótimos Encontrados** |
|---|---|
| 20 | 6 |
| 40 | 0 |
| 60 | 0 |
| 80 | 0 |

menor valor de média entre as heurísticas. Para tanto, executamos o algoritmo SA foi executado durante $0.1 \times n$ segundos.

Os resultados apresentados demonstram que a utilização de uma solução inicial produzida pelo algoritmo SA pelo *solver* é capaz de encontrar o valor ótimo para 15 instâncias com número de tarefas igual a 20. Analisando os resultados obtidos após a execução do modelo, é possível observar que em 86/288 (29,86%) instâncias, o ILOG/CPLEX conseguiu melhorar o limitante superior gerado pelo SA, enquanto em 202/288 (70,14%) este solver não reduziu o limite superior das instâncias testadas. A fim de verificar a diferença entre as soluções fornecidas pelo SA e as soluções encontradas pelo método exato, foi calculado o desvio relativo, dado pela Equação 4.3. Através desta medida, calculamos o quanto a solução fornecida pelo SA ($f_{SA}$) difere da solução encontrada pelo método exato ($f_{CPLEX}$). Os resultados são apresentados na Tabela 4.7 e indicam que a solução encontrada pela heurística SA difere muito pouco da melhor solução encontrada pelo ILOG/CPLEX, em uma hora de execução, com desvio médio de aproximadamente 2,98%.

$$\frac{f_{\text{SA}} - f_{\text{CPLEX}}}{f_{\text{CPLEX}}} \times 100, \tag{4.3}$$

Tabela 4.7: Desvio médio entre o limitante superior fornecido ao CPLEX em relação a melhor solução encontrada pelo CPLEX

| **n** | **Desvio Médio** |
|---|---|
| 20 | 6,78 |
| 40 | 1,71 |
| 60 | 0,67 |
| 80 | 0,36 |
| Média | 2,38 |

## 4.6 Experimento 2: Comparações entre métodos heurísticos

Nesta seção inicialmente é realizada uma comparação entre o desempenho dos algoritmos reimplemtados $MA^*$ e VNS com os algoritmos propostos neste trabalho: ALNS, IG e SA, para as instâncias de grande porte (instâncias da literatura). Em seguida é realizada uma comparação entre as heurísticas ALNS, IG e SA propostas para o problema em estudo. O critério de parada utilizado pelas heurísticas implementadas neste trabalho é baseado na quantidade de tempo de CPU, sendo eles: $numJobs \times 0.05$, $numJobs \times 0.1$ e $numJobs \times 0.2$ segundos. A utilização de diferentes critérios de parada consiste em verificar a convergência dos algoritmos analisados. Para resolver cada instância, cada algoritmo foi executado 10 vezes. Como em [Bilyk et al., 2014], para análise dos resultados, as instâncias são agrupadas em função dos parâmetros do problema.

No primeiro experimento, as heurísticas propostas neste trabalho, ALNS, IG e SA, assim como os algoritmos VNS e o algoritmo $MA^*$ reimplementados são comparados ao algoritmo $MA$ [Chiang et al., 2010]. É verificada a melhoria de cada heurística em relação ao algoritmo $MA$. Utilizando a métrica RPI, (Equação 4.2). O algoritmo $MA^*$ é executado com três diferentes critérios de parada, são eles: $0.05 \times numJobs$, $0.1 \times numJobs$ e $0.2 \times numJobs$, denotando-se assim para cada critério de parada a nomenclatura $MA_1$, $MA_2$ e $MA_3$, respectivamente. As Tabelas 4.8, 4.9 e 4.10 mostram os valores dos RPIs médios dos algoritmos para cada categoria de instância, considerando o critério de parada igual a $0.05 \times numJobs$, $0.1 \times numJobs$ e $0.2 \times numJobs$, respectivamente. O RPI representa a porcentagem

de melhoria obtida pelas heurísticas em relação ao algoritmo $MA$ da literatura. Os maiores percentuais de melhoria estão em negrito.

Tabela 4.8: Porcentagem de melhoria das heurísticas em relação ao algoritmo MA considerando o critério de parada = $0.05 \times n$

| **Fator** | **Nível** | $MA_1$ | **VNS** | **IG** | **ALNS** | **SA** |
|---|---|---|---|---|---|---|
| Número de Famílias | 3 | 6,38 | 10,23 | 13,39 | 16,83 | **21,93** |
| | 6 | 2,86 | 4,86 | 6,60 | 9,16 | **12,18** |
| | 12 | 2,25 | 3,61 | 4,36 | 5,70 | **9,13** |
| Número de Máquinas | 3 | 4,14 | 6,56 | 8,79 | 11,39 | **15,51** |
| | 4 | 3,71 | 6,09 | 7,97 | 10,40 | **14,32** |
| | 5 | 3,64 | 6,05 | 7,55 | 9,91 | **13,41** |
| Tamanho Máximo do Lote | 4 | 3,56 | 6,15 | 8,52 | 11,65 | **17,47** |
| | 8 | 4,10 | 6,31 | 7,68 | 9,48 | **11,36** |
| Tempo de Chegada | Próximo de zero | 3,62 | 5,19 | 5,85 | 6,97 | **10,35** |
| | Moderado | 4,03 | 6,39 | 7,92 | 10,03 | **14,19** |
| | Alto | 3,84 | 7,12 | 10,53 | 14,68 | **18,70** |
| Data de Entrega | Próximo de zero | 2,06 | 3,25 | 3,88 | 4,76 | **6,63** |
| | Moderado | 3,83 | 5,76 | 7,11 | 8,83 | **12,24** |
| | Alto | 5,60 | 9,68 | 13,33 | 18,10 | **24,36** |
| **Média** | | 3,83 | 6,23 | 8,10 | 10,56 | **14,41** |

