VÍVIAN LUDIMILA AGUIAR SANTOS

# SEQUENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS COM DESGASTES DEPENDENTES DA SEQUÊNCIA: RESOLUÇÃO HEURÍSTICA

Dissertação apresentada a 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
2016

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

T
S237s
2016

Santos, Vívian Ludimila Aguiar, 1990-
Sequenciamento de tarefas em máquinas paralelas com desgastes dependentes da sequência : resolução heurística / Vívian Ludimila Aguiar Santos. – Viçosa, MG, 2016.
xi, 79f. : il. (algumas color.) ; 29 cm.

Orientador: José Elias Claudio Arroyo.
Dissertação (mestrado) - Universidade Federal de Viçosa.
Inclui bibliografia.

1. Pesquisa operacional. 2. Otimização combinatória. 3. Solução de problemas. 4. Heurística. I. Universidade Federal de Viçosa. Departamento de Informática. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 22 ed. 001.424

VÍVIAN LUDIMILA AGUIAR SANTOS

**SEQUENCIAMENTO DE TAREFAS EM MÁQUINAS PARALELAS COM DESGASTES DEPENDENTES DA SEQUÊNCIA: RESOLUÇÃO HEURÍSTICA**

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: 06 de julho de 2016.

Mayron César de Oliveira Moreira

André Gustavo dos Santos
Coorientador

José Elias Claudio Arroyo
Orientador

*Dedico este trabalho aos meus amados pais Flor de Maio e Valdemar, e ao meu namorado Thales.*

# Agradecimentos

Agradeço, primeiramente, a Deus porque é a luz, fortaleza, proteção e sabedoria que dá sentido à minha vida. Por tudo que Ele tem realizado por mim.

Aos meus pais, Flor de Maio e Valdemar, meu infinito agradecimento, pela educação, e por todo apoio e incentivo nos meus estudos. Vocês são meus exemplos de vida, generosidade, hospitalidade, honestidade, responsabilidade e determinação.

Aos meus irmãos, Vera e Mateus, por estarem sempre prontos para me ajudar e por terem me dado os melhores presentes que poderiam: meus sobrinhos. Estes anjinhos que tornam meus dias mais doces, alegres e engraçados.

Ao meu orientador, Prof. Arroyo, por sua dedicação, paciência e ensinamentos, e por ser exemplo de profissional. Ao meu coorientador, Prof. André, pela atenção e valiosas sugestões. Ao Prof. Michele Monaci pelas inúmeras contribuições para este trabalho. A todos os docentes e funcionários do Departamento de Informática da UFV que de alguma forma contribuíram para o meu crescimento profissional.

À minha companheira de república, Fá, pelo aconchego e liberdade em sua casa, por toda gentileza, paciência e ensinamentos, e pelas comidas gostosas.

A todos da família do meu namorado, família Mota, por me acolherem nos seus lares e se tornarem a minha família em Viçosa, pelos deliciosos almoços de domingos e por me proporcionaram tantos momentos de lazer.

Agradeço às minhas amadas avós, pelas orações e carinho. A toda minha família, a qual amo muito, e aos meus amigos pela atenção e por fazerem parte da minha vida nos momentos bons e ruins. Em especial, pela amizade da minha irmã de coração, Li. A todos os amigos do mestrado pelo companheirismo e apoio.

Por fim e muito importante, agradeço ao meu grande amor, Thales, por não medir esforços para me ajudar, muitas vezes deixando suas obrigações para se preocupar com as minhas. A sua participação foi essencial para a realização deste trabalho, sem você ao meu lado eu não chegaria até aqui. Muito obrigada por tudo! Principalmente, pelo amor, amizade, carinho, paciência, dedicação, cumplicidade e lealdade comigo; por me tornar uma pessoa melhor e mais feliz.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Resumo

SANTOS, Vívian Ludimila Aguiar, M.Sc., Universidade Federal de Viçosa, julho de 2016. **Sequenciamento de Tarefas em Máquinas Paralelas com Desgastes Dependentes da Sequência: Resolução Heurística.** Orientador: José Elias Claudio Arroyo. Coorientador: André Gustavo dos Santos.

Este trabalho aborda o problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas em que as tarefas causam desgastes nas máquinas. Este fator diminui o desempenho das máquinas levando ao aumento do tempo de processamento das tarefas ao longo do tempo. O objetivo do problema é encontrar as sequências de processamento de tarefas em cada máquina de tal maneira que os desgastes das máquinas sejam reduzidos e, consequentemente, minimizar o tempo máximo de conclusão de todas as tarefas, conhecido como *makespan*. Neste trabalho, inicialmente, é proposto um novo modelo de Programação Inteira Mista baseado na geração de padrões (conjuntos de tarefas) para cada máquina, com objetivo de obter soluções ótimas para o problema. Dado que o problema é NP-Difícil para mais de uma máquina, dois algoritmos heurísticos são propostos para obter soluções de alta qualidade em baixo tempo computacional. Os algoritmos são baseados nas meta-heurísticas *Iterated Local Search* (ILS) e *Iterated Greedy* (IG), respectivamente. Também, as heurísticas ILS e IG são combinadas com uma variante do método *Variable Neighborhood Descent* (VND), que utiliza uma ordenação aleatória das vizinhanças (RVND) na fase da busca local, obtendo dois algoritmos híbridos denominados ILS-RVND e IG-RVND. O *benchmark* usado nos experimentos computacionais usa 900 instâncias de médio porte disponíveis na literatura, e 900 instâncias de grande porte geradas neste trabalho. Os algoritmos são comparados entre si e também com um algoritmo *Simulated Annealing* (SA) proposto na literatura para o mesmo problema. Os testes realizados mostram que os desempenhos dos algoritmos propostos são significativamente superiores em relação ao algoritmo SA.

# Abstract

SANTOS, Vívian Ludmila Aguiar, M.Sc., Universidade Federal de Viçosa, July of 2016. **Unrelated Parallel Machine Scheduling with Sequence Dependents Deteriorations: Resolution Heuristics.** Adviser: José Elias Claudio Arroyo. Co-advisor: André Gustavo dos Santos.

This work addresses an unrelated parallel machine scheduling problem in which the jobs cause deterioration of the machines. This factor decreases the performance of the machines, causing an increasing of the jobs over time. The problem is to find the processing sequence of jobs on each machine in order to reduce the deterioration of the machines and consequently minimize the maximum completion time of jobs (makespan). In this work, initially, we propose a new Mixed-Integer Programming model based on patterns (sets of jobs) generation to find optimal solution of the problem. Since the problem is NP-hard when the number of machines is greater than one, two heuristic algorithms are proposed to obtain near-optimal solutions in reasonable computational time. The algorithms are based on the meta-heuristics Iterated Local Search (ILS) and Iterated Greedy (IG), respectively. Also, the algorithms ILS and IG are coupled with a variant of the Variable Neighborhood Descent (VND) method that uses a random ordering of neighborhoods (RVND) in local search phase, obtaining two hybrid algorithms called ILS-RVND and IG-RVND. The benchmark used in computational experiments uses 900 medium-size instances available in the literature, and 900 large-size instances generated in this work. The algorithms are compared against each other and are also compared with a Simulated Annealing (SA) algorithm proposed in the literature for the problem under study. The tests show that the proposed algorithms have superior performances compared to the SA algorithm.

# Capítulo 1

# Introdução

## 1.1 Considerações iniciais

Pode-se perceber o quanto a oferta de materiais e produtos tem crescido nos últimos anos. Assim como, notar que a globalização e a ampliação de vendas pela internet transformaram a concorrência de local para mundial. Isso provocou desafios decisivos para que as organizações consigam permanecer firmes no mercado de vendas e sobrevivam em um ambiente de intensa competição. Desta forma, tornou-se necessário às empresas atender as crescentes exigências por parte dos consumidores, como por exemplo, melhor qualidade dos produtos, maior variação de modelos, entregas mais confiáveis e rápidas, com menores custos. Para auxiliar as empresas a continuarem engajadas em mercados de dimensões mundiais, surgiram os eficientes sistemas de planejamento e controle da produção(Russomano, 2000).

Segundo Zaccarelli (1979), este sistema é um conjunto de funções inter-relacionadas que objetivam gerenciar o processo de produção e organizá-lo com os outros setores administrativos da empresa. Chase et al. (2006) apresenta alguns dos principais objetivos do planejamento da produção, tais como: cumprir com as datas de entrega, minimizar o tempo de processamento, minimizar o tempo ou custo de preparação, minimizar o estoque, maximizar a utilização da máquina ou a mão de obra.

Dentro do sistema de planejamento e controle da produção, existe a atividade de programação da produção, a qual trata da atribuição dos recursos, de forma eficiente, para o conjunto de processos de fabricação. A programação da produção determina o mais adequado tempo para executar cada operação ou tarefa, tendo em conta as relações entre os processos de fabricação e a capacidade limitada dos recursos de produção (Shen et al., 2006).

A pressão existente nas empresas para melhorar a qualidade na prestação de seus serviços tornou a programação da produção de muito interesse nas indústrias, a fim de que consigam aumentar sua produtividade e lucratividade (Shen et al., 2006). Este processo, visto essencialmente como um viés em otimização, tenta evitar uma programação que funciona bem para uma parte da organização, mas que cria problemas para as outras, ou, mais grave, para os clientes (Chase et al., 2006). Além disso, a programação da produção busca otimizar os critérios que estão associados a utilização eficiente dos recursos, diminuição no tempo total e custos da produção, atender as datas combinadas para entrega dos produtos, obter qualidade, confiabilidade, velocidade e flexibilidade nos processos industriais (Lustosa et al., 2011).

É neste contexto que se inserem os problemas de sequenciamento de tarefas, os quais visam reduzir o tempo de processamento final em cada máquina, assegurando a alocação eficiente dos recursos. Um dos variantes de problemas de sequenciamento de tarefas, considera o desgaste que as tarefas causam nas máquinas. Esta variante tem sido recentemente estudada devido a sua aplicação no gerenciamento logístico, uma vez que o desgaste das máquinas afeta o tempo final do processamento das tarefas. O rendimento da produção de cada máquina pode diminuir devido a uma má posição das ferramentas, mal alinhamento das tarefas ou arranhadura de ferramentas. Nessas situações, o tempo de processamento das tarefas aumenta dependendo das posição que as tarefas se encontram (Joo & Kim, 2015).

Este trabalho tem como foco propor métodos heurísticos para a resolução do problema de sequenciamento de tarefas em máquinas paralelas, levando em conta os desgastes causados nas máquinas dependentes das posições em que se encontram as tarefas.

## 1.2 O problema e sua importância

Em problemas determinísticos da programação da produção, os tempos de processamento das tarefas são geralmente conhecidos com antecedência e permanecem constantes durante o sequenciamento. No entanto, existem muitas situações práticas em que os tempos de processamento das tarefas podem incrementar ao longo do tempo, de tal forma que quanto mais tarde inicia-se o processamento de uma tarefa, mais tempo é necessário para processá-la. Este fenômeno é conhecido como o efeito de deterioração no tempo de processamento de tarefa (Yang et al., 2012).

Os primeiros autores que consideraram o conceito de desgaste ou deterioração

em problemas de programação da produção foram Gupta et al. (1987) e Browne & Yechiali (1990). Por exemplo, Kunnathur & Gupta (1990) mostraram que o tempo e o esforço necessário para controlar um incêndio aumentam se há um atraso no início do combate ao incêndio. Gupta & Gupta (1988) expõem o exemplo na indústria do aço, em que, se a temperatura de um lingote, esperando para entrar na máquina de rolamento, cair abaixo de certo nível, então, o lingote necessitará ser reaquecido antes de ser processado novamente.

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

Além da importância prática, o problema abordado neste trabalho é também interessante do ponto de vista de complexidade computacional, já que é um problema classificado como $\mathcal{NP}$-Difícil (Ruiz-Torres et al., 2013). É um problema desafiador, já que não se conhecem algoritmos eficientes para sua resolução.

## 1.3 Motivação

Uma vez que o problema em questão é considerado de difícil solução, este trabalho baseia-se na hipótese de que, para encontrar soluções de boa qualidade em um tempo computacional aceitável, deve-se utilizar algoritmos baseados em meta-heurísticas. Para tanto, são propostas adaptações das meta-heurísticas *Iterated Local Search, Iterated Greedy e Randon Variable Neighborhood Descent*.

## 1.4 Objetivos

O objetivo geral deste trabalho é propor e desenvolver heurísticas baseadas em meta-heurísticas, que visem encontrar uma sequência de tarefas para cada máquina a fim de minimizar o tempo máximo de conclusão das tarefas. Além disso, propor um novo modelo de Programação Inteira-Mista a fim de obter soluções ótimas para o problema.

### 1.4.1 Objetivos Específicos

Para alcançar o objetivo geral, é necessário que sejam atingidos os objetivos específicos, listados a seguir:

(a) Realizar revisões bibliográficas para obter atualizações dos trabalhos que envolvem o sequenciamento de tarefas em máquinas paralelas e com efeito de desgaste.

(b) Implementar a abordagem da literatura que trata o referido problema.

(c) Propor novas abordagens heurísticas para este problema.

(d) Implementar cada uma das abordagens propostas.

(e) Fazer experimentos computacionais para calibração dos parâmetros utilizados nas estratégias implementadas.

(f) Realizar uma análise comparativa entre os resultados obtidos a partir das heurísticas propostas e os resultados da abordagem heurística da literatura.

(g) Propor uma nova formulação matemática para linearizar o modelo matemático do problema.

(h) Desenvolver técnicas para determinar as soluções ótimas do problema, quando possível, a partir do modelo matemático linearizado.

## 1.5 Estrutura do Trabalho

O restante deste trabalho está organizado da seguinte maneira: o Capítulo 2 aborda as principais características dos métodos de resolução para os problemas de otimização combinatória. No Capítulo 3, é apresentada a caracterização do problema estudado neste trabalho e uma revisão bibliográfica dos trabalhos que tratam de problemas em máquinas paralelas considerando desgastes. O Capítulo 4 apresenta a formulação dos modelos matemáticos propostos para o problema em estudo. No Capítulo 5, são apresentadas as meta-heurísticas aplicadas para solucionar o problema. Os Capítulos 6 e 7 apresentam os experimentos realizados neste trabalho. Os experimentos foram divididos em duas partes. Na primeira parte são mostradas as instâncias utilizadas nos testes, a métrica para avaliação, as calibrações dos algoritmos heurísticos e a implementação do algoritmo SA* proposto na literatura. A segunda parte apresenta os experimentos computacionais com o modelo de programação linear inteiro e com os métodos heurísticos. Finalmente, no Capítulo 8 são apresentadas as conclusões.

# Capítulo 2

# Métodos para Resolução de Problemas de Otimização Combinatória

Problemas de programação da produção são tratados por meio de métodos de otimização combinatória. Estes são classificados, de acordo com Talbi (2009), em dois grupos: métodos exatos e métodos aproximados. Estes dois grupos são explicados nas Seções 2.1 e 2.2, respectivamente.

Anteriormente, faz-se necessário definir alguns termos próprios da Otimização Combinatória para melhor entendimento da seções seguintes. São eles:

- Solução Viável: uma solução é dita ser viável se respeita todas as restrições do problema (Talbi, 2009). Define-se $S$ como o conjunto de soluções viáveis.
- Função Objetivo: a função objetivo $f$ associa a cada solução $s \in S$ um número real indicando a sua qualidade (Talbi, 2009).
- Ótimo global: para um problema de minimização, uma solução $s^* \in S$ é um ótimo global se ela tem uma melhor função objetivo de todas as soluções dos espaço de busca, isto é, $\forall s \in S, f(s^*) \leq f(s)$ (Talbi, 2009).
- Vizinhança: duas soluções são ditas vizinhas se, a partir de determinada alteração em uma solução, se alcança a outra (Talbi, 2009).
- Ótimo Local: uma solução $s \in S$ é dita ser um ótimo local se ela tem a melhor função objetivo dentre todas as soluções que são vizinhas. Isto é, $\forall s' \in N(s), f(s) \leq f(s')$, onde $N(s)$ são os vizinhos de $s$ (Talbi, 2009).

## 2.1 Métodos Exatos

A programação linear é o setor da Pesquisa Operacional (PO) cujo modelo é representado por uma função linear das variáveis (função objetivo) e restrições lineares (Góes, 2005). Já na programação linear inteira, as variáveis só podem assumir valores inteiros. O algoritmo *Simplex* encontra a solução ótima de problemas de programação linear contínuo em um número finito de passos (Nelder & Mead, 1965). Entretanto, existem vários problemas que não estão nesta classe de problemas contínuos, sendo que muitas vezes são problemas de programação inteira mista (Talbi, 2009).

Os métodos exatos procuram a melhor solução para o problema, satisfazendo todas as restrições impostas (Talbi, 2009). Alguns algoritmos clássicos de otimização são utilizados para resolução de problemas de programação linear inteiros. Estes algoritmos são enumerativos e podem ser vistos como algoritmos de busca em árvore. A pesquisa é efetuada ao longo de todo o espaço de busca e o problema é resolvido ao subdividi-lo em problemas mais simples. Entre os métodos enumerativos encontram-se os algoritmos *branch-and-bound*, plano de cortes e *branch-and-cut*, explicados a seguir:

- *Branch-and-bound* é baseado em uma enumeração implícita de todas as soluções do problema. O espaço de busca é explorado através da construção dinâmica de uma árvore cujo nó raiz representa o problema a ser resolvido e está inteiramente associado o seu espaço de busca. Os nós folhas são as soluções possíveis e os nós internos são subproblemas do espaço total da solução. Este algoritmo possui dois procedimentos principais: ramificação e poda. O procedimento de ramificação é responsável pela construção de soluções parciais, que podem ser também ramificadas gerando outras soluções parciais. Estes nós parciais podem também ser podados (não serão ramificados) quando se tem garantia de que não levarão a uma solução ótima. Os nós folhas das árvores são soluções completas e possíveis candidatas a ótimos globais. O procedimento de poda é baseado em uma função delimitadora que poda subárvores que não contêm qualquer solução ótima (Talbi, 2009).

- Plano de Cortes: consiste em abordar a programação inteira como um problema da programação linear e ao encontrar uma solução para este problema impõe condições em que as variáveis devem ser inteiras. Desta forma, acrescentam-se restrições, sucessivamente, que eliminam parte do espaço de soluções, inclusive a solução ótima não inteira, até encontrar a melhor solu-

ção inteira para o problema. Contudo não elimina qualquer solução inteira (Góes, 2005). Em algoritmos de planos de corte, os cortes são adicionados iterativamente através de restrições específicas no problema relaxado. Estas restrições adicionadas ao problema relaxado são feitas de tal forma que este se aproxime cada vez mais do problema inteiro.