Tabela 4.9: Porcentagem de melhoria das heurísticas em relação ao a algoritmo MA considerando o critério de parada = $0.1 \times n$

| **Fator** | **Nível** | $MA_2$ | **VNS** | **IG** | **ALNS** | **SA** |
|---|---|---|---|---|---|---|
| Número de Famílias | 3 | 8,09 | 13,65 | 15,01 | 20,03 | **23,66** |
| | 6 | 4,40 | 9,21 | 10,34 | 13,01 | **15,83** |
| | 12 | 3,79 | 6,85 | 8,30 | 9,70 | **12,61** |
| Número de Máquinas | 3 | 5,87 | 10,43 | 13,05 | 15,47 | **18,82** |
| | 4 | 5,32 | 9,67 | 10,79 | 13,81 | **17,16** |
| | 5 | 5,09 | 9,61 | 9,83 | 13,46 | **16,12** |
| Tamanho Máximo do Lotes | 4 | 5,97 | 10,44 | 13,08 | 16,46 | **20,94** |
| | 8 | 4,89 | 9,37 | 9,36 | 12,03 | **13,80** |
| Tempo de Chegada | Próximo de zero | 4,71 | 7,66 | 7,90 | 9,46 | **11,81** |
| | Moderado | 5,88 | 9,81 | 11,21 | 13,94 | **17,48** |
| | Alto | 5,69 | 12,24 | 14,54 | 19,32 | **22,82** |
| Data de Entrega | Próximo de zero | 2,97 | 4,91 | 6,60 | 6,97 | **8,64** |
| | Moderado | 5,43 | 8,71 | 11,12 | 12,63 | **15,29** |
| | Alto | 7,89 | 16,08 | 15,94 | 23,13 | **28,18** |
| **Média** | | 5,43 | 9,90 | 11,22 | 14,25 | **17,37** |

Tabela 4.10: Porcentagem de melhoria das heurísticas em relação ao algoritmo MA considerando o critério de parada = $0.2 \times n$

| **Fator** | **Nível** | $MA_3$ | **VNS** | **IG** | **ALNS** | **SA** |
|---|---|---|---|---|---|---|
| Número de Famílias | 3 | 8,94 | 21,85 | 22,39 | 26,84 | **24.96** |
| | 6 | 5,01 | 12,90 | 13,58 | **16,35** | 15,84 |
| | 12 | 3,96 | 8,02 | 9,10 | 9,74 | **13,36** |
| Número de Máquinas | 3 | 6,85 | 15,37 | 16,40 | 19,19 | **19,54** |
| | 4 | 5,90 | 13,99 | 14,79 | 17,37 | **17,91** |
| | 5 | 5,15 | 13,39 | 13,87 | 16,38 | **16,72** |
| Tamanho Máximo do Lotes | 4 | 6,98 | 16,83 | 17,31 | 21,10 | **21,35** |
| | 8 | 4,96 | 11,68 | 12,72 | 14,19 | **14,75** |
| Tempo de Chegada | Próximo de zero | 5,23 | 9,15 | 9,96 | 10,41 | **12,13** |
| | Moderado | 6,38 | 13,67 | 13.76 | 16,47 | **17,56** |
| | Alto | 6,29 | 19,94 | 21,32 | **26,06** | 24,47 |
| Data de Entrega | Próximo de zero | 3,69 | 5,90 | 8.20 | 7,15 | **8,89** |
| | Moderado | 5,81 | 11,50 | 11.43 | 13,96 | **15,93** |
| | Alto | 8,41 | 25,36 | 25.41 | **31,83** | 29,33 |
| **Média** | | 5,97 | 14,25 | 15,02 | 17,65 | **18,05** |

Observa-se que a heurística SA se sobressaiu para todas as instâncias. É possível observar também que o algoritmo ALNS apresentou bons resultados seguido do algoritmo IG. Entre as heurísticas analisadas, o algoritmo $MA_1$ obteve os menores valores de média de RPI. Analisando os resultados conclui-se que o algoritmo SA converge mais rapidamente que as outras heurísticas. Isto pode ser observado pelos dados apresentados nas Tabelas 4.9 e 4.10, nas quais para o tempo de $0.1 \times numJobs$, a heurística SA obteve uma melhora de 17.37% em relação ao algoritmo MA. Por outro lado, para o critério de parada de $0.2 \times numJobs$, a melhora alcançada foi de 18.05%.

Os resultados apresentados nas Tabelas 4.8, 4.9 e 4.10 podem ser melhor analisados nas Figuras 4.7, 4.8 e 4.9. Estas Figuras ilustram as médias e os intervalos HSD de Tukey com nível de confiança de 95% obtidas na execução de um teste de Análise de Variância. Pode-se notar observando as Figuras 4.7 e 4.8 que os resultados apresentados são significativamente diferentes e que a heurística SA obteve os melhores resultados, isto é, apresentou uma maior melhoria em relação ao algoritmo MA.

Na Figura 4.9, os resultados apresentados demonstram que não há diferenças significativas entre as heurísticas SA e ALNS, assim como entre as heurísticas IG e VNS. No entanto, há diferenças significativas entre esses grupos de heurísticas. O maior valor de RPI foi obtido pela heurística SA.