- *Branch-and-cut* é um método que combina as estratégias dos métodos *branch-and-bound* e Plano de Corte com o objetivo de reduzir o número de nós na árvore gerada pelo *branch-and-bound*. Desta forma, em cada nó da árvore, o algoritmo *branch-and-cut* adiciona cortes válidos de modo a obter um limitante superior (ou inferior) de forma mais rápido (Talbi, 2009).

Apesar destes métodos encontrarem a solução exata, geralmente são utilizados para problemas de tamanho pequeno, uma vez que problemas de grande porte se tornam inviáveis computacionalmente de serem resolvidos devido ao longo tempo de execução e ao alto gasto de memória. Assim, métodos aproximados são tipicamente utilizados para resolução de problemas maiores. A seção a seguir aborda tais métodos.

## 2.2 Métodos Aproximados

Existem diversos problemas da área de PO para os quais não se conhece nenhum algoritmo exato com complexidade polinomial que encontre a solução ótima global. Estes problemas possuem complexidade computacional pertencente à classe *NP-Difícil*. Para estes tipos de problemas uma solução de boa qualidade já é suficiente, não sendo necessário que a solução ótima global seja encontrada. Nestes casos, utilizam-se os métodos aproximados, os quais encontram soluções próximas da solução ótima em tempo computacional razoável e são muito relevantes para resolver problemas com entrada de dados de grande dimensão (Assis, 2012).

Segundo Talbi (2009), a classe dos métodos aproximados é dividida em duas subclasses: algoritmos aproximativos e heurísticos, explicados a seguir:

- Algoritmos aproximativos proporcionam a qualidade da solução em relação ao ótimo global e prováveis limites de tempo de execução.

- Algoritmos heurísticos procuram alcançar uma solução satisfatória viável, de maneira eficiente, mas sem garantia de qualidade da solução.

Uma vez que não se conhece matematicamente a qualidade da solução do problema em estudo, não é possível utilizar os algoritmos aproximativos. Por isso, neste trabalho, são usados os algoritmos heurísticos para resolução do problema. Estes, por sua vez, são classificados em duas famílias: heurísticas específicas e meta-heurísticas. Estas duas famílias são descritas nas subseções a seguir.

### 2.2.1 Heurísticas

As heurísticas são métodos que não possuem muitas exigências para solucionar um determinado problema, contudo necessitam de uma expertise sobre o problema. Portanto, para os problemas de otimização combinatória, heurística se refere a métodos que produzem uma solução a partir de conhecimentos específicos do problema, mas sem garantir a qualidade do resultado obtido (Melián et al., 2003). A seguir são apresentados os conceitos da heurística construtiva e heurística de melhoria.

- A heurística construtiva constrói uma solução inicial a partir de regras baseadas nos dados do problema. Estes algoritmos são específicos e requerem um conhecimento do problema para a sua elaboração (Papadimitriou, 2003).

- A heurística de melhoria, também conhecida como busca local, é um processo que é inicializado com uma solução viável e através de movimentos nos elementos da solução, mas que mantenham as características da solução original, percorre-se a vizinhança em busca de uma solução melhor que a solução atual até alcançar um ótimo local.

  Existem duas estratégias para se realizar uma busca local, sendo elas, a primeira descida e máxima descida. Na estratégia primeira descida, a primeira solução vizinha com melhor avaliação para a função objetivo, é retornada pelo procedimento da busca local. Esta abordagem evita a exploração exaustiva, sendo que somente no pior caso toda a vizinhança é explorada. Já na estratégia de máxima descida, todas as soluções vizinhas são avaliadas e aquela que melhor otimiza a função objetivo será retornada pelo procedimento da busca local. A abordagem máxima descida encontra uma solução ótima local, o que já não pode ser afirmado em relação à abordagem primeira descida. Porém, ao analisar todas as soluções da vizinhança, a abordagem máxima descida demanda um maior custo computacional (Assis, 2012).

### 2.2.2 Meta-heurísticas

Meta-heurísticas são algoritmos de uso geral que podem ser aplicados para resolver qualquer problema de otimização (Talbi, 2009). Estes algoritmos podem ser organizadas em dois grupos, que são descritos a seguir:

- Meta-heurística baseada em população: evolui toda uma população de soluções. Esta permite uma melhor diversificação em todo o espaço de busca. Exemplos: Enxame de Partículas, Algoritmos Evolucionários.
- Meta-heurística baseada em solução única: manipula e transforma uma única solução durante a busca. Esta tem o poder para intensificar a busca em regiões locais. Exemplos: *Simmulated Anealing*, *Iterated Local Search*, *Variable Neighborhood Search* e *Iterated Greedy*.

O foco deste trabalho é desenvolver adaptações de meta-heurísticas para resolver o problema em estudo. Os métodos propostos são alicerçados em algoritmos de busca sequencial e busca em vizinhança, tais como: *Iterated Local Search*, *Random Variable Neighborhood Descent* e *Iterated Greedy*. As próximas seções tratam da apresentação destas meta-heurísticas.

#### 2.2.2.1 *Iterated Local Search*

A meta-heurística Busca Local Iterativa, *Iterated Local Search* (ILS), foi introduzida por Lourenço et al. (2003). De forma sucinta, o funcionamento do ILS possui quatro passos. No primeiro passo, obtém-se uma solução inicial. O segundo passo consiste em usar uma busca local para atingir algum ótimo local. No terceiro passo, o procedimento de perturbação é utilizado para escapar do ótimo local. Este procedimento deve conservar uma parte da solução e modificar outra. Por fim, o critério de aceitação define em quais condições um novo ótimo local substitui a solução atual.

O Algoritmo 1 apresenta o pseudocódigo do ILS. Percebe-se que na linha 2 gera-se uma solução inicial $S_0$. Essa solução é refinada por um método de busca local, gerando um ótimo local $S^*$ (linha 3). A solução $S^*$ sofre um perturbação, dando origem a uma outra solução $S'$ (linha 5). Esta solução $S'$ passa por um processo de refinamento, resultando em um novo ótimo local $S''$ (linha 6). Em seguida é verificado de qual solução a busca prosseguirá, se do ótimo local corrente ($S^*$) ou do novo ($S''$ ). O procedimento continua até que o critério de parada seja satisfeito (Subramanian et al., 2013a).

**Algoritmo 1:** ILS

```
1  início
2      S_0 ← Solução_Inicial( )
3      S* ← Busca_Local(S_0)
4      repita
5          S' ← Perturbação(S*)
6          S'' ← Busca_Local(S')
7          S* ← Critério_Aceitação (S*, S'')
8      até Critério de Parada;
9      retorna S*
10 fim
```

#### 2.2.2.2 *Random Variable Neighborhood Descent*

A Busca em Vizinhança Variável, *Variable Neighborhood Search* (VNS), é uma metaheurística que explora o espaço de soluções através de trocas sistemáticas da função de vizinhança. Baseia-se nas seguintes definições: um ótimo local de uma vizinhança não é necessariamente ótimo local de outra vizinhança e um ótimo global é um ótimo local de todas as estruturas de vizinhanças (Hansen & Mladenović, 2001). Existem diversas variações do VNS que alteram a forma como as vizinhanças são exploradas.

Uma destas extensões é o método de Descida em Vizinhança Variável, *Variable Neighborhood Descent* (VND), proposto por Mladenović & Hansen (1997). Neste trabalho, é utilizado o método de Descida em Vizinhança Variável Aleatória, *Random Variable Neighborhood Descent* (RVND), proposto por Hansen & Mladenović (1999). Ao contrário do VND que utiliza uma ordem pré-definida de vizinhanças para explorar o espaço de soluções, o RVND utiliza uma ordem aleatória a cada chamada (Hansen et al., 2008). Este método tem sido usado em vários trabalho de otimização combinatória (Subramanian et al., 2013a), (Subramanian et al., 2013b), (Penna et al., 2013).

O funcionamento do RVND é explicado no Algoritmo 2. LV é uma Lista de Vizinhanças (linha 2). Repetidamente, uma vizinhança $N_i \in LV$ é selecionada aleatoriamente (linha 4) e o melhor vizinho ($s'$) de $s$ é determinado (linha 5). Se a função objetivo de ($s'$) for menor que a de ($s$), então a solução $s$ é atualizada e LV é reinicializada com todas as vizinhanças (linhas 6 a 9). Caso contrário, $N_i$ é removida da LV (linha 11). O laço de repetições termina quando a lista LV tornar-se vazia.

**Algoritmo 2:** RVND

```
1  início
2      Inicialize a Lista de Vizinhanças (LV)
3      enquanto LV ≠ ∅ faça
4          Escolha uma vizinhança N_i ∈ LV aleatoriamente
5          Encontre o melhor vizinho s' de s ∈ N_i
6          se f(s') < f(s) então
7              s ← s'
8              Atualize a LV
9          fim
10         senão
11             Remova N_i da LV
12         fim
13     fim
14     retorna s
15 fim
```

#### 2.2.2.3 *Iterated Greedy*

A meta-heurística Iterada Gulosa, *Iterated Greedy* (IG), foi introduzida por Ruiz & Stützle (2007) para o problema permutacional *flowshop scheduling*. A IG possui duas fases: destruição e reconstrução. Na fase de destruição, alguns elementos são extraídos aleatoriamente da solução. Na fase de reconstrução, estes elementos são reinseridas um por um, de forma gulosa, cada um na melhor posição da solução parcial.

O Algoritmo 3 demonstra o funcionamento do IG. Na linha 2, percebe-se que $s_0$ obtém a solução inicial. Em seguida, busca-se um refinamento desta solução $s_0$ com a busca local. No próximo passo, dentro de um laço de repetições, ocorrem as etapas de destruição e reconstrução em que o parâmetro $d$ representa a quantidade de elementos que são removidos e inseridos da solução (linha 5). Em seguida, novamente, busca-se um melhoramento da solução com a busca local (linhas 6). Nas próximas linhas (8 a 12) a solução corrente $s$ e a melhor solução encontrada até o momento $s^*$ são atualizadas. Se a solução $s''$ não é melhor que a solução $s$, um critério de aceitação é aplicado. Por fim, a melhor solução $s^*$ é retornada.

**Algoritmo 3:** IG

```
1  início
2      s_0 ← Solução_Inicial( )
3      s ← Busca_Local(s_0)
4      enquanto Critério de Parada não Satisfeito faça
5          s' ← Destruição_Construção (s, d)
6          s'' ← Busca_Local(s')
7          se f(s'') < f(s) então
8              s ← s''
9              se f(s) < f(s*) então
10                 s* ← s
11             fim
12         fim
13         senão
14             s ← Critério_Aceitação(s, s'', histórico)
15         fim
16     fim
17     retorna s*
18 fim
```

# Capítulo 3

# Sequenciamento de Tarefas em Máquinas Paralelas Considerando Desgastes Dependentes Da Sequência

Para uma melhor compreensão do problema abordado neste trabalho, as próximas seções apresentam as características dos problemas de sequenciamento de tarefas em máquinas paralelas não-relacionadas considerando o efeito de desgaste dependente da sequência. Alguns dos trabalhos mais proeminentes encontrados na literatura são apresentados. Antes, faz-se necessário definir os problemas de programação da produção.

## 3.1 Problemas de programação da produção

De acordo com Lopes et al. (2013), a Pesquisa Operacional (PO) é uma área que busca desenvolver modelos matemáticos e algoritmos para resolver problemas de tomadas de decisão reais e complexos. A linha de pesquisa otimização combinatória está envolvida na grande área da PO. Dentre os problemas de otimização combinatória, encontram-se os problemas de programação da produção (em inglês *Scheduling*), introduzido por Henry Gantt em 1918, quem desenvolveu um sistema de programação da produção, baseado em gráficos e cálculos, e fundamentado em restrições de capacidade e tempo (Lustosa et al., 2011). Assim, a partir da obra de Henry Gantt, a programação da produção obteve importância no processo de fabricação.

Em um problema da programação da produção, existe um conjunto $J$ de $n$ tarefas $J_i$ *(i = 1,..., n)* e um conjunto $M$ de $m$ máquinas $M_k$ *(k = 1,...,m)*. Cada tarefa deverá ser processada em um subconjunto de $M$. O problema então visa encontrar a alocação de tarefas à(s) máquina(s) e o sequenciamento destas tarefas de modo que determinado(s) objetivo(s) de produção seja(m) alcançado(s) e as demais restrições associadas tanto às tarefas quanto às máquinas sejam respeitadas (Brucker, 1998).

Existem diversas variações do clássico problema de programação da produção, como pode ser observado em Graham et al. (1979), onde são exibidos os problemas de máquinas simples, máquinas paralelas, *flow-shop* (produção contínua[1]) e *job-shop* (produção intermitente[2]). O problema abordado neste trabalho é uma destas variações e suas principais características são definidas na seção seguinte.

## 3.2 Definição do Problema Abordado

O presente problema consiste em processar as tarefas em máquinas paralelas levando em conta que cada tarefa executada provoca um certo desgaste na máquina. Este desgaste diminuirá o desempenho da máquina, aumentando o tempo de processamento da tarefa seguinte a ser efetuada. A posição de uma tarefa na sequência é muito significativa para o rendimento total da máquina, ou seja, o desgaste da máquina depende da ordem das tarefas.

O problema é definido por Ruiz-Torres et al. (2013) da seguinte forma:

(a) Existem $n$ tarefas, estabelecendo o conjunto $N = \{1, \ldots, j, \ldots, n\}$, para serem processadas em $m$ máquinas paralelas, que formam o conjunto $M = \{1, \ldots, k, \ldots, m\}$.

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

(c) Cada máquina pode processar somente uma tarefa de cada vez, não pode ficar ociosa até a última tarefa atribuída ter sido finalizada e estas máquinas são

[1]No ambiente de produção contínua existem $m$ máquinas diferentes em série e $n$ tarefas. As tarefas devem ser processadas exatamente uma vez em cada máquina e a ordem entre o processamento das tarefas nas máquinas deve ser respeitada.

[2]O ambiente da produção intermitente é o caso geral da fábrica de produção contínua no qual existem $m$ máquinas diferentes e $n$ tarefas, mas estas têm uma rota diferente e predefinida e não necessariamente devem ser processadas em todas as máquinas

classificadas como máquinas paralelas não-relacionadas, isto é, cada tarefa $j$ possui um tempo de processamento que depende da máquina $k$ na qual será alocada.

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

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

(f) Ruiz-Torres et al. (2013) apresenta uma equação para calcular o desempenho da máquina $k$ antes de processar uma tarefa. Considere uma sequência de tarefas para a máquina $k$. O nível de desempenho da máquina $k$ para processar a tarefa na posição $h$ ($h > 1$) é determinado por:

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

onde, $q_{k[h-1]}$ é o efeito de desgaste da tarefa na posição $h-1$. É assumido que as máquinas iniciam sem efeito de desgaste, portanto, o nível de desempenho para processar a tarefa que está na primeira posição é 100%, isto é, $q_{k[1]} = 1, \forall k \in M$.

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

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

(h) O objetivo deste problema é sequenciar as tarefas nas máquinas a fim de minimizar o máximo tempo de conclusão do processamento das tarefas, também conhecido como *makespan* ($C_{max}$).

(i) Por fim, é importante destacar que este problema é NP-difícil para $m \geq 2$ (Ruiz-Torres et al., 2013).

De acordo com a classificação de Graham et al. (1979), este problema será denotado $Rm|Sdd|C_{max}$ em que $Rm$ significa o ambiente das máquinas não-relacionadas, $Sdd$ corresponde às restrições de desgaste dependentes da sequência e $C_{max}$ é relativo ao critério de otimização.

Um exemplo prático do problema é o uso de ferramentas de corte. Estas ferramentas deterioram (desgastam) diferentemente dependendo da espessura do

material a ser cortado. Se materiais menos resistentes são cortados primeiros, as ferramentas irão desgastar menos, portanto, as máquinas manterão alto nível de desempenho (velocidade de corte). Por outro lado, se materiais mais resistentes são cortados primeiros, as ferramentas irão se desgastar mais, portanto, a velocidade de corte da máquina diminuirá mais rápido e, por consequência, materiais menos resistentes cortados depois gastarão mais tempo.

A seguir apresenta-se um exemplo numérico para o problema $Rm|Sdd|C_{max}$.

### 3.2.1 Exemplo numérico

Para mostrar as características do problema em estudo, a seguir é apresentado um exemplo para uma instância do problema com $n = 8$ tarefas e $m = 3$ máquinas. A Tabela 3.1 apresenta os valores de tempos de processamento ($p_{jk}$) e as porcentagens dos desgastes ($d_{jk}$) de cada tarefa $j$ em cada máquina $k$ ($j = 1, ..., 8, k = 1,..., 3$).

**Tabela 3.1.** Exemplo de instância para 8 tarefas e 3 máquinas

| Tarefa | $p_{j1}$ | $p_{j2}$ | $p_{j3}$ | $d_{j1}$ | $d_{j2}$ | $d_{j3}$ |
|---|---|---|---|---|---|---|
| 1 | 26,5 | 63,5 | 65,5 | 0,04 | 0,01 | 0,01 |
| 2 | 20,0 | 27,9 | 46,6 | 0,03 | 0,02 | 0,03 |
| 3 | 30,5 | 47,9 | 62,4 | 0,02 | 0,02 | 0,01 |
| 4 | 31,1 | 22,4 | 80,0 | 0,02 | 0,02 | 0,01 |
| 5 | 82,8 | 77,4 | 92,3 | 0,03 | 0,04 | 0,04 |
| 6 | 50,0 | 90,6 | 99,3 | 0,01 | 0,04 | 0,01 |
| 7 | 56,4 | 28,2 | 76,2 | 0,03 | 0,03 | 0,01 |
| 8 | 21,8 | 42,3 | 24,5 | 0,02 | 0,03 | 0,03 |

Na Figura 3.1, mostra-se um sequenciamento no qual as tarefas 2, 6 e 3 são atribuídas aleatoriamente para a máquina 1, as tarefas 7, 4 e 5 para a máquina 2 e as tarefas 8 e 1 para a máquina 3. Para encontrar o tempo de processamento atual das tarefas é necessário calcular o desempenho da máquina após a execução de uma tarefa. Na Figura 3.1, observa-se que as máquinas $M_1$, $M_2$ e $M_3$, na primeira posição, executam as tarefas 2, 7 e 8, respectivamente, com 100% de desempenho. Estas primeiras tarefas causam certos desgastes diminuindo o desempenho das máquinas. Por exemplo, a tarefa 7 provocou um desgaste na máquina 2, então o novo desempenho desta máquina na posição $h = 2$ (após executar a tarefa 7) é obtido pela Equação (3.1) da seguinte maneira: $q_{2[2]} = (1 - d_{[1]2}) \times q_{2[1]} = (1 - 0{,}03) \times 1 = 0{,}97 = 97\%$. Como o desempenho da máquina diminuiu, então a tarefa que ocupa a segunda posição será executada em um tempo maior. O tempo de processamento

atual para a tarefa 4, que ocupa a posição $h = 2$ na máquina 2, é obtido da seguinte maneira: $\frac{22,4}{0,97} = 23,1$. Na Figura 3.1, este tempo está entre parênteses dentro do retângulo que representa a tarefa. Isto significa que, após uma máquina $k$ executar uma tarefa $j$, o desempenho da máquina cai e o tempo de processamento da tarefa aumenta. Desta forma, determinam-se os tempos reais de processamento de todas as tarefas e o desempenho das máquinas, após executar uma tarefa. Ainda na Figura 3.1, os valores mostrados na linha do tempo são os tempos finais do processamento das tarefas, ou seja, o instante de conclusão das tarefas. O *makespan* é o maior instante de conclusão que corresponde à máquina 2, o qual é igual a 132,8.