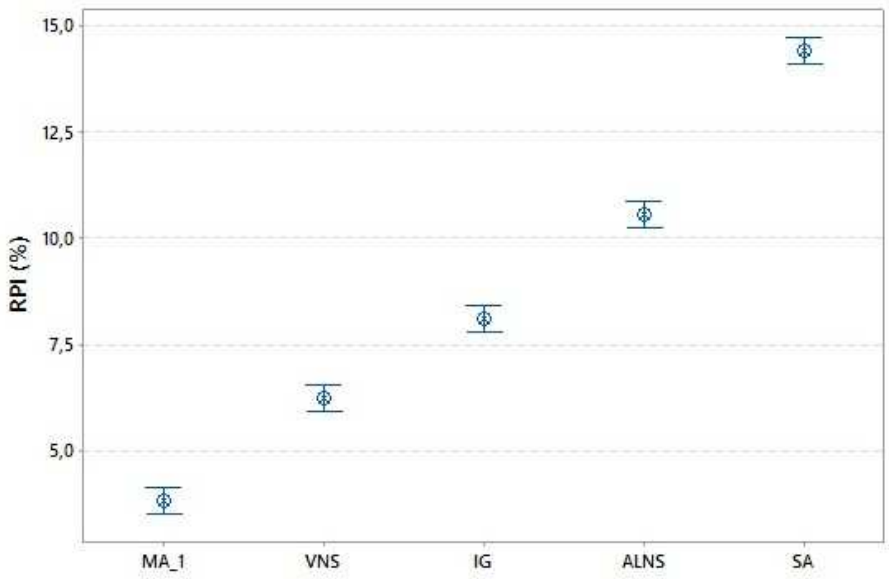



Figura 4.7: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do RPI (%) das heurísticas para o critério de parada = $0.05 \times n$


Figura 4.8: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do RPI (%) das heurísticas para o critério de parada = $0.1 \times n$

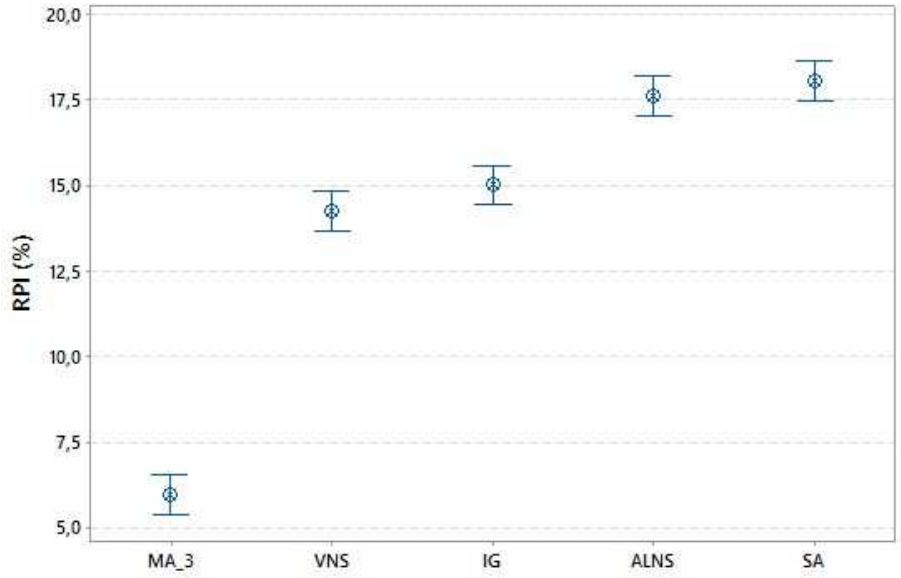



Figura 4.9: Gráfico de médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do RPI (%) das heurísticas para o critério de parada = $0.2 \times n$

O segundo experimento realiza uma comparação entre as heurísticas ALNS, IG e SA propostas para o problema em estudo. As médias de RPDs, Equação 4.1, são agrupadas em função dos principais parâmetros do problema. O intuito deste experimento consiste em verificar o desempenho das heurísticas propostas para o problema. As Tabelas 4.11, 4.12 e 4.13 mostram os valores dos RPDs médios dos algoritmos ALNS, IG e SA, para cada categoria de instância considerando o critério de parada igual a $0.05 \times numJobs$, $0.1 \times numJobs$ e $0.2 \times numJobs$, respectivamente. As melhores médias estão em negrito.

Tabela 4.11: Valores médios de RPDs para as heurísticas ALNS, IG e SA para o critério de parada = $0.05 \times n$

| **Fator** | **Nível** | **IG** | **ALNS** | **SA** |
|---|---|---|---|---|
| Número de Famílias | 3 | 7,19 | 5,01 | **0,38** |
| | 6 | 12,93 | 7,94 | **0,25** |
| | 12 | 9,01 | 4,96 | **0,49** |
| Número de Máquinas | 3 | 10,93 | 6,95 | **0,48** |
| | 4 | 9,77 | 5,90 | **0,30** |
| | 5 | 8,42 | 5,05 | **0,34** |
| Tamanho Máximo do Lote | 4 | 13,97 | 8,98 | **0,28** |
| | 8 | 5,45 | 2,96 | **0,47** |
| Tempo de Chegada | Próximo de zero | 5,73 | 4,23 | **0,11** |
| | Moderado | 9,29 | 6,38 | **0,26** |
| | Alto | 14,10 | 7,29 | **0,74** |
| Data de Entrega | Próximo de zero | 3,81 | 2,69 | **0,33** |
| | Moderado | 6,67 | 4,81 | **0,36** |
| | Alto | 18,64 | 10,41 | **0,42** |
| **Média** | | 9,71 | 5,97 | **0,37** |

Tabela 4.12: Valores médios de RPDs para as heurísticas ALNS, IG e SA para o critério de parada = $0.1 \times n$

| **Fator** | **Nível** | **IG** | **ALNS** | **SA** |
|---|---|---|---|---|
| Número de Famílias | 3 | 5,18 | 2,45 | **0,49** |
| | 6 | 15,89 | 5,96 | **1,07** |
| | 12 | 5,12 | 2,58 | **0,54** |
| Número de Máquinas | 3 | 5,69 | 2,92 | **0,74** |
| | 4 | 10,88 | 4,21 | **0,89** |
| | 5 | 9,61 | 3,84 | **0,47** |
| Tamanho Máximo do Lotes | 4 | 11,19 | 5,63 | **0,76** |
| | 8 | 6,27 | 1,68 | **0,64** |
| Tempo de Chegada | Próximo de zero | 6,88 | 3,29 | **1,32** |
| | Moderado | 8,38 | 3,17 | **0,48** |
| | Alto | 10,93 | 4,52 | **0,30** |
| Data de Entrega | Próximo de zero | 3,03 | **0,62** | 0,89 |
| | Moderado | 8,60 | 4,05 | **0,62** |
| | Alto | 14,57 | 6,31 | **0,58** |
| **Média** | | 8,73 | 3,66 | **0,70** |

Tabela 4.13: Valores médios de RPDs para as heurísticas ALNS, IG e SA para o critério de parada = $0.2 \times n$

| **Fator** | **Nível** | **IG** | **ALNS** | **SA** |
|---|---|---|---|---|
| Número de Famílias | 3 | 5,38 | 3,03 | **0,99** |
| | 6 | 8,17 | **2,48** | 2,82 |
| | 12 | 5,79 | **1,67** | 2,64 |
| Número de Máquinas | 3 | 7,04 | 2,69 | **2,49** |
| | 4 | 6,48 | 2,45 | **2,06** |
| | 5 | 5,82 | 2,05 | **1,90** |
| Tamanho Máximo do Lotes | 4 | 8,86 | 3,48 | **2,22** |
| | 8 | 4,03 | **1,30** | 2,07 |
| Tempo de Chegada | Próximo de zero | 4,07 | 2,56 | **0,49** |
| | Moderado | 6,68 | 3,01 | **1,78** |
| | Alto | 8,58 | **1,61** | 4,17 |
| Data de Entrega | Próximo de zero | 2,96 | 1,80 | **0,88** |
| | Moderado | 5,56 | 2,68 | **1,75** |
| | Alto | 10,81 | **2,71** | 3,81 |
| **Média** | | 6,45 | 2,39 | **2,15** |

De forma similar a análise realizada no primeiro experimento, a fim de validar os resultados obtidos pelos algoritmos e verificar se as diferenças observadas são estatisticamente significantes é realizada a análise dos resultados da ANOVA. As Figuras 4.10, 4.11 e 4.12 apresentam os gráficos de médias resultantes do teste HSD de Tukey com nível de confiança de 95% do resultado da ANOVA destes algoritmos.

Na Figura 4.10 é possível observar que há diferenças significativas entre as heurísticas propostas. Analisando a média total de RPD, é possível verificar também que o algoritmo SA obteve o menor média de RPD. Nas Figuras 4.11 e 4.12 não foi observado diferenças significativas entre as heurísticas ALNS e SA, embora exista diferença entre essas heurística e a heurística IG. Analisando os resultados apresentados, pode-se concluir que o algoritmo SA obteve o melhor desempenho entre os algoritmos testados.

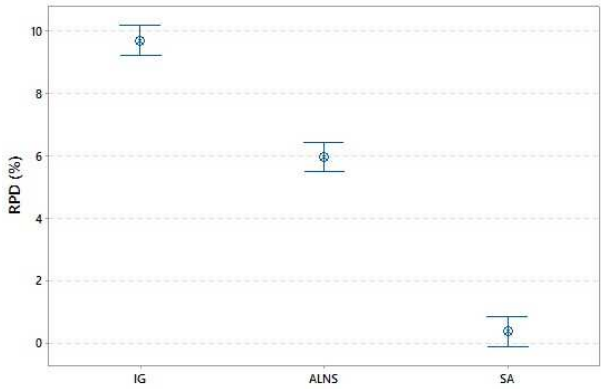



Figura 4.10: Gráfico de médias e itervalos HSD de Tukey com nível de confiança de 95% para comparação da heurísticas ALNS, IG e SA para o critério de parada = $0.05 \times n$

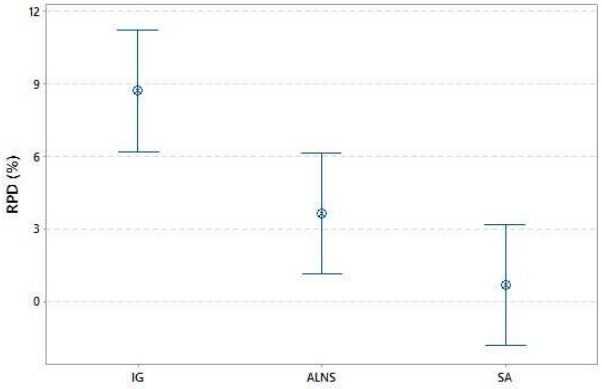



Figura 4.11: Gráfico de médias e itervalos HSD de Tukey com nível de confiança de 95% para comparação da heurísticas ALNS, IG e SA para o critério de parada = $0.1 \times n$