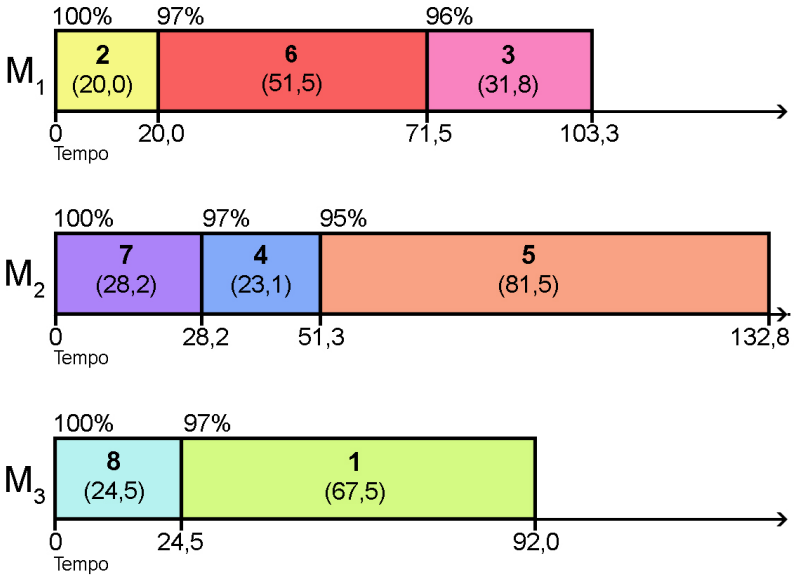


**Figura 3.1.** Tarefas ordenadas aleatoriamente

Ruiz-Torres et al. (2013) demonstrou que, para determinar o menor tempo de conclusão de um conjunto de tarefas $J$ alocado a uma máquina $k$, as tarefas deste conjunto devem estar ordenadas, em ordem decrescente, pela seguinte relação:

$$r_{jk} = p_{jk}(1 - d_{jk})/d_{jk}, \forall j \in J, \forall k \in M \tag{3.3}$$

Esta relação determina que, geralmente, as tarefas com os maiores tempos de processamento e com os menores desgastes devem ser processadas primeiras.

Na Figura 3.2 apresenta-se o sequenciamento das tarefas reordenadas pela relação $r_{jk}$. Observa-se que os tempos finais do processamento das tarefas em todas as máquinas são menores em comparação ao exemplo da Figura 3.1.

Conclui-se que, no problema abordado, basta determinar o conjunto de tarefas $J$ a serem alocadas em cada máquina. A ordenação das tarefas será determinada

pela relação $r_{jk}$.

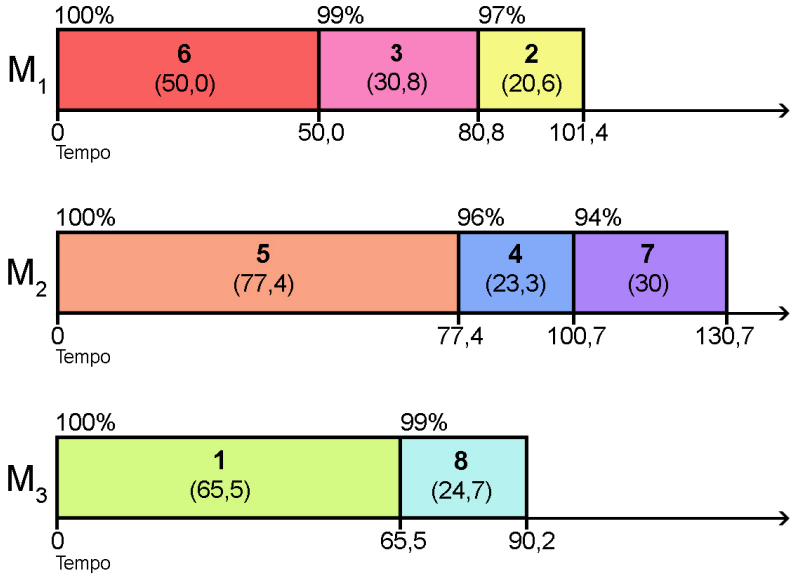


**Figura 3.2.** Tarefas ordenadas por $r_{jk}$

## 3.3 Revisão Bibliográfica

Na literatura, são encontrados vários trabalhos que abordam os problemas de programação da produção. Allahverdi et al. (1999) fazem uma análise abrangente sobre problemas de programação envolvendo tempos de preparação (custos), resumem os resultados de pesquisa da época para diferentes tipos de problemas e fornecem diretrizes para as pesquisas futuras. Os autores classificam os problemas de programação em lote e não-lotes, configuração independente e dependente da sequência. Além disso, categorizam os trabalhos de acordo com os ambientes de programação em uma única máquina, máquinas paralelas, *flowshops* e *job shops*. Allahverdi et al. (2008) abordam, novamente, sobre problemas de programação com tempos de preparação. Desta vez, categorizam ainda mais os trabalhos encontrados na literatura de acordo com os ambientes, incluindo única máquina, máquinas paralelas, *flowshop*, *flowshop* sem espera, *flowshop* flexível, *job shop*, *open shop*. Allahverdi (2015), mais uma vez, realiza um *survey* com estes problemas, envolvendo problemas estáticos, dinâmicos, deterministas e estocásticos para diferentes ambientes de programação com tempos de preparação desde a primeira pesquisa sobre o tema que surgiu em meados dos anos 1960.

O problema estudado neste trabalho é uma variante dos problemas de programação da produção e considera que ocorre um aumento do tempo de processamento das tarefas devido a deterioração da máquina. Da mesma forma que Yang (2011) e Yang et al. (2012), este trabalho segue a linha de pesquisa em que as tarefas não são por si deterioradas, mas ao invés destas, as máquinas que sofrem deteriorações (Ruiz-Torres et al., 2013). Desta forma, os tempos de processamento das tarefas podem incrementar de acordo com a ordem do processamento nas máquinas (Joo & Kim, 2015). Esse incremento é devido ao desgaste das máquinas com o passar do tempo, ou seja, a medida que as máquinas realizam as tarefas, elas sofrem degastes (Lee et al., 2010).

Os trabalhos apresentados nesta seção abordam o efeito de deterioração no ambiente de máquina paralelas. Foram encontrados trabalhos que consideram dois tipos diferentes de máquinas paralelas: as máquinas paralelas idênticas [3] e as máquinas paralelas não-relacionadas [4]. No problema abordado neste trabalho são consideradas máquinas paralelas não-relacionadas.

Existem na literatura pelo menos dois modelos de sequenciamento com efeito de deterioração do tempo de processamento das tarefas. No primeiro modelo, o tempo de processamento atual de uma tarefa é definido por uma função do seu tempo de início, isto é, $p'_{j,k} = p_{j,k} + d_{j,k} \times S_{j,k}$, onde $p'_{j,k}$, $p_{j,k}$, $d_{j,k}$ e $S_{j,k}$ são o tempo de processamento atual, o tempo de processamento normal, a razão de deterioração e o tempo de início da tarefa $j$ na máquina $k$, respectivamente. De acordo com este modelo, enquanto as tarefas esperam para serem processadas, o tempo de processamento atual das tarefas aumentam, isto é, o atraso no início do processamento de uma tarefa pode resultar em um tempo maior para processá-la (Mazdeh et al., 2010).

O segundo modelo define que o tempo de processamento da tarefa está em função das posições das tarefas na sequência da máquina, isto é, o tempo de processamento atual da tarefa pode variar dependendo da posição em que uma tarefa está no sequenciamento. Neste modelo, o tempo de processamento atual da tarefa $j$ (se processada na posição $h$ da máquina $k$) pode ser definida por: $p'_{j,k} = p_{j,k} + h \times d_{j,k}$, onde $d_{j,k}$ é o efeito de deterioração da tarefa $j$ na máquina $k$, e a posição $h$ depende do número de tarefas depois de um evento de manutenção (Yang, 2011). A atividade de manutenção é frequentemente realizada no sistema de fabricação para melhorar o efeito de deterioração, com o objetivo de manter a eficiência da produção

[3] O tempo de processamento de cada tarefa é independente da máquina na qual será alocada.

[4] Cada tarefa $j$ possui um tempo de processamento que depende da máquina $k$ na qual será alocada.

(Lee et al., 2013).

### 3.3.1 Deterioração baseada no tempo de início de processamento de uma tarefa

A seguir são apresentados os trabalhos encontrados na literatura que seguem o primeiro modelo, isto é, os trabalhos em que o tempo de processamento da tarefa é definido em função do seu tempo de início.

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

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

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

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

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

Cheng et al. (2012) estudam o problema de sequenciamento em máquinas paralelas idênticas para minimizar o maior tempo de processamento final das tarefas, em que o tempo de processamento de cada tarefa é definido por uma função do tempo de início e uma data que é individual para todas as tarefas. Um modelo de programação inteira mista é apresentado para o problema e são desenvolvidos um algoritmo modificado de busca em largura e um algoritmo VNS. Os resultados computacionais mostram que as abordagens propostas obtêm soluções próximas da ótima em um tempo computacional viável para instâncias de grande porte.

No trabalho de Huang et al. (2014), o problema de sequenciamento em máquinas paralelas idênticas considera o efeito de deterioração e o efeito de aprendizagem. O tempo de processamento de cada tarefa é dado por função do seu tempo de início no processamento. Os autores concentram-se em dois objetivos separadamente: o primeiro é minimizar uma função de custo que contém o maior tempo de processamento final e a diferença absoluta entre os tempos de processamentos finais das máquinas. O segundo objetivo é minimizar uma função de custo que contém o tempo de espera total e a diferença absoluta dos tempos de esperas das tarefas para serem processadas nas máquinas.

Já o trabalho de Guo et al. (2015) trata o problema de sequenciamento em máquinas paralelas idênticas com efeito de deterioração e a sequência dependente do tempo de configuração. O objetivo é minimizar o atraso total determinado pela sequência de tarefas nas máquinas. O tempo de processamento de cada tarefa é definido dependente do tempo de início do processamento da mesma. São propostos um modelo de programação inteira mista e um algoritmo híbrido de busca discreta *Cuckoo* para resolver o problema. Para gerar uma boa solução inicial, a heurística modificada *Biskup-Hermann-Gupta* (BHG) é incorporada à inicialização da população e um procedimento de busca local baseado no *Variable Neighbourhood Descent* (VND) é integrado ao algoritmo como uma estratégia para melhorar a qualidade das soluções. Os resultados mostram que o algoritmo proposto pode produzir soluções melhores que o software IBM CPLEX [5] em uma hora de tempo limite.

[5]CPLEX é um software utilizado para resolver modelos matemáticos com restrições lineares ou quadráticas, função objetivo linear e variáveis contínuas ou inteiras (Entringer & Arica,

### 3.3.2 Deterioração baseada na posição da tarefa no sequenciamento

Nesta Seção, são apresentados os trabalhos que seguem o segundo modelo, isto é, artigos que definem o tempo de processamento da tarefa dependente da posição em que a tarefa está no sequenciamento e de uma atividade de manutenção.

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

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

No trabalho de Yang (2011), o problema de sequenciamento em máquinas paralelas idênticas faz considerações simultâneas de atividades de manutenção e de efeitos de deterioração dependentes da posição. Assume-se que a atividade de manutenção deve ser realizada exatamente uma vez em cada máquina e logo após o término do processamento de qualquer tarefa. Além disso, após a manutenção, a máquina reverte para sua condição inicial e os efeitos de deterioração iniciam-se. O objetivo do problema é encontrar a melhor sequência das tarefas e a melhor posição das atividades de manutenção para minimizar a carga total da máquina.

Yang et al. (2012) abordam o problema de sequenciamento em máquinas paralelas não-relacionadas considerando simultaneamente efeitos de aprendizagem e atividades de manutenção. O objetivo é encontrar juntamente a frequência de manutenção, as posições ótimas para as atividades de manutenção e a sequência ótima de tarefas para minimizar a carga total da máquina. Os autores aplicaram um grupo de princípios de balanceamento para obter as posições ótimas das atividades de manutenção e o número de tarefas em cada grupo do sequenciamento de cada máquina. Além disso, um algoritmo eficiente foi desenvolvido para resolver o problema quando a frequência de manutenção das máquinas é dada.

Hsu et al. (2013) consideram o problema de sequenciamento em máquinas paralelas não-relacionadas com efeitos de aprendizagem e atividades de manuten-

2013).

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

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

O trabalho de Gara-Ali et al. (2014) introduz um modelo geral para o problema de sequenciamento em máquinas paralelas não-relacionadas com o efeito de deterioração dependente da posição e de múltiplas atividades de manutenção em cada máquina. O objetivo do problema é encontrar juntamente a posição ótima da atividade de manutenção, as frequências das manutenções e o sequenciamento ótimo de tarefas para minimizar os critérios de desempenho. Os autores apresentam uma abordagem geral para resolver o modelo sob vários critérios de desempenho e desenvolvem um algoritmo em tempo polinomial para o problema quando o número de máquinas é fixado.

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

Os trabalhos apresentados nesta seção abordam problemas com diferentes características, se diferenciando principalmente na forma de definir o tempo de processamento atual da tarefa (considerando o efeito de deterioração), nos tipos de máquinas paralelas e nos objetivos a serem alcançados.

Ruiz-Torres et al. (2013) aborda sobre um diferente problema de sequenciamento de tarefas com efeito de deterioração. Neste problema, a deterioração de cada máquina está em função da sequência de tarefas que tem sido previamente processada pela máquina. A deterioração da máquina produz incrementos nos tempos de processamentos das tarefas. Sendo assim, o tempo de processamento de uma tarefa

em uma máquina específica depende das tarefas previamente processadas nesta máquina. O objetivo deste problema é minimizar o *makespan*. Os autores mostram que para uma única máquina o problema pode ser resolvido em tempo polinomial e que para duas ou mais máquinas o problema é considerado NP-difícil. Para este último caso, os autores desenvolvem um conjunto de listas de algoritmos de sequenciamento para construir a solução inicial e algoritmos baseados na meta-heurística *Simulated Annealing* que resolvem o problema para um grande número de instâncias.

Neste trabalho, é estudado o problema de sequenciamento de tarefas em máquina paralelas não-relacionadas formulado por Ruiz-Torres et al. (2013). De acordo com o que se sabe, este problema tem sido estudado somente por esses autores.

# Capítulo 4

# Modelagens Matemáticas do Problema $Rm|Sdd|C_{max}$

Neste capítulo são apresentados dois modelos matemáticos para o problema $Rm|Sdd|C_{max}$. O primeiro é um modelo de programação não-linear inteiro e o segundo é um modelo de programação inteira mista baseado na geração de padrões.

## 4.1 Modelo de programação não-linear inteiro

Esta seção apresenta um modelo de programação não-linear inteiro do problema $Rm|Sdd|C_{max}$, proposto em Ruiz-Torres et al. (2013). Neste modelo, as seguintes variáveis de decisão são definidas:

- $C_{max}$ define o *makespan*.
- $q_{kh}$ é o desempenho da máquina $k$ após executar a tarefa da posição $h$.
- $x_{jkh} = \begin{cases} 1 & \text{, se a tarefa } j \text{ é atribuída para a máquina } k \text{ na posição } h, \\ 0 & \text{, caso contrário.} \end{cases}$
- Denota-se $G$ como o conjunto das possíveis posições $h$.

O modelo matemático do problema $Rm|Sdd|C_{max}$ é dado por:

$$\min C_{max} \tag{4.1}$$

$$\sum_{j \in N} x_{jkh} \leq 1, \forall h \in G, k \in M \tag{4.2}$$

$$\sum_{h \in G} \sum_{k \in M} x_{jkh} = 1, \forall j \in N \tag{4.3}$$

$$\sum_{j \in N} \sum_{h \in G} \frac{p_{jk}}{q_{kh}} \times x_{jkh} \leq C_{max}, \forall k \in M \tag{4.4}$$

$$\sum_{l \in N} x_{lk(h-1)} \geq x_{jkh}, \forall j \in N, k \in M, h \in G \backslash \{1\} \tag{4.5}$$

$$\sum_{j \in N} (1 - d_{jk}) \times q_{k(h-1)} \times x_{jk(h-1)} = q_{kh}, \forall h \in G \backslash \{1\}, k \in M \tag{4.6}$$

$$q_{k1} = 1, \forall k \in M \tag{4.7}$$

$$x_{jkh} \in \{0,1\}, \forall j \in N, k \in M, h \in G \tag{4.8}$$

A função objetivo (4.1) minimiza o *makespan* ($C_{max}$). A Restrição (4.2) garante que cada posição em cada máquina pode ter no máximo uma tarefa atribuída e a Restrição (4.3) garante que cada tarefa deve ser atribuída para exatamente uma posição em uma máquina. A Restrição (4.4) estabelece que a soma total dos processamentos em cada máquina não deve ser maior que $C_{max}$. A Restrição (4.5) garante atribuições de tarefas contínuas. As Equações (4.6) e (4.7) definem o nível de desempenho em cada posição da máquina, sendo que para a primeira posição, todas as máquinas têm desempenho igual a um (100% de produtividade). Por fim, a Equação (4.8) define o domínio das variáveis $x_{jkh}$.