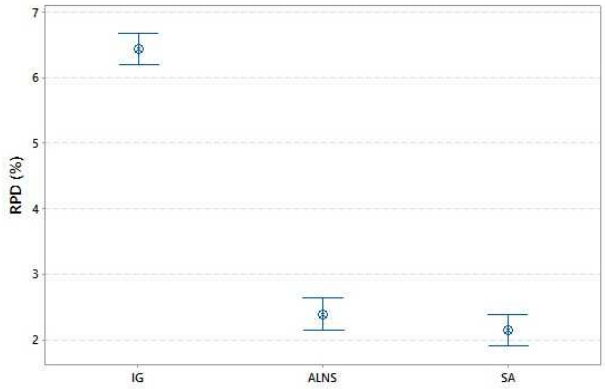



Figura 4.12: Gráfico de médias e itervalos HSD de Tukey com nível de confiança de 95% para comparação da heurísticas ALNS, IG e SA para o critério de parada = $0.2 \times n$

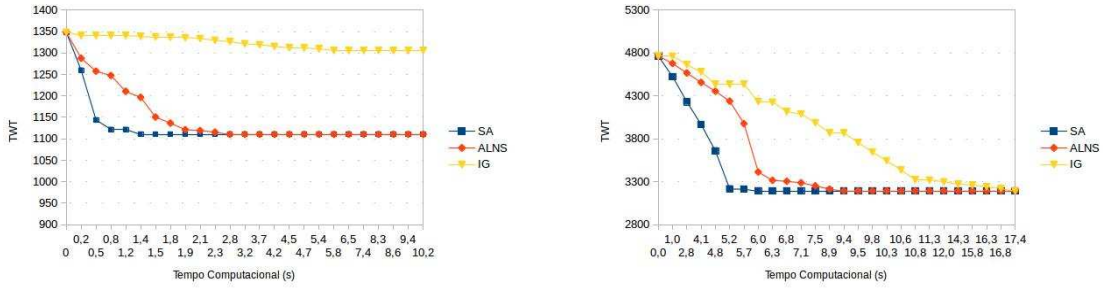
## 4.7 Análise de Convergência das Heurísticas

Os experimentos descritos nas Seção 4.6 foram realizados focando na qualidade das soluções em relação ao valor da função objetivo. No entanto, é importante verificar a qualidade das heurísticas em relação ao tempo de execução. Por essa razão seis instâncias com diferentes valores de parâmetros foram selecionadas. As principais características das instâncias utilizadas, número de tarefas($n$), máquinas($m$), capacidade do lote($B$) e número de famílias($f$), são descritas na Tabela 4.14.

Tabela 4.14: Características das instâncias

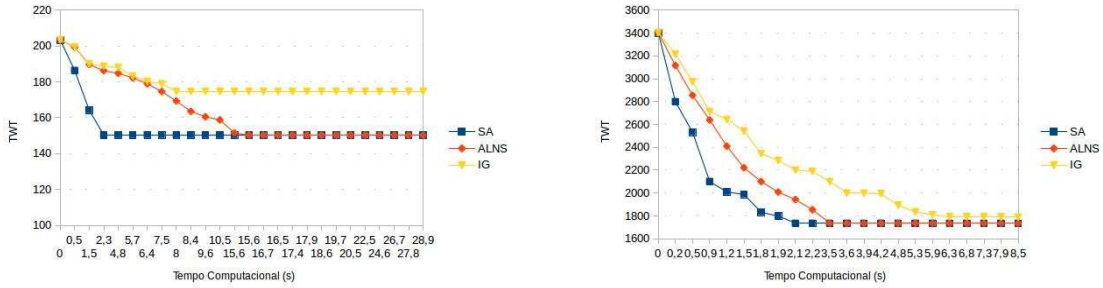
| **Instância** | **n** | **m** | **B** | **F** |
|---|---|---|---|---|
| 1 | 180 | 4 | 4 | 3 |
| 2 | 180 | 5 | 5 | 3 |
| 3 | 240 | 4 | 4 | 6 |
| 4 | 240 | 5 | 5 | 6 |
| 5 | 300 | 4 | 4 | 12 |
| 6 | 300 | 5 | 5 | 12 |

O experimento consiste em fixar um valor de tempo e executar os algoritmos a partir do valor inicial encontrado, utilizando a heurística TWD. Assim, em cada melhora obtida por uma heurística o resultado e o tempo gasto são armazenados para realizar a análise da convergência das heurísticas em função do tempo. Analisando as Figuras 4.13, 4.14 e 4.15, pode-se observar que entre as heurísticas propostas, a heurística SA converge mais rapidamente para a melhor solução. Observa-se também, assim como o SA, que em algumas instâncias a heurística ALNS também converge rapidamente.



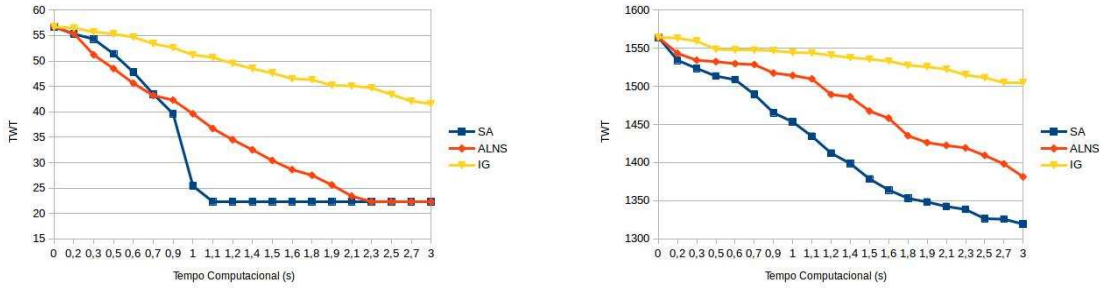

(a) Inst1-F3-M4-N180-B4-0.75-0.75 (b) Inst2-F3-M5-N180-B4-0.75-0.75

Figura 4.13: Convergência das soluções ao decorrer do tempo usando os algoritmos ALNS, IG e SA