Pode-se perceber que este modelo de programação inteira é não-linear devido a expressão $\frac{x_{jkh}}{q_{kh}}$ na Restrição (4.4) e a expressão $q \times x$ na Equação (4.6). Esta expressão não é tão simples de linearizá-la, não sendo possível resolver o modelo através do software IBM CPLEX. Sendo assim, propomos uma nova formulação para o problema em questão baseada no modelo de geração de padrões.

## 4.2 Modelo Baseado na Geração de Padrões

O problema de sequenciamento abordado neste trabalho pode ser formulado considerando os possíveis padrões (subconjuntos) de tarefas para cada máquina.

Existem alguns problemas de otimização combinatória cujos modelos podem ser baseados em padrões, tal como, o problema de empacotamento em mochilas (*Bin Packing*), o qual consiste em determinar o número mínimo de mochilas da mesma capacidade $Q$ que empacotem $n$ itens de peso $p_j, j = 1, ..., n$ (Johnson, 1974), (Garey & Johnson, 1985), (Dyckhoff, 1990), (Martello & Toth, 1990), (Cheng et al., 1994).

Outro problema similar é o problema de corte e empacotamento unidimensional, que consiste em minimizar o número de barras de tamanho único $L$ a serem cortadas para a produção de itens de tamanhos menores $l_1, l_2$,...,$l_m$ com demandas, $d_1, d_2$,...,$d_m$, respectivamente. Nesta formulação, são descritos os possíveis padrões de corte, em que: $A_p = (a_{1p}, a_{2p}...a_{jp}...a_{mp})$ é o vetor que representa o padrão de corte $p$ e, o elemento $a_{jp}$ é o número de itens de tamanho $l_j$ contidos no padrão de corte $p$ (Gilmore & Gomory, 1961).

Baseando-se nos modelos matemáticos dos problemas citados anteriormente, apresentamos a seguir o modelo de programação inteira mista (MPIM) proposto neste trabalho. Neste modelo, usam-se os seguintes parâmetros:

- $S \subseteq N$ como um subconjunto de tarefas (denominado padrão).
- $T = \{S : S \subseteq N\}$ como a família que contém todos os padrões.
- Para cada padrão $S \in T$ e máquina $k \in [1,...,m]$, denota-se por $C(S,k)$ o tempo de conclusão das tarefas da máquina $k$, no caso em que todas as tarefas em $S$ são atribuídas para esta máquina.
- Para cada padrão $S \in T$ e $k \in [1,...,m]$ foi introduzida uma variável binária $X_{S,k}$, definida como segue:

  $$X_{S,k} = \begin{cases} 1 & \text{, se o conjunto de tarefas } S \text{ é atribuída à máquina } k, \\ 0 & \text{, caso contrário.} \end{cases}$$

  onde, $S \in T$; $k \in 1,...,m$.

O modelo de programação inteira mista (MPIM) para o problema $Rm|Sdd|C_{max}$ é dado como segue:

$$\min C_{max} \tag{4.9}$$

$$\sum_{S \in T} C(S,k) X_{S,k} \leq C_{max} \tag{4.10}$$

$$\sum_{k \in m} \sum_{S \in T : j \in S} X_{S,k} = 1, \forall j \in N \tag{4.11}$$

$$\sum_{S \in T} X_{S,k} = 1, \forall k \in m \tag{4.12}$$

$$X_{S,k} \in \{0,1\}, \forall S \in T, k \in M \tag{4.13}$$

A função objetivo (4.9) minimiza o *makespan*, o qual é definido pela Restrição (4.10) como o máximo tempo de conclusão das tarefas entre todas as máquinas. A Restrição (4.11) impõe que, para cada tarefa $j$, exatamente um padrão que contém $j$ é selecionado, isto é, sequenciado em uma máquina. Inversamente, a Restrição (4.12) define que cada máquina $k$ deve receber exatamente um padrão. Por fim, a Restrição (4.13) impõe que a variável $X$ é binária.

O modelo apresentado consiste em determinar o melhor subconjunto de tarefas para cada máquina. Dado que cada máquina só pode conter um subconjunto, e uma tarefa pode estar contida em apenas um subconjunto, busca-se a melhor combinação de tarefas em cada subconjunto a fim de minimizar o tempo de conclusão do processamento das tarefas em cada máquina, e consequentemente, minimizar o *makespan*.

Para determinar o tempo de conclusão do processamento das tarefas na máquina são utilizadas as mesmas características definidas na Seção 3.2, ou seja, o desempenho da máquina (Equação 3.1) e o tempo de processamento real da tarefa (Equação 3.2). Desta forma, calcula-se o tempo de conclusão para cada subconjunto de tarefas que é atribuído a uma máquina e define-se o *makespan*.

Um maior detalhamento da forma como os padrões são representados e da execução do MPIM no CPLEX está descrito na Seção 7.1, onde são explicados os experimentos com o MPIM.

# Capítulo 5

# Meta-heurísticas propostas para o Problema $Rm|Sdd|C_{max}$

Neste capítulo, estão descritas as quatro heurísticas usadas para a resolução do problema $Rm|Sdd|C_{max}$. A primeira é baseada na meta-heurística *Iterated Local Search* (ILS). A segunda foi obtida pela combinação do ILS com uma variante do método *Variable Neighborhood Descent* (VND), que utiliza uma ordenação aleatória das vizinhanças (RVND) na fase da busca local, nomeada ILS-RVND. A terceira baseada na meta-heurística *Iterated Greedy* (IG) e a última obtida pela combinação do IG com o RVND, denotada como IG-RVND.

A heurística proposta baseada na meta-heurística *Iterated Local Search* (ILS) é uma heurística de simples implementação e bastante eficiente na solução de problemas de sequenciamento de tarefas (Lourenço et al., 2003). Heurísticas baseadas na meta-heurística *Iterated Greedy* (IG) têm obtidos excelentes resultados para diversos problemas de sequenciamento (Ruiz & Stützle, 2007). De acordo com a revisão da literatura, percebe-se que tanto a meta-heurística ILS quanto a meta-heurística IG ainda não foram aplicadas para o problema em estudo.

Os dois algoritmos propostos (ILS e ILS-RVND) usam perturbação dinâmica, cujo tamanho é determinado durante o processo de busca. Da mesma forma, os algoritmos IG e IG-RVND usam um controle dinâmico da quantidade de tarefas que devem ser removidas e reinseridas, nas fases de destruição e reconstrução.

Este capítulo está dividido em quatro seções, sendo cada seção referente a uma das quatro heurísticas propostas. A Seção 5.1 exibe a descrição da heurística ILS. Dentro desta seção, são apresentadas: a heurística utilizada para construir a solução inicial dos algoritmos propostos (Subseção 5.1.1), o método de Perturbação utilizado pelos algoritmos ILS e ILS-RVND (Subseção 5.1.2), as buscas locais que são

utilizadas nas meta-heurísticas propostas (Subseção 5.1.3) e o critério de aceitação utilizado nos algoritmos propostos (Subseção 5.1.4), respectivamente. A Seção 5.2 mostra as características do algoritmo ILS-RVND e descreve o procedimento RVND utilizado como busca local nos algoritmos ILS-RVND e IG-RVND. A Seção 5.3 apresenta a heurística IG e descreve as fases de destruição e reconstrução utilizadas nos algoritmos IG e IG-RVND. A última Seção 5.4 aborda a heurística IG-RVND.

## 5.1 ILS

ILS é um meta-heurística simples e de aplicação geral que usa, iterativamente, um procedimento de perturbação como um mecanismo de diversificação, e uma busca local como uma heurística de melhoria. Em cada iteração, uma nova solução inicial é gerada realizando modificações randômicas (procedimento de perturbação), em uma boa solução encontrada anteriormente. Em vez de gerar uma nova solução inicial a partir do zero, o mecanismo de perturbação gera uma solução inicial promissora retendo parte da estrutura que fez a solução atual tornar-se uma boa solução. A solução perturbada é melhorada pela heurística de busca local, obtendo uma nova solução. Esta nova solução é aceita como a nova solução em algumas condições definidas pelos critérios de aceitação. Uma detalhada explicação da meta-heurística ILS pode ser encontrada em Lourenço et al. (2010).

O algoritmo ILS provou ser uma abordagem muito bem sucedida para resolver diferentes problemas de otimização combinatória, tais como problemas de sequenciamento (Dong et al., 2009); (Della Croce et al., 2012); (Ribas et al., 2013), problemas de atribuição quadrática (Stützle, 2006) e problemas de roteamento de veículos (Vansteenwegen et al., 2009); (Penna et al., 2013); (Vansteenwegen & Mateo, 2014).

Uma descrição geral do pseudocódigo do algoritmo ILS proposto neste trabalho é mostrado no Algoritmo 4. Este algoritmo tem quatro parâmetros de entrada: *Critério de Parada* (controla o tempo máximo de execução), $t_{min}$ (parâmetro que representa o nível mínimo de perturbação), $t_{max}$ (parâmetro que representa o nível máximo de perturbação), $\beta$ ( parâmetro que define a probabilidade de aceitação de uma solução pior). Nas linhas 2 e 3, uma solução inicial é construída e refinada pelo procedimento de busca local. Define-se $t$ como a quantidade das máquinas (em porcentagem) que são usadas no procedimento da perturbação, isto é, nível de perturbação. Na linha 4 e 5 este nível de perturbação $t$ é inicializado com o valor de $t_{min}$ e a melhor solução $S^*$ é inicializada com $S$, respectivamente. As iterações do

algoritmo ILS são executadas entre as linhas 6 e 27 até que o critério de parada seja satisfeito. Durante cada iteração, a solução corrente $S$ sofre uma perturbação (linha 7) e é melhorada pela busca local, obtendo a solução $S_2$ (linha 8). Nas linhas 9 a 12, se a função objetivo da solução $S_2$ é melhor que da melhor solução obtida $S^*$, o nível de perturbação é reinicializado com o valor mínimo ($t = t_{min}$) e $S^*$ é atualizada. Caso contrário, o nível de perturbação é incrementado (linha 14). O máximo valor que o nível de perturbação pode obter é $t_{max}$, então se o nível de perturbação alcança este limite, este deve ser reinicializado ($t = t_{min}$). Nas linhas 19 a 26 é testado o critério de aceitação. Se a função objetivo de $S_2$ é melhor que a função objetivo da solução corrente $S$, então, $S$ é atualizada . Caso contrário, o critério pode ser aceito de acordo com a probabilidade $\beta$. A melhor solução encontrada em todas as iterações é retornada (linha 28.)

Os procedimentos SOLUÇÃO_INICIAL e BUSCA_LOCAL possuem o mesmo funcionamento para todas as heurísticas propostas (ILS, ILS-RVND, IG, IG-RVND). Já a PERTURBAÇÃO é utilizada apenas nas heurísticas ILS e ILS-RVND. Estes procedimentos são explicados nas subseções a seguir.

### 5.1.1 Geração da Solução Inicial

A heurística construtiva utilizada para construir a solução inicial dos algoritmos propostos é apresentada no Algoritmo 5. Inicialmente, ordenam-se as tarefas de acordo com uma regra de prioridade, obtendo uma *Lista* de tarefas. Para cada tarefa da *Lista*, procura-se a melhor máquina para ser inserida, ou seja, a máquina que produza o menor tempo de conclusão do processamento das tarefas. As tarefas são inseridas nas máquinas nas posições determinadas pela relação $r_{jk}$ (Equação 3.3). O algoritmo finaliza quando todas as tarefas da *Lista* ordenada são inseridas nas máquinas, isto é, a *Lista* estiver vazia.

Foram implementadas 9 regras de prioridade para ordenar as tarefas, em ordem decrescente. As 8 primeiras foram propostas por Ruiz-Torres et al. (2013) e a última foi proposta neste trabalho. As regras utilizam as seguintes medidas para ordenar as tarefas:

- $p_j^{min} = min_{k \in M} \{p_{jk}\} \; \forall j \in N$ (usada na Regra 1).
- $p_j^{max} = max_{k \in M} \{p_{jk}\} \; \forall j \in N$ (usada na Regra 2)
- $d_j^{min} = min_{k \in M} \{d_{jk}\} \; \forall j \in N$ (usada na Regra 3).
- $d_j^{max} = max_{k \in M} \{d_{jk}\} \; \forall j \in N$ (usada na Regra 4).

**Algoritmo 4:** ILS ($CriterioDeParada$, $t_{min}$, $t_{max}$, $\beta$)

```
1  início
2      S ← SOLUÇÃO_INICIAL();
3      S ← BUSCA_LOCAL(S);
4      t ← t_min;
5      S* ← S;
6      enquanto Critério de Parada Não é Satisfeito faça
7          S_1 ← PERTURBAÇÃO(S, t);
8          S_2 ← BUSCA_LOCAL(S_1);
9          se f(S_2) < f(S*) então
10             S* ← S_2;
11             t ← t_min;
12         fim
13         senão
14             t ← t + incremento;
15             se t > t_max então
16                 t ← t_min;
17             fim
18         fim
19         se f(S_2) ≤ f(S) então
20             S ← S_2;
21         fim
22         senão
23             se rand(0..1) ≤ β então
24                 S ← S_2;
25             fim
26         fim
27     fim
28     retorna S*
29 fim
```

- $r_j^{min} = min_{k \in M} \{r_{jk}\} \; \forall j \in N$ (usada na Regra 5).
- $r_j^{max} = max_{k \in M} \{r_{jk}\} \; \forall j \in N$ (usada na Regra 6).
- $v_j^{min} = min_{k \in M} \{p_{jk}/(1 - d_{jk})\} \; \forall j \in N$ (usada na Regra 7).
- $v_j^{max} = max_{k \in M} \{p_{jk}/(1 - d_{jk})\} \; \forall j \in N$ (usada na Regra 8).
- $r_j^{medio} = \frac{\sum_{k \in M} p_{jk}(1-d_{jk})/d_{jk}}{m} \; \forall j \in N$ (usada na Regra 9).

A calibração dos parâmetros (Seção 6.3.1) mostra que a Regra 9 produz as melhores soluções. Portanto, esta é a regra utilizada para ordenar as tarefas na

construção da solução inicial. As outras 8 regras de prioridade são utilizadas no algoritmo *Simulated Annealing* de Ruiz-Torres et al. (2013).

**Algoritmo 5:** SOLUCAO_INICIAL()

```
1  início
2      para cada regra faça
3          Lista ← OrdenaçãoRegraDePrioridade(regra)
4          enquanto Lista ≠ ∅ faça
5              tarefa ← primeira tarefa da Lista
6              Escolher melhor máquina para a tarefa
7              Inserir tarefa na melhor posição da máquina k
8              Lista ← Lista - {tarefa}
9          fim
10         Seja S a solução obtida
11     fim
12     retorna a melhor solução S
13 fim
```

### 5.1.2 Perturbação

A perturbação utilizada nos métodos ILS e ILS-RVND é baseada em uma realocação circular de tarefas em máquinas distintas. Primeiro, um conjunto de $t$ máquinas é randomicamente escolhido: $\{M_1, M_2, ..., M_t\}$. Este conjunto deve conter a máquina com maior tempo de conclusão das tarefas. No próximo passo, uma tarefa da máquina $M_1$ é inserida na máquina $M_2$, uma tarefa da máquina $M_2$ é inserida na máquina $M_3$, assim por diante até que uma tarefa da máquina $M_t$ é inserida na máquina $M_1$. As tarefas de cada máquina são escolhidas aleatoriamente. Este tipo de movimento é chamado de *Ejection Chain* e é utilizado em outros tipos de problemas (Assis, 2012).

O parâmetro $t$ define o tamanho da perturbação (ou nível de perturbação). No algoritmo ILS, o tamanho da perturbação é modificado durante o processo de busca. O parâmetro $t$ varia no intervalo de $[t_{min}, t_{max}]$ e é inicializado com o valor mínimo $t_{min}$. Se a melhor solução atual não é melhorada, $t$ é incrementado por um. Se esta melhor solução é melhorada ou o tamanho da perturbação excede o valor máximo ($t_{max}$), $t$ é reiniciado com o valor mínimo.

A Figura 5.1 mostra o movimento de perturbação para $L = \{M_1, M_2, M_3\}$. Na primeira solução, apresentada na Figura 5.1 (a), as realocações que são feitas entre as máquinas são ilustradas. Na Figura 5.1 (b), a tarefa 6 é removida da máquina

1 e inserida na máquina 2. Na Figura 5.1 (c), a tarefa 5 é removida da máquina 2 e inserida na máquina 3. Por fim, na Figura 5.1 (d), a tarefa 1 é removida da máquina 3 e inserida na máquina 1.

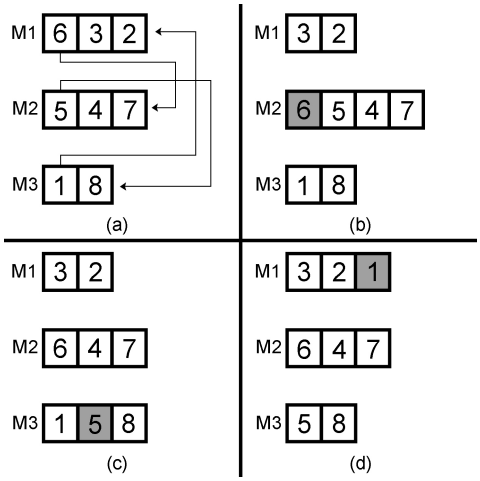


**Figura 5.1.** Exemplo de perturbação

### 5.1.3 Busca Local

O método de busca local tem como objetivo melhorar a solução inicial e as soluções obtidas pelo procedimento de perturbação. Este método tenta melhorar a solução através de uma busca em vizinhança. A vizinhança de uma solução contém todas as soluções alcançadas através de movimentos individuais em sua estrutura. Neste conjunto, a solução melhor do que a solução atual é selecionada e o processo continua até que um ótimo local seja atingido.

As estruturas de vizinhanças utilizadas na busca local são definidas a seguir:

- $N_1$ (*Troca*): é a vizinhança formada por soluções obtidas realizando trocas de tarefas que estão na máquina com maior tempo de conclusão e tarefas das outras máquinas.

- $N_2$ (*Inserção*): esta vizinhança é formada por soluções obtidas retirando-se tarefas da máquina com maior tempo de conclusão e inserindo-as em outra máquina.

Para melhor entendimento das estruturas de vizinhanças, as Figuras 5.2 e 5.3 mostram o funcionamento de $N_1$ e $N_2$, respectivamente. Considere a Figura 5.2 (a) como a solução inicial para a estrutura $N_1$. No exemplo apresentado na Figura

5.2, o movimento de troca inicia-se da tarefa 5 da máquina com maior tempo de conclusão das tarefas, neste caso, a máquina 2. É feito, então, a primeira iteração dos movimentos de $N_1$, isto é, a troca da tarefa 5 com todas as outras tarefas das máquinas restantes. Na Figura 5.2 (b), a tarefa 5 da máquina 2 é trocada com a tarefa 6 da máquina 1; Na Figura 5.2 (c), a tarefa 5 da máquina 2 é trocada com a tarefa 3 da máquina 1; a tarefa 5 da máquina 2 é trocada com a tarefa 2 da máquina 1 (Figura 5.2 (d)); Nas Figuras 5.2 (e) e 5.2(f), a tarefa 5 da máquina 2 é trocada com a tarefa 1 e com a tarefa 8 da máquina 3, respectivamente. Para todas as estruturas de vizinhanças, as tarefas são realocadas nas máquinas de acordo com a melhor posição definida por $r_{jk}$ (Equação 3.3).

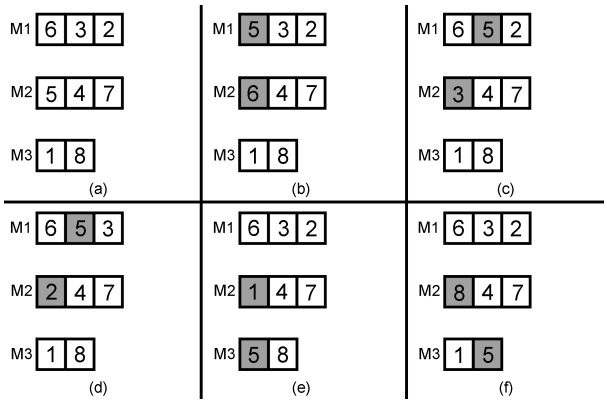


**Figura 5.2.** Exemplo da estrutura de vizinhança $N_1$ (*Troca*)

A Figura 5.3 apresenta o exemplo de uma iteração da estrutura de vizinhança $N_2$, a qual, uma tarefa da máquina com maior tempo de conclusão das tarefas é removida desta máquina e inserida nas outras. Considere a Figura 5.3 (a) como a solução inicial para a estrutura $N_2$. Escolhe-se a tarefa 5 da máquina 2 na solução inicial para ser removida. Na Figura 5.3 (b) esta tarefa é inserida na máquina 1. E na Figura 5.3 (c) esta tarefa é inserida na máquina 3.

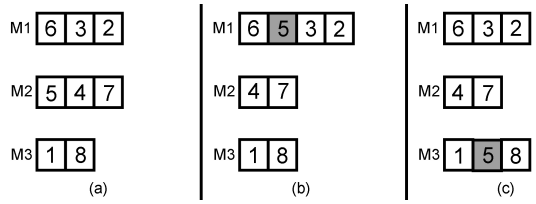


**Figura 5.3.** Exemplo da estrutura de vizinhança $N_2$ (*Inserção*)

O número total de soluções analisado pela vizinhança $N_1$ é dado por $n_s(n-n_s)$,

e para $N_2$ é igual a $n_s(m-1)$, em que $n_s$ representa o total de tarefas na máquina com o maior tempo de conclusão (Ruiz-Torres et al., 2013).

Os passos da busca local utilizada pelo ILS são apresentados no Algoritmo 6. O procedimento recebe uma solução $S$. A partir desta solução, determinam-se as melhores soluções vizinhas nas vizinhanças $N_1$ e $N_2$. A partir dessas duas soluções vizinhas, seleciona-se a melhor ($S_{melhor}$). Esta solução é comparada com a solução atual $S$. Caso exista melhoria, atualiza-se $S$ e repete-se o procedimento. Caso contrário, o melhor vizinho encontrado é retornado e o procedimento é finalizado.

**Algoritmo 6:** BUSCA_LOCAL($S$)

```
1  início
2  |  enquanto melhorou = true faça
3  |  |  S_Vizinho1 ← N_1(S) // retorna o melhor vizinho ∈ N_1
4  |  |  S_Vizinho2 ← N_2(S) // retorna o melhor vizinho ∈ N_2
5  |  |  S_melhor ← melhor(S_Vizinho1, S_Vizinho2)
6  |  |  se f(S_melhor) < f(S) então
7  |  |  |  S ← S_melhor
8  |  |  fim
9  |  |  senão
10 |  |  |  melhorou ← false
11 |  |  fim
12 |  fim
13 |  retorna S
14 fim
```

### 5.1.4 Critério de Aceitação

O critério de aceitação utilizado no algoritmo ILS tem como objetivo obter um balanceamento entre intensificação e diversificação. Uma intensificação muito forte é alcançada se apenas soluções melhores que a solução atual são aceitas. Por outro lado, uma grande diversificação é atingida quando soluções são aceitas aleatoriamente, independente do valor da função objetivo.

No pseudocódigo do método ILS (Algoritmo 4), o critério de aceitação define se o ótimo local em $S_2$ será aceito, ou seja, se a solução $S_2$ é melhor que a solução atual $S$ , então ela é aceita ($S_2$ substitui $S$). No entanto, mesmo havendo um piora da solução $S_2$, ela pode ser aceita com uma pequena probabilidade $\beta$. Desta forma, o critério de aceitação decide qual solução é a próxima a ser aplicada no procedimento de perturbação.

## 5.2 ILS-RVND

O algoritmo ILS-RVND é similar ao algoritmo ILS, utiliza os mesmos procedimentos para gerar a solução inicial e a perturbação. A diferença entre estes algoritmos está no procedimento de busca local. No algoritmo ILS-RVND, a busca local é realizada pelo procedimento RVND, o qual utiliza uma ordenação de vizinhança randômica e inclui novas estruturas de vizinhanças no procedimento de busca local. O procedimento RVND é explicado a seguir.

### 5.2.1 Procedimento RVND

Esta seção descreve o procedimento de busca local RVND, no qual a ordem das vizinhanças serem exploradas é determinada aleatoriamente. O procedimento RVND utiliza 5 vizinhanças. As duas primeiras ($N_1$ e $N_2$) são as mesmas utilizadas na busca local (Seção 5.1.3), enquanto as outras três vizinhanças foram implementadas apenas para este procedimento. Apresentamos as estruturas de vizinhanças adicionais na sequência.

- $N_3$ (*Inserção2*): formada por soluções obtidas retirando duas tarefas da máquina com maior tempo de conclusão e inserindo-as em outra máquina qualquer.
- $N_4$ (*Troca2*): formada por soluções obtidas realizando trocas de duas tarefas que estão na máquina com maior tempo de conclusão por duas tarefas de outras máquinas.
- $N_5$ (*Troca2Por1*): formada por soluções obtidas realizando trocas de duas tarefas que estão na máquina com maior tempo de conclusão por uma tarefa de outras máquinas.

O procedimento do RVND considera todas as possíveis combinações que podem ser feitas entre as duas vizinhanças da busca local ($N_1$ e $N_2$) e as outras 3 vizinhanças($N_3$, $N_4$ e $N_5$). Desta forma, para definir quais vizinhanças são utilizadas no procedimento RVND são testadas as seguintes combinações: ($N_1$, $N_2$ e $N_3$); ($N_1$, $N_2$ e $N_4$); ($N_1$, $N_2$ e $N_5$); ($N_1$, $N_2$, $N_3$ e $N_4$); ($N_1$, $N_2$, $N_3$ e $N_5$); ($N_1$, $N_2$, $N_4$ e $N_5$) e ($N_1$, $N_2$, $N_3$, $N_4$ e $N_5$).

Através da calibração de parâmetros (Seção 6.3), percebeu-se que com a combinação das vizinhanças $N_1$, $N_2$ e $N_3$ encontram-se as melhores soluções para o algoritmo ILS-RVND, e que a combinação das três vizinhanças ($N_1$, $N_2$ e $N_4$)

encontram-se as melhores soluções para o algoritmo IG-RVND. Sendo assim, apenas estas vizinhanças são utilizadas no procedimento RVND destes algoritmos.

O procedimento RVND é apresentado no Algoritmo 7, o qual recebe uma solução $S$ como entrada. Inicialmente, forma-se a lista $LV$ com as estruturas de vizinhanças $N_i$ (para $i = 1, ..., k$) a serem utilizadas na busca. Aleatoriamente, escolhe-se uma vizinhança $N_i$ da lista e determina-se a melhor solução desta vizinhança ($S_{melhor}$). Se a solução $S_{melhor}$ for melhor do que a solução corrente, a solução corrente $S$ recebe a melhor solução e a lista $LV$ é reinicializada com todas as vizinhanças. Caso contrário, remove-se a vizinhança $N_i$ da lista $LV$. O algoritmo finaliza quando a lista de vizinhanças estiver vazia, isto é, quando analisam-se todas as vizinhanças e não é encontrada uma solução melhor.

**Algoritmo 7:** RVND($S$)

```
1  início
2  |  Lista de Vizinhanças: LV ← {N_1, N_2,..., N_k}
3  |  enquanto LV ≠ ∅ faça
4  |  |  i ← Random(1, 2,..., k)
5  |  |  S_melhor ← N_i(S)
6  |  |  se f(S_melhor) < f(S) então
7  |  |  |  S ← S_melhor
8  |  |  |  LV ← {N_1, N_2,..., N_k}
9  |  |  fim
10 |  |  senão
11 |  |  |  LV ← LV − {N_i}
12 |  |  fim
13 |  fim
14 |  retorna S
15 fim
```

## 5.3 IG

A meta-heurística *Iterated Greedy* (IG) é uma simples e efetiva meta-heurística que foi introduzida por Ruiz & Stützle (2007). O IG apresenta duas fases: destruição e reconstrução. A IG inicia com uma solução inicial que iterativamente é modificada por uma sequência de procedimento de destruição e reconstrução. Na fase de destruição, algumas tarefas são extraídas aleatoriamente da solução, levando a uma solução parcial. Na fase de reconstrução, estas tarefas são reinseridas uma por uma, de forma gulosa, cada uma na melhor posição da solução parcial, até reconstruir

uma solução completa. Estas duas fases constituem a diversificação da solução. Em seguida, um procedimento de busca local é aplicado para melhorar a solução reconstruída. Ao final de cada iteração, um critério de aceitação define se a nova solução irá substituir a solução atual. O procedimento retorna, ao fim, a melhor solução encontrada.

O pseudocódigo do IG é apresentado no Algoritmo 8, cujos parâmetros iniciais são dados por *CritérioDeParada*, $d_{min}$, $d_{max}$, $\beta$ e $p$. O primeiro parâmetro armazena o tempo máximo de execução utilizado como critério de parada do algoritmo. Os parâmetros $d_{min}$ e $d_{max}$ são usados para definir a quantidade mínima e máxima de tarefas que são removidas e reinseridas durante as fases de destruição e reconstrução, respectivamente. O parâmetro $\beta$ define a probabilidade de aceitação de uma solução pior. O último parâmetro, $p$ define o percentual das máquinas utilizadas na destruição e reconstrução da solução que são escolhidas aleatoriamente ou que é a máquina que possui o maior tempo de conclusão das tarefas.

As linhas 2 e 3 do Algoritmo 8 geram uma solução inicial $S$ e esta solução é refinada por um método de busca local. Nas linhas 4 e 5, a variável $d$ é inicializada com o valor de $d_{min}$ e a melhor solução ( $S^*$) é inicializada como $S$.

Nas linhas de 7 e 8, o IG tenta melhorar iterativamente a solução atual através de dois procedimentos. O primeiro deles consiste em destruir a solução atual removendo uma certa quantidade $d$ de seus elementos e, logo após, a solução é reconstruída, por meio da reinserção gulosa dos elementos removidos. Na sequência aplica-se uma busca local sobre a solução reconstruída $S_1$ a fim de encontrar um novo ótimo local $S_2$.

Nas linhas 9 a 12, se o valor da função objetivo da solução $S_2$ é o menor, a variável $d$ é reinicializada com o valor mínimo ($d_{min}$) e a melhor solução ($S^*$) é atualizada . Caso contrário, a quantidade de tarefas que são removidas e reinseridas ($d$) é incrementada (linha 14). Nas linhas 19 a 21, a solução corrente $S$ é atualizada se a função objetivo de $S_2$ melhorou a função objetivo de $S$. Caso a solução $S_2$ não melhore a solução corrente $S$, um critério de aceitação é aplicado, de acordo com a probabilidade $\beta$ (linhas 22 a 25). Finalmente na linha 28, a melhor solução encontrada em todas as iterações é retornada.

Os procedimentos SOLUÇÃO_INICIAL e BUSCA_LOCAL são os mesmos que foram utilizados para o método ILS, explicados nas Seções 5.1.1 e 5.1.3, respectivamente. Já o procedimento DESTRUIÇÃO_RECONSTRUÇÃO é apresentado na subseção a seguir.

**Algoritmo 8:** IG($CriterioDeParada, d_{min}, d_{max}, \beta, p$)

```
1  início
2      S ← SOLUÇÃO_INICIAL( )
3      S ← BUSCA_LOCAL(S)
4      d ← d_min
5      S* ← S
6      repita
7          S_1 ← DESTRUIÇÃO_RECONSTRUÇÃO(S, d, p)
8          S_2 ← BUSCA_LOCAL(S_1)
9          se f(S_2) < f(S*) então
10             S* ← S_2
11             d ← d_min
12         fim
13         senão
14             d ← d + incremento;
15             se d > d_max então
16                 d ← d_min;
17             fim
18         fim
19         se f(S_2) ≤ f(S) então
20             S ← S_2;
21         fim
22         senão
23             se rand(0..1) ≤ β então
24                 S ← S_2;
25             fim
26         fim
27     até CriterioDeParada ser satisfeito;
28     retorna S*
29 fim
```

### 5.3.1 Fases de Destruição e Reconstrução

O Algoritmo 9 demonstra o funcionamento das fases de destruição e reconstrução do procedimento IG. Este algoritmo possui os seguintes parâmetros: solução inicial ($S$), o número de tarefas a ser removidas ($d$) e a probabilidade de as máquinas utilizadas serem escolhidas aleatoriamente ou de ser a máquina com o maior tempo de conclusão do processamento das tarefas ($p$). Na primeira linha do Algoritmo 9, $J_r$ é definido como o conjunto que armazena as tarefas que são removidas.

As linhas 3 a 14 mostram a fase de destruição. Em cada iteração, escolhe-se um valor $r$ aleatoriamente, verifica-se se este valor é menor ou igual a $p$. Caso seja, a máquina que terá uma tarefa removida é escolhida aleatoriamente. Caso

**Algoritmo 9:** DESTRUIÇÃO_RECONSTRUÇÃO$(S, d, p)$

```
1  início
2      Conjunto de tarefas a serem removidas: J_r ← {}
3      para i = 1 até d faça
4          r ← Random(0, ..., 100)
5          se r ≤ p então
6              k ← máquina selecionada aleatoriamente
7          fim
8          senão
9              k ← máquina com o maior tempo de conclusão das tarefas
10         fim
11         j ← tarefa selecionada aleatoriamente da máquina k
12         S ← remove tarefa j da máquina k
13         J_r ← J_r + {j}
14     fim
15     para cada j de J_r faça
16         S ← insere a tarefa j na máquina k que produza o menor tempo de
           conclusão das tarefas
17     fim
18     retorna S
19 fim
```

contrário, define-se $k$ como a máquina que contém o maior tempo de conclusão do processamento das tarefas. Em seguida, seleciona-se uma tarefa $j$ de forma aleatória na máquina $k$. Remove-se esta tarefa $j$ da máquina $k$ e insere $j$ no conjunto de tarefas removidas $J_r$, o qual não pode conter tarefas repetidas. Este procedimento é repetido até que $i$ seja igual ao número definido de tarefas a serem removidas ($d$).

As linhas 15, 16 e 17 mostram a fase de reconstrução, em que cada tarefa $j$ é selecionada do conjunto $J_r$ e reinserida na máquinas $k$ que produza o menor tempo de conclusão das tarefas, de acordo com a melhor posição conhecida, dado pela relação $r_{jk}$ (Equação 3.3). Esse processo é repetido até que o conjunto $J_r$ fique vazio, ou seja, quando todas as tarefas de $J_r$ estejam reinseridas na solução $S$, obtendo assim uma nova solução completa.

## 5.4 IG-RVND

A heurística proposta IG-RVND difere da IG (Algoritmo 8) apenas no procedimento de busca local. No IG-RVND, a busca local é realizada pelo procedimento RVND, descrito na Seção 5.2.1.

# Capítulo 6

# Experimentos Computacionais - Parte 1

Os experimentos computacionais foram divididos em duas partes para melhor entendimento. A primeira parte está dividida em quatro Seções. Na Seção 6.1, as instâncias dos problemas-teste são mostradas. A Seção 6.2 apresenta a métrica utilizada para avaliar os algoritmos heurísticos. A Seção 6.3 apresenta as calibrações dos parâmetros utilizados nos algoritmos propostos. Por fim, a Seção 6.4 explica a reimplementação do algoritmo proposto por Ruiz-Torres et al. (2013). Estes autores desenvolveram um algoritmo *Simulated Annealing* (SA), denominado SA*, para resolver o problema abordado neste trabalho.

Os testes foram feitos em um computador Intel Core i7, 4GHz, com 32 GB de RAM, sistema operacional Ubuntu 14.04 - 64 bits. Os algoritmos propostos foram implementados em C++ e compilados com Netbeans IDE 8.0.2. Os algoritmos foram executados com uma simples *thread*.

## 6.1 Instâncias do problema

Nos experimentos foram utilizadas 1800 instâncias classificadas em dois grupos, variando de acordo com o número de tarefas ($n$), o número de máquinas ($m$), o valor do tempo de processamento das tarefas ($p_{jk}$) e valor dos efeitos de desgaste das máquinas ($d_{jk}$). As subseções a seguir apresentam estes dois grupos de instâncias.

### 6.1.1 Instâncias (de Médio Porte) da literatura

Um grupo das instâncias de médio porte, para o problema $Rm|Sdd|C_{max}$ foi produzido por Ruiz-Torres et al. (2013) e estão disponíveis em: `http://ruiz-torres.uprrp.edu/dm/`. Estas instâncias possuem número de tarefas $n \in \{20,35,50\}$ e número de máquinas $m \in \{4,7,10\}$.

O tempo de processamento das tarefas ($p_{jk}$) foram gerados aleatoriamente, com distribuição uniforme, sobre dois tipos de intervalos: [1; 100] e [100; 200]. Os efeitos de desgaste das máquinas produzidas pelas tarefas ($d_{jk}$) também foram gerados aleatoriamente, com distribuição uniforme, sobre dois tipos de intervalos: [1%; 5%] e [5%; 10%]. Combinando os parâmetros: $n$, $m$, os dois tipos de $p_{jk}$ e os dois tipos de $d_{jk}$, têm-se 36 configurações possíveis. Para cada configuração, 25 instâncias foram geradas, num total de 900 instâncias.