(a) Inst3-F6-M4-N240-B4-0.75-0.75 (b) Inst4-F6-M5-N240-B4-0.75-0.75

Figura 4.14: Convergência das soluções ao decorrer do tempo usando os algoritmos ALNS, IG e SA




(a) Inst5-F12-M4-N300-B4-0.75-0.75 (b) Inst6-F12-M5-N300-B4-0.75-0.75

Figura 4.15: Convergência das soluções ao decorrer do tempo usando os algoritmos ALNS, IG e SA

## 4.8 Conclusão

Entre as heurísticas implementadas, observou-se que a heurística SA obteve melhores resultados tanto na comparação em relação ao valor da função objetivo quanto ao tempo de execução. Esse desempenho resulta da escolha dos métodos utilizados para

busca na vizinhança da solução, uma vez que os métodos utilizados são simples e rápidos e que exploram as características relevantes do problema como tempo de chegada, data de entrega e atraso das tarefas. É observado também que o SA apresentou resultados bons com critérios de para menores, uma possível razão para isso seria a aceitabilidade de muitas soluções de pior qualidade. Assim como a heurística SA, o IG é uma meta-heurística simples, eficiente e de fácil implementação. O IG proposto neste trabalho considera o tamanho da destruição variável.

A heurística ALNS consiste em uma heurística de busca local composta de um número de sub-heurísticas concorrentes, que são utilizadas para modificar a solução atual. Os métodos de inserção e remoção utilizados possibilitam intensificar e diversificar a pesquisa. Ao utilizar o algoritmo ALNS, é possível explorar complexas e extensas vizinhanças de busca. Embora o tempo computacional gasto para realizar a pesquisa em uma vizinhança extensa seja consideravelmente alto, o ALNS é capaz de escapar de ótimos locais e encontrar soluções de alta qualidade.

Segundo [Pisinger & Ropke, 2007], a estrutura do ALNS apresenta várias vantagens, uma vez que a heurística ALNS, além de ser um algoritmo capaz de se adaptar a várias características das instâncias. Além disso, ALNS é capaz de explorar grandes partes do espaço de soluções de forma estruturada. É possível observar como exposto anteriormente que apenas são recompensadas as soluções que ainda não foram visitadas. Essa estratégia faz com que os métodos possam ser capazes de explorar novas partes do espaço da solução.

Ao utilizar múltiplas vizinhanças de remoção e inserção durante o processo de busca, é possível realizar uma pesquisa mais abrangente na vizinhança da solução, uma vez que os métodos utilizados no ALNS levam em consideração fatores considerados críticos para o problema como o tempo de chegada e data de entrega. Embora a heurística ALNS tenha obtido bons resultados para o problema em estudo, o ALNS é considerado de difícil implementação devido ao número de parâmetros a serem calibrados.

A heurística VNS apresenta uma busca local eficiente, composta de movimentos que levam em consideração a movimentação de lotes de tarefas. No entanto, esse processo leva a um maior gasto de tempo computacional, uma vez que o algoritmo VNS leva 1 minuto para conseguir uma porcentagem de 10,28% de melhoria em relação ao algoritmo MA.

O algoritmo MA utiliza uma codificação para representação na qual tanto a formação quanto o sequenciamento dos lotes são considerados simultaneamente. O algoritmo MA requer somente um pequeno tempo computacional de 3 segundos por instância. No entanto, não há um método que combine e que particione os lotes

durante o processo, portanto, após encontrada uma solução inicial, não há mudança no número de lotes durante o processo. Desse modo, o número de lotes de cada família é fixado. Um outro fator a ser observado é que assim como na heurística VNS, a busca local proposta não utiliza características do problema como tempo de chegada e data de entrega para gerar a vizinhança de soluções [Chiang et al., 2010].

O algoritmo IG assim como o VNS levam um tempo maior para convergir. No algoritmo VNS isto acontece devido a busca local utilizada. As estruturas propostas para a busca local consistem na movimentação de lotes e tarefas nos quais são testados todas as movimentações possíveis. Esse processo é realizado enquanto não ocorra melhora na solução. A utilização da busca local leva a soluções de qualidade, no entanto, o tempo computacional gasto torna-se elevado, uma vez que os melhores resultados são obtidos com um tempo de 60 segundos por instância. O IG, ao contrário das heurísticas ALNS e SA, utiliza a etapa de busca local durante o processo de busca, o que faz com que a heurística leve um tempo maior para convergir quando comparada ao ALNS e SA.

Os algoritmos ALNS e SA obtiveram melhores resultados devido aos processos utilizados para busca nas vizinhanças, uma vez que esses métodos conseguem explorar de forma eficaz utilizando fatores que as heurísticas VNS, MA e IG não utilizam. Ao levarem em consideração caraterísticas das tarefas que determinam o atraso, como tempo de chegada e data de entrega, essas heurísticas podem explorar de forma mais ampla o espaço de busca. Os movimentos de busca utilizados pelas heurísticas VNS, MA e IG são baseados em movimentos de tarefas não levando em consideração esses fatores, isso pode explicar o porquê dos resultados das heurísticas ALNS e SA terem sido superiores.

Um ponto de relevância a ser considerado para explicar os melhores resultados apresentados pelas heurísticas propostas neste trabalho, ALNS, SA e IG, quando comparadas as heurísticas presentes na literatura, VNS e MA, é a possibilidade de aceitar soluções de pior qualidade durante o processo de busca. Esse processo pode explorar com maior eficiência o espaço de busca, o que pode contribuir para escapar de mínimos locais.

# Capítulo 5

# Conclusão

Este trabalho abordou o problema de sequenciamento de lotes de tarefas em máquinas de processamento idênticas. Esse problema foi proposto por [Mönch et al., 2005], consistindo de uma extensão do trabalho de [Balasubramanian et al., 2004]. O trabalho em estudo pode ser denotado por: $P|rj, dj, batch, incompatible family|\sum w_j T_j$, no qual considera tempo de chegada dinâmico para as tarefas, ou seja, as tarefas nem sempre estão disponíveis no início do processamento. O uso de tempo de chegada dinâmico torna o problema mais realístico, uma vez que em indústrias, muitos processos não estão disponíveis no início do sequenciamento. O objetivo do problema consiste em minimizar o atraso total ponderado.

O problema $P|rj, dj, batch, incompatible family|\sum w_j T_j$ também é abordado em [Chiang et al., 2010]. Os autores propõem um *Memetic Algorithm* (MA) e os resultados apresentados no trabalho são comparados aos resultados de [Mönch et al., 2005]. Os experimentos computacionais demonstram que o algoritmo MA proposto supera os algoritmos GAs do trabalho de [Mönch et al., 2005]..

[Bilyk et al., 2014] acrescenta restrições de precedência ao problema ($P|rj, dj, batch, incompatible family|\sum w_j T_j$) e propõem os algoritmos *Variable Neighborhood Search* (VNS) e *Greedy Randomized Adaptive Search Procedure* (GRASP) para o problema $Pm|rj, d_j, p-batching, incompatible, prec|TWT$. Embora o problema principal do trabalho envolva precedência, também é realizada uma comparação entre os resultados obtidos pelo algoritmo VNS e o algoritmo MA proposto por [Chiang et al., 2010], para o problema sem precedência. Os resultados obtidos demonstram que o VNS é capaz de alcançar uma melhoria de 10,28% em relação ao MA quando o critério de parada é de um minuto por instância.

O problema $P|rj, dj, batch, incompatible family|\sum w_j T_j$ é classificado como NP-difícil, assim, neste trabalho o estudo de métodos heurísticos que possam obter

soluções próximas da ótima em tempo computacional aceitável para resolução do problema é realizado. Para resolver o problema abordado, três algoritmos baseados em meta-heurísticas foram desenvolvidos: Adaptive Large Neighborhood Search (ALNS), Iterated Greedy (IG) e Simulated Annealing (SA). Todos estes algoritmos utilizam técnicas de busca em vizinhança para melhorar a qualidade de uma solução. Além disso, as meta-heurísticas ALNS, IG e SA tem sido aplicadas satisfatoriamente para resolver diferentes problemas de otimização combinatória.

Este trabalho também utiliza a formulação matemática proposta em [Klemmt et al., 2009] para verificar a eficiência do modelo em um conjunto de 288 instâncias de pequeno porte aqui propostas. Para resolver o modelo o software IBM CPLEX é utilizado. O modelo apenas conseguiu encontrar seis valores ótimos para um total de 288 instâncias. Resultados demonstram que há diferença significativa entre os valores encontrados pelo modelo e as heurísticas propostas, sendo que o algoritmo SA obteve a menor média de RPD. O segundo experimento consistiu em verificar a utilização da melhor solução do SA como limitante superior do modelo. Resultados apresentados verificaram que o número de soluções ótimas encontradas foram de 15 instâncias.

O algoritmo *Memetic Algorithm* (MA) proposto por [Chiang et al., 2010] e o algoritmo *Variable Neighborhood Search* (VNS) proposto por [Bilyk et al., 2014] foram reimplemetados e as versões VNS e $MA^*$ foram utilizadas para comparação com as heurísticas propostas. Os algoritmos foram testados em um conjunto de 4860 instâncias de grande porte, disponibilizadas por [Chiang et al., 2010]. Teste estatísticos são utilizados para validar os resultados.

O desempenho dos algoritmos foi avaliado considerando o percentual de melhoria em relação aos melhores resultados disponíveis na literatura. Os resultados obtidos demonstram que o algoritmo SA proposto é eficiente quando comparado com as heurísticas da literatura. Verificou-se também que o algoritmo SA converge mais rapidamente para a melhor solução embora o algoritmo ALNS também seja capaz de obter boas soluções. Observou-se também que os algoritmos VNS e IG são competitivos apresentando resultados similares em algumas situações. Os bons resultados e rápida convergência do algoritmo SA podem ser explicados pela estrutura de perturbação utilizada neste trabalho que consiste em utilizar processos rápidos e eficientes.

Portanto, pode-se concluir que os algoritmos propostos neste trabalho apresentaram resultados de qualidade para resolver o problema em estudo. Experimentos demonstram que as heurísticas desenvolvidas neste trabalho são capazes de superar os melhores resultados reportadas na literatura.

# Referências Bibliográficas


Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345--378.

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

Balasubramanian, H.; Mönch, L.; Fowler, J. & Pfund, M. (2004). Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. *International Journal of Production Research*, 42(8):1621--1638.

Bilyk, A.; Mönch, L. & Almeder, C. (2014). Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. *Computers & Industrial Engineering*, 78:175--185.

Cheng, B.; Wang, Q.; Yang, S. & Hu, X. (2013). An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes. *Applied Soft Computing*, 13(2):765--772.