Este grupo foi usado nos testes do Modelo de Programação inteira Mista (MPIM) e nos testes dos algoritmos heurísticos.

### 6.1.2 Geração de instâncias de Grande Porte

Neste trabalho, instâncias de grande porte foram geradas para o problema $Rm|Sdd|C_{max}$. Este grupo possui número de tarefas $n \in \{80,100,150\}$ e número de máquinas $m \in \{5,10,20\}$.

Os tempos de processamento ($p_{jk}$) e os efeitos de desgaste ($d_{jk}$) das máquinas foram produzidas da mesma forma que Ruiz-Torres et al. (2013), num total de 900 instâncias.

## 6.2 Métrica para avaliação dos algoritmos heurísticos

Tanto na calibração de parâmetros das heurísticas quanto nos experimentos de comparação dos algoritmos (Seções 7.2 e 7.3), foi utilizada a medida do Desvio Percentual Relativo (RPD do inglês: *Relative Percentage Deviation*). Para uma dada instância, o RPD é calculado da seguinte forma (Vallada et al., 2008):

$$RPD = \frac{f_{\text{media}} - f_{\text{Best}}}{f_{\text{Best}}} \times 100\% \tag{6.1}$$

em que $f_{\text{media}}$ é a média das funções objetivos de todas as execuções do algoritmo e $f_{\text{Best}}$ é a melhor solução obtida entre todos os métodos comparados. É importante

ressaltar que quanto menor valor do RPD, melhor será a qualidade da solução.

Nem sempre comparar com a solução ótima é viável, uma vez que o valor ótimo pode não ser encontrado para todas as instâncias utilizadas. Por isso, são realizadas comparações com a melhor solução conhecida.

Porém, neste trabalho (Seção 7.1.3), também é feita a análise considerando algumas soluções ótimas. Estas soluções foram obtidas através da resolução do Modelo de Programação Inteira Mista (MPIM) no software IBM CPLEX versão 12.5.

## 6.3 Calibração dos parâmetros dos algoritmos heurísticos

Assim como no trabalho de Jacob et al. (2014), para a calibrar os parâmetros, considerou-se um conjunto independente de 216 novas instâncias: 108 instâncias de médio porte e 108 instâncias de grande porte. Em cada grupo, foram geradas 2 instâncias de cada combinação possível entre o número de tarefas ($n$), número de máquinas ($m$) e os valores dos intervalos dos tempos de processamento ($p_{jk}$) e os efeitos de desgaste ($d_{jk}$), como descrito na Seção 6.1. Isto foi feito visto que ao utilizar as mesmas instâncias para calibração dos parâmetros e comparação dos métodos pode-se levar a resultados tendenciosos.

Na calibração, os algoritmos foram executados 10 vezes para cada instância. Os resultados obtidos são analisados utilizando o RPD, o qual é calculado considerando a média do *makespan* das 10 execuções para cada uma das instâncias. Para validar os resultados obtidos pelos algoritmos e verificar se as diferenças observadas são estatisticamente significantes, foi realizada uma Análise de Variância (ANOVA) paramétrica. Ainda, foram verificadas as três pressuposições da ANOVA para que os resultados do teste sejam estatisticamente válidos: normalidade, igualdade de variância e a independência dos resíduos.

Os parâmetros analisados são:

- *Critério de Parada*: define o tempo máximo de execução de todos os algoritmos propostos;
- *Regra de Prioridade*: utilizada na heurística construtiva para definir a ordem em que as tarefas são inseridas na solução inicial;
- $t_{min}$ e $t_{max}$: definem os níveis mínimo e máximo, respectivamente, de perturbação para os algoritmos ILS e ILS-RVND;

- $\beta$: define a probabilidade de aceitação de uma solução pior (critério de aceitação);
- *número de vizinhanças*: define quantas estruturas de vizinhanças são utilizadas no procedimento RVND.
- $d_{min}$ e $d_{max}$: determinam a quantidade mínima e máxima de tarefas, respectivamente, que são removidas e reinseridas durante as fases de destruição e reconstrução dos algoritmos IG e IG-RVND;
- $p$: define a porcentagem de escolha de máquinas selecionadas ao acaso na renovação de tarefas ou a opção pela máquina que possui o maior tempo de conclusão das tarefas (fases de destruição e reconstrução) dos algoritmos IG e IG-RVND;

Para o parâmetro *Critério de Parada*, que representa o tempo de execução para os algoritmos propostos, foi estabelecido o tempo máximo de CPU pela expressão $\frac{n}{m}$ segundos. Nesta expressão, o tempo do CPU é influenciado pelo número de tarefas ($n$) e pelo número de máquinas ($m$). Para $n$ fixado, o tempo de execução aumenta quando o valor de $m$ decresce.

Para definir o valor de *Critério de Parada*, observou-se através de testes computacionais que os melhores resultados eram alcançados quando o tempo limite de execução era proporcional ao aumento do número de tarefas e inversamente proporcional ao número de máquinas.

Nas subseções seguintes são mostrados os resultados obtidos para a calibração dos parâmetros específicos de cada algoritmo.

### 6.3.1 Análise das heurísticas construtivas para a geração de soluções iniciais

Na heurística construtiva, uma lista de tarefas ordenadas é utilizada de acordo com uma regra de prioridade. A Figura 6.1 mostra o gráfico de médias resultante do teste Tukey da Diferença Honestamente Significativa (HSD) com nível de confiança de 95% para cada uma das 9 regras de prioridades. Na Figura 6.1, pode-se ver que a regra de prioridade número 9, correspondente ao $r_j^{medio}$ produz as menores médias. Com base nesta constatação, a Regra 9 foi adotada na geração de solução inicial dos algoritmos.

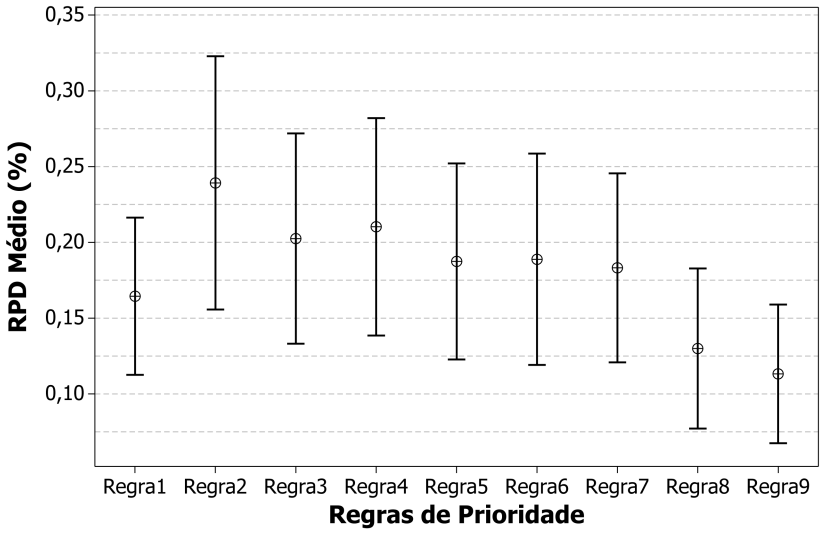


**Figura 6.1.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração das Regras de Prioridades

### 6.3.2 Calibração dos parâmetros dos algoritmos do ILS e ILS-RVND

O valor do parâmetro $t_{min}$ foi fixado em 2 (dois), pois é o número mínimo de máquinas que se deve ter para realizar o procedimento de perturbação. Os valores testados para os parâmetros estão especificados na Tabela 6.1. Observe que há um total de 9 combinações de parâmetros do ILS para avaliar. A Tabela 6.2 mostra qual é cada uma destas configurações.

**Tabela 6.1.** Conjunto de valores de parâmetros para a calibração do ILS

| Parâmetro | Valores Testados | Quantidade |
|---|---|---|
| $t_{min}$ | {2} | 1 |
| $t_{max}$ | {20%m; 70%m; 90%m} | 3 |
| $\beta$ | {0,0; 0,3; 0,6} | 3 |
| Total | | 9 |

A Figura 6.2 mostra o gráfico de médias resultante do teste HSD de Tukey, com nível de confiança de 95% para as 9 configurações testadas do algoritmo ILS.

Embora as duas melhores configurações (1 e 7) apresentem as menores médias de RPD, não são entre si estatisticamente diferentes. É possível ver pela Figura 6.2 que estas configurações apresentam diferenças significativas quando comparadas com as outras configurações, tais como as configurações 3, 5, 6, 8 e 9, pois os

**Tabela 6.2.** Configuração dos parâmetros do ILS

| Configuração | Parâmetros | | |
|---|---|---|---|
| | $t_{min}$ | $t_{max}$ | $\beta$ |
| 1 | 2 | 20%m | 0,0 |
| 2 | 2 | 20%m | 0,3 |
| 3 | 2 | 20%m | 0,6 |
| 4 | 2 | 70%m | 0,0 |
| 5 | 2 | 70%m | 0,3 |
| 6 | 2 | 70%m | 0,6 |
| 7 | 2 | 90%m | 0,0 |
| 8 | 2 | 90%m | 0,3 |
| 9 | 2 | 90%m | 0,6 |

intervalos de 1 e 7 não sobrepõem os intervalos das outras configurações.

Apesar das duas configurações (1 e 7) serem estatisticamente iguais, optou-se pelo maior valor de $t_{max}$, apenas para obter uma maior diversificação da perturbação no algoritmo ILS. Os valores correspondentes a configuração 7 são: $t_{min} = 2$; $t_{max} = 90\%$ e $\beta = 0{,}0$. Observe que, nesta configuração, o parâmetro $\beta$ possui valor 0,0, indicando que, para esta implementação, não se deve considerar soluções de qualidade inferior a solução corrente no critério de aceitação. Além disso, vale ressaltar que as três configurações que possuem menores médias (1, 4 e 7) possuem o valor de $\beta = 0$, reafirmando que não se deve aceitar soluções de qualidade inferior.

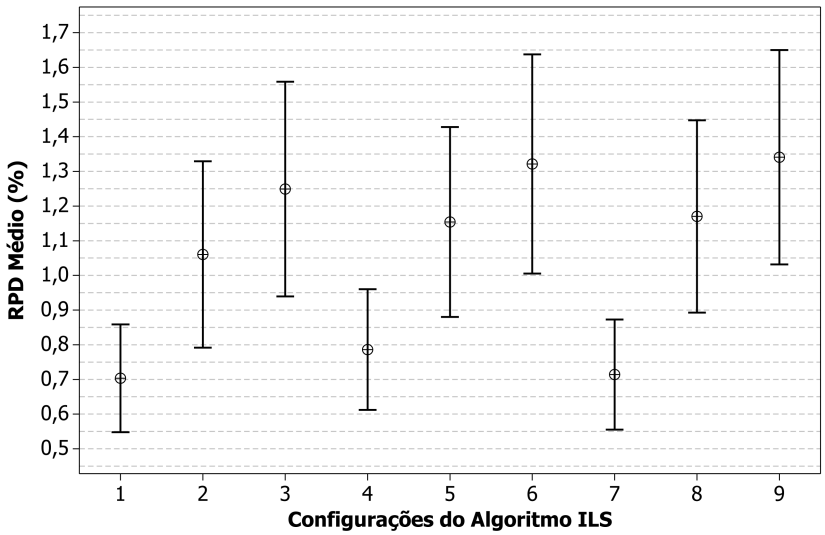


**Figura 6.2.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração dos parâmetros do ILS.

Ainda é preciso definir quais vizinhanças são utilizadas no procedimento RVND. Considerando as vizinhanças $N_1$ e $N_2$ da busca local (Seção 5.1.3) e as outras três implementadas $N_3$, $N_4$ e $N_5$ (Seção 5.2.1), obteve-se no total 5 vizinhanças para o procedimento RVND do algoritmo ILS-RVND. Para calibrar o número de vizinhanças, considerou-se a inclusão das três novas vizinhanças formando algumas das combinações possíveis entre as cinco vizinhanças, porém sempre mantendo as vizinhanças da busca local ($N_1$ e $N_2$).

A Figura 6.3 mostra o gráfico de médias resultante do teste HSD de Tukey com nível de confiança de 95% para as configurações testadas entre as vizinhanças. Percebe-se através desta figura que não existe diferença significativa entre as configurações, pois os intervalos se sobrepõem. No entanto, é possível ver nesta figura que a menor média de RPD é obtida com a combinação das três vizinhanças: $N_1$, $N_2$ e $N_3$. Sendo assim, o procedimento RVND no algoritmo ILS-RVND utiliza estas três vizinhanças.

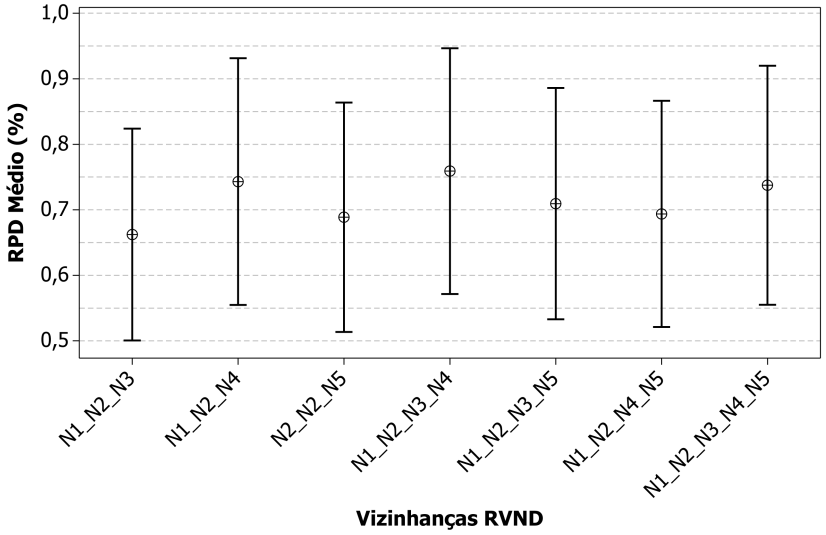


**Figura 6.3.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração das vizinhanças do procedimento RVND do algoritmo ILS-RVND.

### 6.3.3 Calibração dos parâmetros dos algoritmos do IG e IG-RVND

Como explicado na Seção 5.3, os parâmetros $d_{min}$ e $d_{max}$ estabelecem a quantidade mínima e máxima, respectivamente, de tarefas que são removidas e reinseridas nas

máquinas. O parâmetro $p$ define se as máquinas utilizadas são escolhidas aleatoriamente ou se é a máquina com o maior tempo de conclusão. Por exemplo, quando $p$ é igual a 30%, significa que a probabilidade das máquinas serem escolhidas aleatoriamente é de 30%, por outro lado, a probabilidade da máquina ser a que contém o maior tempo de conclusão das tarefas é de 70%. Além desses, o Algoritmo 8 também possui como parâmetro $\beta$, o qual define a probabilidade de aceitação de uma solução pior (critério de aceitação). O valor de $d_{min}$ foi estabelecido como 1 e os outros valores testados para os parâmetros estão especificados na Tabela 6.3. Observe que há um total de 36 combinações dos parâmetros do IG para avaliar. A Tabela 6.4 apresenta cada uma destas configurações.

**Tabela 6.3.** Conjunto de valores de parâmetros para a calibração do IG

| Parâmetro | Valores Testados | Quantidade |
|---|---|---|
| $d_{min}$ | {1} | 1 |
| $d_{max}$ | {8; 12; 16; 20} | 4 |
| $p$ | {50%; 80%; 100%} | 3 |
| $\beta$ | {0,0; 0,3; 0,6} | 3 |
| Total | | 36 |

A Figura 6.4 mostra o gráfico de médias resultante do teste HSD de Tukey com nível de confiança de 95%, considerando as configurações dos parâmetros do algoritmo IG. Nota-se que existe diferença significativa entre as configurações, pois há diversos intervalos que não se sobrepõem, mesmo que não haja uma única configuração que seja estatisticamente melhor que todas as outras. Por exemplo, existem diferenças significativas entre as configurações 9 e 13, 18 e 22, 30 e 31.

Apesar das melhores configurações (13, 22 e 31) não apresentarem diferenças estatisticamente significantes, a configuração 22 apresenta a menor média de RPD. Os valores dos parâmetros da configuração 22 são: $d_{max} = 16$; $p = 80\%$ e $\beta = 0{,}0$. Desta forma, no máximo 16 tarefas são realocadas entre as máquinas e estas possuem 80% de probabilidade de serem escolhidas aleatoriamente. É importante destacar que, assim como ocorreu para o ILS, o critério de aceitação determinou que não se deve considerar soluções de qualidade inferior a solução corrente no critério de aceitação para o algoritmo IG, uma vez que o valor de $\beta = 0{,}0$. Além disso, as configurações que possuem menores médias (10, 13, 19, 22 e 31) apresentam o valor de $\beta = 0{,}0$, mostrando que é inviável a aceitação de soluções de qualidade inferior.