Cheng, H.-C.; Chiang, T.-C. & Fu, L.-C. (2008). A memetic algorithm for parallel batch machine scheduling with incompatible job families and dynamic job arrivals. In *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*, pp. 541--546. IEEE.

Chiang, T.-C.; Cheng, H.-C. & Fu, L.-C. (2010). A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrival. *Computers & Operations Research*, 37(12):2257--2269.

Damodaran, P. & Velez-Gallego, M. C. (2010). Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times. *The International Journal of Advanced Manufacturing Technology*, 49(9-12):1119--1128.

Damodaran, P. & Vélez-Gallego, M. C. (2012). A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times. *Expert Systems with Applications*, 39(1):1451--1458.

de Lacerda, E. G. & de Carvalho, A. (1999). Introdução aos algoritmos genéticos. *Sistemas inteligentes: aplicaçoes a recursos hıdricos e ciências ambientais*, 1:99--148.

Demir, E.; Bektaş, T. & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2):346--359.

Eriksson, L.; Johansson, E.; Kettaneh-Wold, N.; Wikström, C. & Wold, S. (2000). Design of experiments. *Principles and applications*, pp. 172--174.

Fanjul-Peyro, L. & Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55--69.

Graham, R. L.; Lawler, E. L.; Lenstra, J. K. & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5:287--326.

Graves, S. C. (1981). A review of production scheduling. *Operations research*, 29(4):646--675.

Hemmelmayr, V. C.; Cordeau, J.-F. & Crainic, T. G. (2012). An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & operations research*, 39(12):3215--3228.

Herr, O. & Goel, A. (2016). Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints. *European Journal of Operational Research*, 248(1):123--135.

Herrmann, J. W. et al. (2005). A history of decision-making tools for production scheduling. In *multidisciplinary conference on scheduling: theory and applications, New York*.

Iris, Ç.; Pacino, D. & 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*, 105:123--147.

Jia, Z.-h.; Li, K. & Leung, J. Y.-T. (2015). Effective heuristic for makespan minimization in parallel batch machines with non-identical capacities. *International Journal of Production Economics*, 169:1--10.

Jia, Z.-h.; Wang, C. & Leung, J. Y.-T. (2016). An aco algorithm for makespan minimization in parallel batch machines with non-identical job sizes and incompatible job families. *Applied Soft Computing*, 38:395--404.

Kashan, A. H.; Karimi, B. & Jenabi, M. (2008). A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers & Operations Research*, 35(4):1084--1098.

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

Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. et al. (1983). Optimization by simmulated annealing. *science*, 220(4598):671--680.

Klemmt, A.; Weigert, G.; Almeder, C. & Mönch, L. (2009). A comparison of mip-based decomposition techniques and vns approaches for batch scheduling problems. In *Winter Simulation Conference*, pp. 1686--1694. Winter Simulation Conference.

Kovacs, A. A.; Parragh, S. N.; Doerner, K. F. & Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of scheduling*, 15(5):579--600.

Lausch, S. & Mönch, L. (2016). Metaheuristic approaches for scheduling jobs on parallel batch processing machines. In *Heuristics, Metaheuristics and Approximate Methods in Planning and Scheduling*, pp. 187--207. Springer.

Li, K.; Yang, S.-L. & Ma, H.-W. (2011). A simulated annealing approach to minimize the maximum lateness on uniform parallel machines. *Mathematical and Computer Modelling*, 53(5):854--860.

Mathirajan, M. & Sivakumar, A. (2006). A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. *The International Journal of Advanced Manufacturing Technology*, 29(9-10):990--1001.

Mehta, S. V. & Uzsoy, R. (1998). Minimizing total tardiness on a batch processing machine with incompatible job families. *IIE transactions*, 30(2):165--178.

Mladenović, N. & Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, 24(11):1097--1100.

Mönch, L.; Balasubramanian, H.; Fowler, J. W. & Pfund, M. E. (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research*, 32(11):2731--2750.

Mönch, L.; Fowler, J. W.; Dauzère-Pérès, S.; Mason, S. J. & Rose, O. (2011). A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *Journal of Scheduling*, 14(6):583--599.

Muller, L. F. (2009). An adaptive large neighborhood search algorithm for the resourceconstrained project scheduling problem. In *Proceedings of the VIII metaheuristics international conference (MIC)*.

Palomo-Martínez, P. J.; Salazar-Aguilar, M. A. & Laporte, G. (2017). Planning a selective delivery schedule through adaptive large neighborhood search. *Computers & Industrial Engineering*, 112:368--378.

Perez, I. C.; Fowler, J. W. & Carlyle, W. M. (2005). Minimizing total weighted tardiness on a single batch process machine with incompatible job families. *Computers & Operations Research*, 32(2):327--341.

Pillac, V.; Gueret, C. & Medaglia, A. L. (2013). A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525--1535.

Pinedo, M. L. (2012). *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.

Pisinger, D. & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403--2435.

Pisinger, D. & Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics*, pp. 399--419. Springer.

Potts, C. N. & Kovalyov, M. Y. (2000). Scheduling with batching: a review. *European journal of operational research*, 120(2):228--249.

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

Rodriguez, F. J.; Lozano, M.; Blum, C. & GarcíA-MartíNez, C. (2013). An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers & Operations Research*, 40(7):1829--1841.

Ropke, S. & Pisinger, D. (2006a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455--472.

Ropke, S. & Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750--775.

Ruiz, R. & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033--2049.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*.

Uzsoy, R. (1995). Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33(10):2685--2708.

Vepsalainen, A. P. & Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management science*, 33(8):1035--1047.

Wight, O. W. (1984). *Production and inventory management in the computer age*. John Wiley & Sons, Inc.