Também é necessário definir quais vizinhanças são utilizadas no procedimento RVND do algoritmo IG-RVND. Ao todo obteve-se 5 vizinhanças para o procedi-

**Tabela 6.4.** Configuração dos parâmetros do IG

| Configuração | Parâmetros | | | |
|---|---|---|---|---|
| | $d_{min}$ | $d_{max}$ | $p$ | $\beta$ |
| 1 | 1 | 8 | 50%m | 0,0 |
| 2 | 1 | 8 | 50%m | 0,3 |
| 3 | 1 | 8 | 50%m | 0,6 |
| 4 | 1 | 8 | 80%m | 0,0 |
| 5 | 1 | 8 | 80%m | 0,3 |
| 6 | 1 | 8 | 80%m | 0,6 |
| 7 | 1 | 8 | 100%m | 0,0 |
| 8 | 1 | 8 | 100%m | 0,3 |
| 9 | 1 | 8 | 100%m | 0,6 |
| 10 | 1 | 12 | 50%m | 0,0 |
| 11 | 1 | 12 | 50%m | 0,3 |
| 12 | 1 | 12 | 50%m | 0,6 |
| 13 | 1 | 12 | 80%m | 0,0 |
| 14 | 1 | 12 | 80%m | 0,3 |
| 15 | 1 | 12 | 80%m | 0,6 |
| 16 | 1 | 12 | 100%m | 0,0 |
| 17 | 1 | 12 | 100%m | 0,3 |
| 18 | 1 | 12 | 100%m | 0,6 |
| 19 | 1 | 16 | 50%m | 0,0 |
| 20 | 1 | 16 | 50%m | 0,3 |
| 21 | 1 | 16 | 50%m | 0,6 |
| 22 | 1 | 16 | 80%m | 0,0 |
| 23 | 1 | 16 | 80%m | 0,3 |
| 24 | 1 | 16 | 80%m | 0,6 |
| 25 | 1 | 16 | 100%m | 0,0 |
| 26 | 1 | 16 | 100%m | 0,3 |
| 27 | 1 | 16 | 100%m | 0,6 |
| 28 | 1 | 20 | 50%m | 0,0 |
| 29 | 1 | 20 | 50%m | 0,3 |
| 30 | 1 | 20 | 50%m | 0,6 |
| 31 | 1 | 20 | 80%m | 0,0 |
| 32 | 1 | 20 | 80%m | 0,3 |
| 33 | 1 | 20 | 80%m | 0,6 |
| 34 | 1 | 20 | 100%m | 0,0 |
| 35 | 1 | 20 | 100%m | 0,3 |
| 36 | 1 | 20 | 100%m | 0,6 |

mento RVND: $N_1$, $N_2$, $N_3$, $N_4$ e $N_5$ (Seção 5.2.1). Nesta calibração, foram consideradas todas as possíveis combinações que poderiam ser feitas entre as duas vizinhanças da busca local ($N_1$ e $N_2$) e as outras 3 vizinhanças($N_3$, $N_4$ e $N_5$) implementadas para o procedimento RVND.

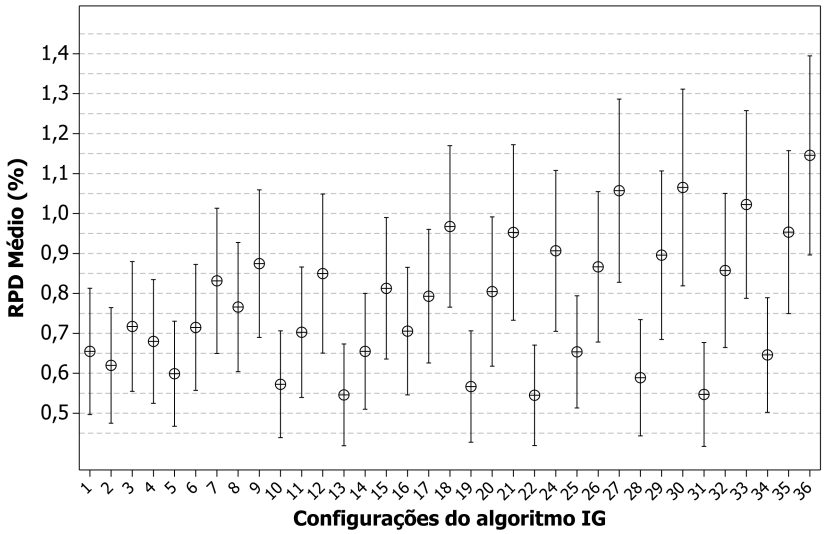


**Figura 6.4.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração dos parâmetros do IG .

A Figura 6.5 apresenta o gráfico de médias resultante do teste HSD de Tukey, com nível de confiança de 95% para as combinações entre as vizinhanças. É possível ver, nesta figura, que a menor média de RPD é obtida com a combinação das três vizinhanças: $N_1$, $N_2$ e $N_4$. Apesar de não existir diferença estatisticamente significativa entre as configurações, uma vez que os intervalos se sobrepõem, pode-se concluir que a combinação entre as outras estruturas de vizinhança não trouxe melhoras para as soluções. Desta forma, o procedimento RVND do algoritmo IG-RVND utiliza as três vizinhanças $N_1$, $N_2$ e $N_4$.

## 6.4 Implementação do algoritmo SA* proposto por Ruiz-Torres et al. (2013)

De acordo com a revisão da literatura, há somente uma heurística desenvolvida para o problema abordado, proposta por Ruiz-Torres et al. (2013). Estes autores implementaram um algoritmo baseado na meta-heurística *Simulated Annealing*, chamado de SA*.

O algoritmo SA* é baseado na aceitação de soluções de qualidade inferior com uma certa probabilidade, com o objetivo de escapar do mínimo local. Quando o número de iterações incrementa, a probabilidade de aceitar estas soluções ruins diminui. O processo finaliza quando o número de iterações sem melhora ($2n$) é atingido, em que $n$ representa o número total de tarefas. A probabilidade de aceitar

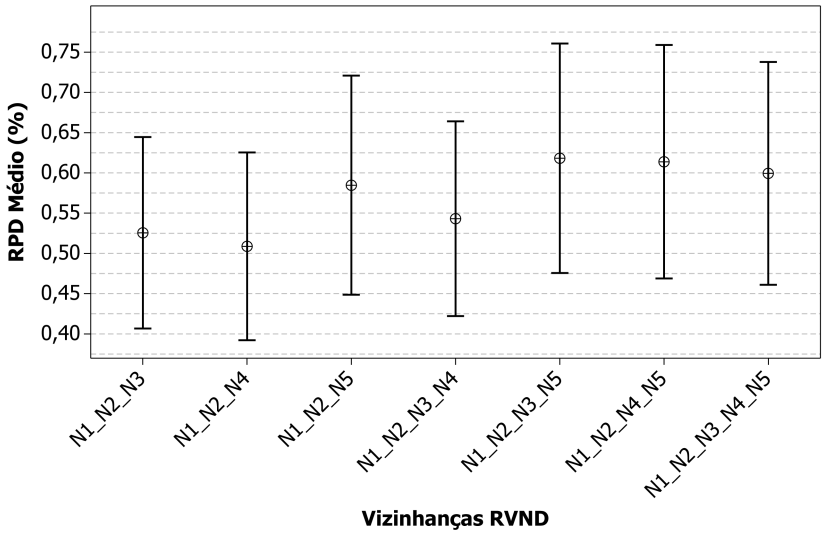


**Figura 6.5.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para a calibração das vizinhanças do procedimento RVND do algoritmo IG-RVND.

as soluções ruins é controlado pelo processo de temperatura. Em cada iteração do algoritmo, as estruturas de vizinhanças $N_1$ (*Troca*) e $N_2$ (*Inserção*) são aplicadas para obter um ótimo local (Ruiz-Torres et al., 2013).

O algoritmo SA* é composto pela melhor implementação de duas versões do algoritmo *Simulated Annealing*, denominadas $SA_1$ e $SA_2$. Estas duas versões se diferenciam na forma de avaliar se a qualidade da solução melhorou. No $SA_1$, verifica-se o *makespan*. No $SA_2$, considera-se a soma dos tempos de conclusão das tarefas em todas as máquinas (Ruiz-Torres et al., 2013).

Cada versão do algoritmo é executada 8 vezes. Como definido na Seção 5.1.1, as regras de prioridade de 1 a 8 ($p_j^{min}, p_j^{max}, d_j^{min}, d_j^{max}, r_j^{min}, r_j^{max}, v_j^{min}, v_j^{max}$), foram definidas por Ruiz-Torres et al. (2013). Estas 8 regras são utilizadas para construir a solução inicial de cada uma das 8 execuções dos algoritmos $SA_1$ e $SA_2$. Desta forma, o SA* obtém a melhor solução de 16 execuções (8 execuções do $SA_1$ e 8 execuções do $SA_2$).

Como os resultados do SA* são obtidos através de 16 execuções, os algoritmos propostos neste trabalho também são executados 16 vezes. No entanto, os algoritmos propostos utilizam apenas a regra de prioridade 9 ($r_j^{medio}$) para construir a solução inicial, como é mostrado na Seção 6.3.1 de calibração dos parâmetros.

O algoritmo SA* foi testado em 900 instâncias de médio porte. Estas instâncias e os melhores resultados obtidos por Ruiz-Torres et al. (2013) estão disponíveis em:

`http://ruiz-torres.uprrp.edu/dm/`.

Neste trabalho, foi feita a reimplementação do algoritmo SA* seguindo os passos originais descritos no artigo do Ruiz-Torres et al. (2013). Este algoritmo reimplementado foi executado no computador usado para este trabalho usando dois diferentes critérios de parada. O primeiro critério de parada é o mesmo usado no artigo do Ruiz-Torres et al. (2013), o qual é estabelecido em $2n$ iterações sem melhora da melhor solução corrente. O segundo critério de parada é baseado no tempo máximo de CPU definido por $\frac{n}{m}$ segundos, ou seja, a mesma condição utilizado nos algoritmos proposto neste trabalho. O algoritmo SA* com o primeiro critério de parada é chamado de *SA*1* e com o segundo critério de parada é denominado *SA*2*.

Nesta seção, é feita a comparação do resultado dos algoritmos *SA*1* e *SA*2* com o melhor resultado do algoritmo SA* apresentado por Ruiz-Torres et al. (2013). Na Figura 6.6, que mostra o gráfico de médias resultante do teste HSD de Tukey com nível de confiança de 95%, é possível observar que o resultado obtido pelas implementações dos algoritmos *SA*1* e *SA*2* são estatisticamente melhores que o resultados apresentados pelo algoritmo SA*. Embora o *SA*2* esteja um pouco melhor que o *SA*1*, não há diferença significativa entre estas duas implementações. Isto mostra que os dois critérios de paradas aplicados são compatíveis.

A Tabela 6.5 apresenta o RPD dos melhores resultados obtidos pelos algoritmos SA*, SA*1 e SA*2 em 16 execuções. É possível ver que os algoritmos reimplementados neste trabalho (SA*1 e SA*2) apresentam os melhores resultados (valores em negrito) para todas as instâncias. É possível notar, também, a similaridade dos resultados obtidos pelos algoritmos SA*1 e SA*2.

Assim, adotou-se o algoritmo *SA*2* para comparações com as meta-heurísticas adaptadas neste trabalho, uma vez que utiliza o mesmo critério de parada ($\frac{n}{m}$ segundos) destas.

Vale ressaltar que, apesar dos algoritmos implementados, SA*1 e SA*2, seguirem os passos originais do SA* proposto por Ruiz-Torres et al. (2013), existe a diferença da linguagem de programação e hardware utilizados para realização dos testes. Isto explica o desempenho inferior do SA* em comparação aos algoritmos implementados.

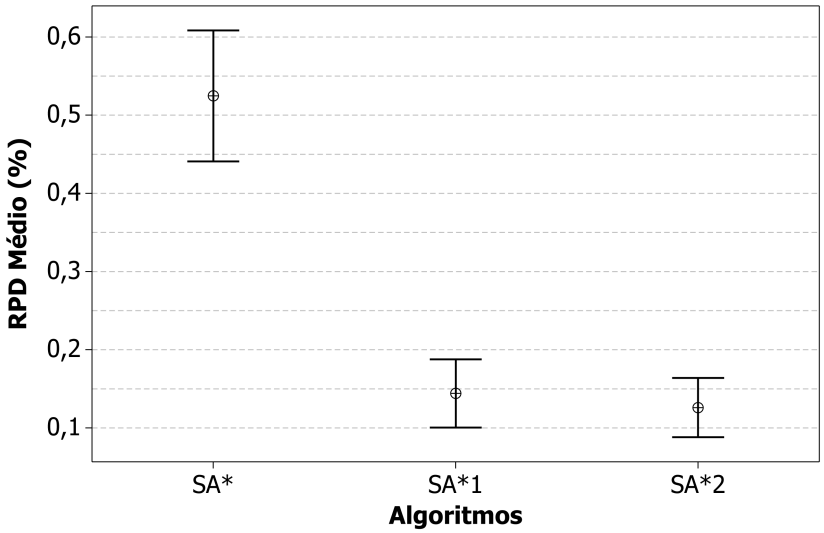


**Figura 6.6.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para os algoritmos SA*, SA*1 e SA*2.

**Tabela 6.5.** RPDs dos melhores resultados obtidos pelos algoritmos em 16 execuções (instâncias da literatura)

| Instância $n$ x $m$ | SA* | SA*1 | SA*2 |
|---|---|---|---|
| 20 x 4 | 0,1 | **0,0** | **0,0** |
| 20 x 7 | 0,6 | **0,1** | **0,1** |
| 20 x 10 | 0,4 | **0,2** | **0,2** |
| 35 x 4 | 0,1 | **0,0** | **0,0** |
| 35 x 7 | 0,5 | **0,1** | **0,1** |
| 35 x 10 | 1,2 | 0,5 | **0,3** |
| 50 x 4 | 0,1 | **0,0** | **0,0** |
| 50 x 7 | 0,7 | **0,0** | 0,1 |
| 50 x 10 | 1,1 | 0,4 | **0,3** |
| Média | 0,5 | **0,1** | **0,1** |

# Capítulo 7

# Experimentos Computacionais - Parte 2

Na segunda parte dos experimentos, são apresentados os testes feitos com o Modelo de Programação Inteira Mista (MPIM). Além disso, são apresentados os resultados obtidos pelos algoritmos propostos: ILS, ILS-RVND, IG e IG-RVND. Estes algoritmos são comparados entre si e também são comparados com o algoritmo estado-da-arte da literatura. Ao nosso conhecimento, o único algoritmo para o problema abordado foi proposto por Ruiz-Torres et al. (2013).

Este capítulo está dividido em quatro seções. A Seção 7.1 explica detalhes da execução do MPIM no CPLEX e mostra os resultados obtidos. As Seções 7.2 e 7.3 apresentam os resultados obtidos pelos algoritmos heurísticos para as instâncias de médio e grande porte, respectivamente. E por fim, a Seção 7.4 faz a análise de convergência dos resultados dos algoritmos heurísticos propostos neste trabalho.

## 7.1 Experimentos Computacionais com o Modelo de Programação Inteira Mista (MPIM)

Nesta seção, os experimentos computacionais realizados utilizando as instâncias de médio porte são apresentados. O modelo matemático proposto na Seção 4.2 foi implementado utilizando a biblioteca ILOG Concert e a linguagem de programação C++. Sua resolução se deu através do software IBM CPLEX versão 12.5, configurado com os parâmetros *default*.

### 7.1.1 Execução do MPIM no CPLEX

A execução do MPIM pelo IBM CPLEX requer a geração de todos as possíveis combinações de tarefas, ou seja, as quais chamamos de padrões. Um padrão $S$ é representado computacionalmente por um vetor de tamanho $n$, onde cada posição pode conter os valores 0 ou 1, em que 1 indica a tarefa contida nesse padrão, e 0, caso contrário. Como exemplo, dado um padrão com quatro tarefas representado por [0,0,1,1] indica que as tarefas 1 e 2 não estão presentes e as tarefas 3 e 4 estão presentes nesse padrão. Já o padrão [1,0,1,1] indica que as tarefas 1, 3 e 4 estão presentes neste padrão e a tarefa 2 não está.

Assim, para representar todos os possíveis padrões que podem ser atribuídos às máquinas, é utilizado uma matriz de inteiros, onde as linhas representam os padrões e as colunas representam as tarefas de cada padrão. Desta forma, para gerar todos os padrões para um problema com $n$ tarefas, gera-se $2^{(n)} - 1$ vetores de números binários. A Figura 7.1 ilustra o exemplo de um problema com 4 tarefas que teria os seguintes padrões:

| Decimal | Padrão |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |

| Decimal | Padrão |
|---|---|
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| 10 | 1 0 1 0 |
| 11 | 1 0 1 1 |
| 12 | 1 1 0 0 |
| 13 | 1 1 0 1 |
| 14 | 1 1 1 0 |
| 15 | 1 1 1 1 |

**Figura 7.1.** Exemplo de vetor binário para 4 tarefas

A Tabela 7.1 mostra a quantidade de padrões para determinada quantidade de tarefas. Pode-se perceber que a partir de 30 tarefas, a quantidade de padrões necessários para representá-las é muito grande, passando de um bilhão.

**Tabela 7.1.** Quantidade de padrões possíveis para o número de tarefas

| Número de Tarefas | Quantidade de padrões |
|---|---|
| 4 | 16 |
| 8 | 256 |
| 16 | 65536 |
| 25 | 32768 |
| 30 | 1073741824 |
| 50 | $1{,}125899907X10^{15}$ |

Antes de executar o modelo pelo software IBM CPLEX, para cada padrão gerado, aplicam-se dois tipos de filtros. Isto é necessário para diminuir a quantidade total de padrões, pois, as instâncias com muitas tarefas (por exemplo, 35 a 50 tarefas), tornam-se impossíveis de serem executadas computacionalmente, devido ao grande gasto de memória para alocar todos os padrões.

As duas filtragens de padrões que são realizadas utilizam um valor limite para o máximo tempo de conclusão do processamento das tarefas (*makespan*), denominado $U$, que é obtido através dos resultados do método IG (Algoritmo 8). A seguir são explicadas as duas filtragens aplicadas para diminuir o número total de padrões:

- A primeira filtragem consiste em determinar o número máximo de tarefas que pode ser atribuída a uma máquina sem que o tempo de conclusão das tarefas ultrapasse o valor de $U$. Considere uma máquina $k$ e assume-se que as tarefas estão ordenadas de acordo com a ordem crescente do valores de tempo de processamento $p_{jk}$. Define-se $r(k) = min\{i : \sum_{j=1}^{i} p_{jk} \geq U\}$ como o índice da primeira tarefa que não pode ser executada na máquina $k$ sem atingir o valor de $U$. Então, qualquer padrão que for sequenciado na máquina $k$ para melhorar a solução, ou seja, diminuir o *makespan*, pode ter no máximo $r(k)$ tarefas.

  Note que esta avaliação não considera o valor de desgaste $d_{jk}$, tornando-se uma visão otimista do problema, no qual as máquinas não se desgastam. Como resultado, pode-se gerar padrões $S \in T$ cujos reais *makespan* $C(S, k)$ estão estritamente maiores do que o esperado. Então, faz-se necessário considerar os desgastes. Para isso, considere novamente uma máquina $k$ e define-se que $dmin_k = min\{d_{jk}, j \in N\}$ é o menor desgaste para a máquina $k$. Então, aplica-se este menor desgaste sobre as tarefas, através da multiplicação do tempo de processamento $p_{jk}$ por $\frac{1}{1-dmin_k}$, o que aumentará o tempo de processamento de cada tarefa da máquina $k$.

  Obtendo-se estes valores, somam-se quantas tarefas é possível alocar em cada máquina sem que o tempo de processamento final das tarefas ultrapasse o valor de $U$. Entre todas as máquinas, aquela que possui o maior número de tarefas, sem que o tempo de processamento final das tarefas ultrapasse o valor de $U$, determinará o número máximo de tarefas $r(k)$ possíveis que cada padrão $S$ pode conter. Os padrões que contém mais que $r(k)$ tarefas são descartados.

- Na segunda filtragem, para cada padrão, calculam-se os tempos de processamentos reais das tarefas (Equação 3.2), levando em conta os efeitos de des-

gastes das tarefas previamente processadas na máquina e o desempenho desta máquina (Equação 3.1), assim como são explicados na Seção 3.2. Logo, verifica se o padrão tem o tempo de conclusão das tarefas superior ao limite $U$ em todas as máquinas, isto é, $C(S,k) \geq U; k \in [1,...,m]$. Caso isto ocorra, este padrão é excluído.

Os padrões que não foram excluídos durantes estas duas filtragens são salvos na memória para serem, então, analisados pelo software IBM CPLEX.

Como visto anteriormente na Tabela 7.1, para instâncias de 50 tarefas, são necessários $1{,}125899907X10^{15}$ padrões. Nem mesmo utilizando as filtragens foi possível executar este tipo de instância no CPLEX, uma vez que o gasto de memória, até para gerar o possíveis padrões e avaliá-los, é muito excessivo.

Para as instâncias de 35 tarefas e 4 máquinas, foi possível gerar e avaliar todos os padrões. No entanto, o número de padrões viáveis após as filtragens ainda permaneceu muito alto, fazendo com que o CPLEX não fosse capaz de executar o modelo.

Dessa forma, o CPLEX foi capaz de encontrar a solução ótima para 500 instâncias, do total de 900, compreendendo os tipos: 20 tarefas e {4, 7 e 10} máquinas; e 35 tarefas e {7 e 10} máquinas, sendo 100 instâncias para cada tipo. Por outro lado, o CPLEX não conseguiu executar 400 instâncias dos seguintes tipos: 35 tarefas e 4 máquinas; e 50 tarefas e {4, 7 e 10} máquinas, sendo 100 instâncias para cada tipo. Para as instâncias do último caso, foi proposta uma nova metodologia, explicada a seguir.

### 7.1.2 Definição de padrões para executar o MPIM no CPLEX

Nesta metodologia, ao invés de gerar todos os possíveis padrões para cada instância, são utilizados no CPLEX os padrões explorados pela heurística IG (Algoritmo 8). Esta abordagem não garante a solução ótima, contudo cria a possibilidade de encontrar uma melhor combinação entre esses padrões, ou seja, uma combinação que não foi avaliada pelo IG.

Inicialmente, são armazenados em um arquivo todos os padrões que compõem uma boa solução explorada pelo IG, desde o início até o fim da execução do algoritmo (cada padrão é representado por um vetor binário no arquivo). Em seguida os padrões duplicados são eliminados do arquivo e aplicam-se as duas filtragens explicadas na seção anterior. Os padrões remanescentes são utilizados na execução do modelo pelo CPLEX. Os resultados obtidos com o MPIM utilizando essa metodologia são apresentados na próxima seção.

A Figura 7.2 exemplifica os seis principais passos desta metodologia para uma instância real do problema, contendo oito tarefas e três maquinas. Primeiramente, todos os padrões que compõem uma boa solução explorada pelo IG são armazenadas em vetores binários (7.2, passo 1). Em seguida, eliminam-se os padrões que estão duplicados (7.2, passo 2). Para reduzir a quantidade de padrões a serem avaliados, aplicam-se duas filtragens. Na primeira filtragem, suponha que o valor limite $U = 98{,}0$ e o número máximo de tarefas $r(k)$ que se pode ter em um máquina sem ultrapassar o valor de U é igual a 5, isto é, r(k) = 5. Logo, eliminam-se os padrões com mais de 5 tarefas (7.2, passo 3). Na segunda filtragem elimina-se cada padrão que tem o tempo de conclusão das tarefas superior ao limite $U$ em todas as máquinas, isto é, $C(S,k) \geq U; k \in [1,...,m]$ (7.2, passo 4). Os padrões que não foram excluídos até aqui são analisados pelo software IBM CPLEX (7.2, passo 5). Finalmente, obtém-se o melhor subconjunto de tarefas para cada máquina (7.2, passo 6).

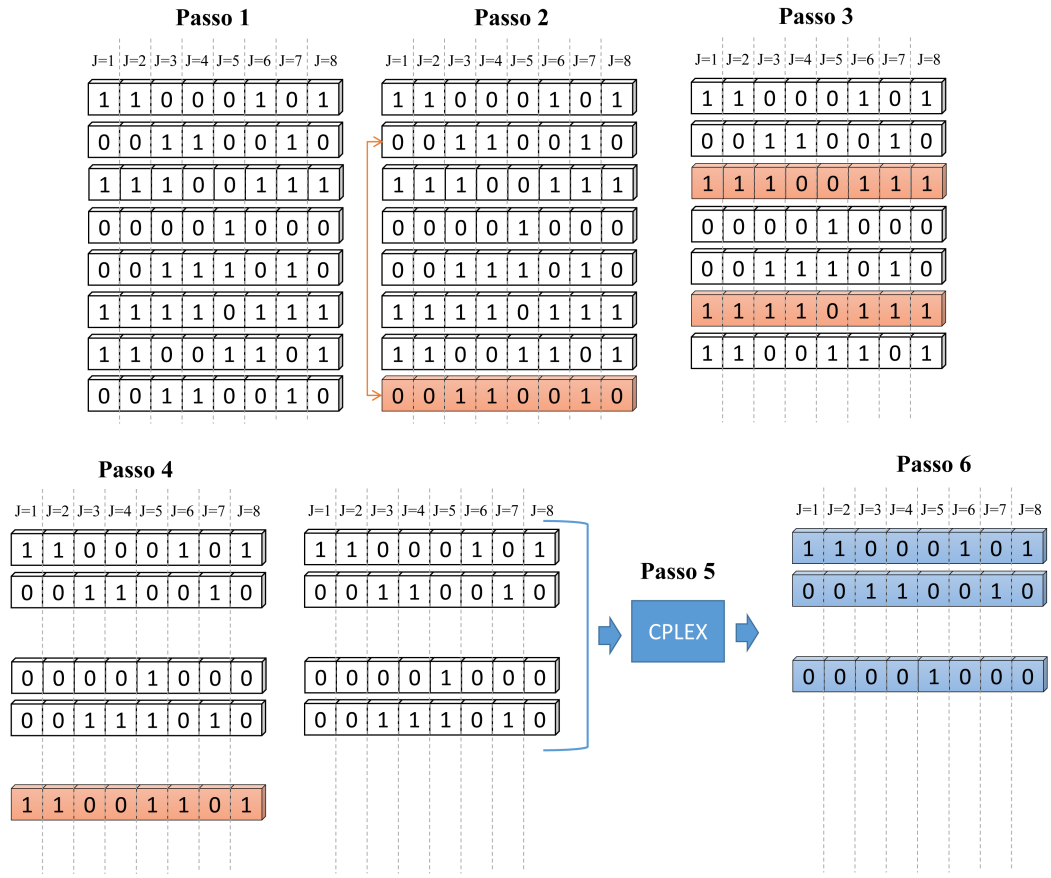


**Figura 7.2.** Exemplo dos padrões para um instância de 8 tarefas e 3 máquinas

Para este exemplo, a melhor solução é formada pelo primeiro, segundo e terceiro padrões remanescentes nas máquinas 1, 2 e 3 , respectivamente. Ou seja, as

tarefas 1, 2, 6 e 8 foram atribuídas à máquina 1; as tarefas 3, 4 e 7 à maquina 2; e a tarefa 5 atribuída à máquina 3. Levando em conta a instância original do exemplo, esta solução obtida é a ótima, com os tempos de conclusão do processamento das tarefas iguais a: 96,7, 90,1 e 92,3 nas máquinas 1, 2 e 3, respectivamente. Vale lembrar que essa metodologia não garante a solução ótima, porém é um forma de melhorar a solução encontrada pelo algoritmo IG utilizando apenas os padrões explorados em todas as iterações desse algoritmo.

### 7.1.3 Resultados do MPIM

Nesta seção, os resultados obtidos pelos algoritmos propostos (ILS, ILS-RVND, IG e IG-RVND) e pela versão implementada da literatura (SA*2) são comparados com os resultados encontrados pelo CPLEX. Estes experimentos foram realizados com o grupo das instâncias de médio porte que contém 900 no total. Para melhor entendimento, este grupo de 900 instâncias são divididos em dois subgrupos. O primeiro subgrupo é referente às 500 instâncias que o CPLEX obteve as soluções ótimas, e o segundo grupo referente às outras 400 que através da metodologia utilizada, não obtiveram otimalidade comprovada.

A partir das 500 soluções ótimas obtidas pelo CPLEX (1º subgrupo), buscou-se verificar quantas vezes os algoritmos propostos e o SA*2 atingiram estes valores, avaliando a média dos valores de 16 execuções para cada instância. A Tabela 7.2 mostra esta comparação. Pode-se observar que o algoritmo que obteve valores médios mais próximos das soluções ótimas foi o IG-RVND. Em segundo lugar ficou o IG, seguido do ILS-RVND e ILS, respectivamente. Em último lugar está o SA*2, com apenas 3 valores médios iguais às soluções ótimas.

**Tabela 7.2.** Número de soluções ótimas atingidas (Média dos resultados de 16 execuções)

| Algoritmo | Quantidade de menores *makespan* | Porcentagem (%) |
|---|---|---|
| SA*2 | 3 | 0,6 |
| ILS | 406 | 81,2 |
| ILS-RVND | 413 | 82,6 |
| IG | 437 | 87,4 |
| IG-RVND | **440** | **88,0** |

Para verificar a eficácia da metodologia que utiliza padrões pré-definidos (Seção 7.1.2), o primeiro subgrupo das 500 instâncias, que possuem a solução ótima

conhecida também foi testado.

A Tabela 7.3 mostra o número de soluções ótimas atingidas pelos algoritmos desenvolvidos, avaliando os melhores valores encontrados em 16 execuções. Apresenta, ainda, na última linha, o número de soluções ótimas encontradas pela metodologia do CPLEX com padrões pré-definidos, vale ressaltar que esta metodologia foi executada apenas uma vez. Os resultados obtidos pelo modelo foram satisfatórios, com 96% de soluções idênticas às ótimas. Através da Tabela 7.3, nota-se também que os algoritmos propostos (ILS-RVND, IG, e IG-RVND) encontram a solução ótima pelo menos uma vez em 16 execuções para todas as instâncias. Além disso, percebe-se o algoritmo SA*2 atinge a solução ótima pelo menos uma vez para 87% da instâncias, valor bem mais alto de quando se compara a média de soluções, que não atingiu nem 1%.

**Tabela 7.3.** Número de soluções ótimas atingidas (Melhores resultados em 16 execuções)

| Algoritmo | Quantidade de menores *makespan* | Porcentagem (%) |
|---|---|---|
| SA*2 | 435 | 87,0 |
| ILS | 499 | 99,8 |
| ILS-RVND | **500** | **100,0** |
| IG | **500** | **100,0** |
| IG-RVND | **500** | **100,0** |
| CPLEX (Padrões) | 482 | 96,4 |

A seguir são mostrados os resultados obtidos com o segundo subgrupo formado pelas 400 instâncias.

Na Tabela 7.4, apresenta-se o número de melhores soluções atingidas pelos algoritmos desenvolvidos, avaliando a média dos valores em 16 execuções. A Tabela 7.4 apresenta também para quantas instâncias a metodologia do CPLEX com padrões pré-definidos encontra o melhor valor conhecido, lembrando que esta metodologia é executada apenas uma vez. Nota-se que a metodologia do CPLEX (padrões) atinge as melhores soluções em 79,5% das instâncias. Em segundo lugar encontra-se o IG-RVND com 69,0%, seguido do IG, ILS-RVND, ILS e SA*2, respectivamente.

A Tabela 7.5 realiza a mesma comparação feita na Tabela 7.4, mas desta vez, para os algoritmos desenvolvidos, considera-se o melhor dos valores obtidos em 16 execuções. Observa-se que o IG-RVND atinge as melhores soluções para um número maior de instâncias (99,7%). Em seguida, estão os algoritmos IG, ILS-RVND, ILS e

**Tabela 7.4.** Número de melhores soluções encontradas (Média dos resultados em 16 execuções)

| Algoritmo | Quantidade de menores *makespan* | Porcentagem (%) |
|---|---|---|
| SA*2 | 12 | 3,0 |
| ILS | 182 | 45,5 |
| ILS-RVND | 225 | 56,2 |
| IG | 244 | 61,0 |
| IG-RVND | 276 | 69,0 |
| CPLEX (Padrões) | **318** | **79,5** |

SA*2, respectivamente. Por último, encontra-se a metodologia do CPLEX com uso de padrões pré-definidos, com 74,0%.

**Tabela 7.5.** Número de melhores soluções encontradas (Melhores resultados em 16 execuções)

| Algoritmo | Quantidade de menores *makespan* | Porcentagem (%) |
|---|---|---|
| SA*2 | 309 | 77,2 |
| ILS | 378 | 94,5 |
| ILS-RVND | 393 | 98,2 |
| IG | 392 | 98,0 |
| IG-RVND | **399** | **99,7** |
| CPLEX (Padrões) | 296 | 74,0 |

## 7.2 Comparação dos algoritmos heurísticos nas instâncias de médio porte

Nesta seção, é feita a comparação do desempenho do algoritmo SA*2, com os algoritmos propostos neste trabalho: ILS, ILS-RVND, IG e IG-RVND para as instâncias do grupo de médio porte (instâncias da literatura). Todos os algoritmos foram executados 16 vezes com o mesmo tempo de CPU ($\frac{n}{m}$ segundos) como critério de parada. Os resultados obtidos são analisados utilizando o valor do RPD (Equação 6.1), o qual é calculado através da média, o melhor e o pior *makespan* em 16 execuções de cada instância.

A Tabela 7.6 mostra os valores de RPDs calculados a partir dos resultados médios (de 16 execuções) encontrados pelos algoritmos comparados para todas as

instâncias de médio porte. Os valores médios de RPDs são agrupadas pelo tamanho, de acordo com número de tarefas e de máquinas ($nxm$). Os valores em negrito representam os melhores resultados obtidos. Nesta tabela, estão exibidos também, na última coluna, os tempos de CPU consumidos pelos algoritmos. Claramente, é possível perceber que o algoritmo IG-RVND encontrou as melhores médias em todas os grupos de instâncias. Em segundo lugar, está o algoritmo IG, que obteve as melhores médias em seis dos nove grupos. O SA*2, ILS, ILS-RVND, IG e IG-RVND possuem a média global de RPDs iguais a 2,3%, 0,2%, 0,2%, 0,1% e 0,1%, respectivamente.

**Tabela 7.6.** Valores médios de RPDs (em 16 execuções) e tempos de CPU para os algoritmos comparados

| Instância $n$ x $m$ | SA*2 | ILS | ILS-RVND | IG | IG-RVND | Tempo de CPU (em segundos) |
|---|---|---|---|---|---|---|
| 20 x 4 | 1,3 | **0,0** | **0,0** | **0,0** | **0,0** | 5,0 |
| 20 x 7 | 3,6 | **0,0** | **0,0** | **0,0** | **0,0** | 2,8 |
| 20 x 10 | 3,8 | **0,0** | **0,0** | **0,0** | **0,0** | 2,0 |
| 35 x 4 | 0,6 | **0,0** | **0,0** | **0,0** | **0,0** | 8,7 |
| 35 x 7 | 2,1 | 0,1 | 0,1 | 0,1 | **0,0** | 5,0 |
| 35 x 10 | 4,3 | 0,3 | 0,2 | 0,2 | **0,1** | 3,5 |
| 50 x 4 | 0,4 | **0,0** | **0,0** | **0,0** | **0,0** | 12,5 |
| 50 x 7 | 1,7 | 0,3 | 0,3 | **0,2** | **0,2** | 7,1 |
| 50 x 10 | 3,2 | 0,9 | 0,9 | 0,5 | **0,4** | 5,0 |
| Média | 2,3 | 0,2 | 0,2 | **0,1** | **0,1** | 5,7 |

A Tabela 7.7 apresenta os RPDs dos algoritmos, considerando as melhores e piores soluções obtidas pelos algoritmos em 16 execuções, respectivamente. Para a comparação dos melhores resultados é possível ver que os algoritmos propostos apresentam o melhor desempenho para todas as instâncias. Dado que quanto menor valor do RPD melhor será a qualidade da solução, pode-se concluir que os algoritmos propostos conseguem atingir a melhor solução conhecida ao menos uma vez em 16 execuções. Na comparação dos piores resultados encontrados percebe-se que o algoritmos SA*2 apresenta os resultados menos satisfatórios. O SA*2, ILS, ILS-RVND, IG e IG-RVND possuem a média global RPDs de 8,5%, 0,5%, 0,5%, 0,4% e 0,3%, respectivamente, para os piores resultados obtidos. Através das tabelas, é fácil notar que os algoritmos propostos têm melhores desempenhos que o algoritmo SA*2.

Para validar os resultados obtidos pelos algoritmos e verificar se as diferenças observadas são estatisticamente significantes, foi realizada a ANOVA paramétrica.

**Tabela 7.7.** RPDs dos melhores e piores resultados obtidos pelos algoritmos

| | Comparação dos melhores resultados | | | | | Comparação dos piores resultados | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instância $n$ x $m$ | SA*2 | ILS | ILS RVND | IG | IG RVND | SA*2 | ILS | ILS RVND | IG | IG RVND |
| 20 x 4 | **0,0** | **0,0** | **0,0** | **0,0** | **0,0** | 7,7 | **0,0** | **0,0** | **0,0** | **0,0** |
| 20 x 7 | 0,1 | **0,0** | **0,0** | **0,0** | **0,0** | 12,2 | 0,1 | **0,0** | **0,0** | **0,0** |
| 20 x 10 | 0,2 | **0,0** | **0,0** | **0,0** | **0,0** | 12,9 | 0,1 | **0,0** | **0,0** | **0,0** |
| 35 x 4 | **0,0** | **0,0** | **0,0** | **0,0** | **0,0** | 3,5 | **0,0** | **0,0** | **0,0** | **0,0** |
| 35 x 7 | 0,1 | **0,0** | **0,0** | **0,0** | **0,0** | 9,4 | 0,3 | 0,2 | 0,2 | **0,1** |
| 35 x 10 | 0,5 | **0,0** | **0,0** | **0,0** | **0,0** | 12,3 | 1,3 | 1,1 | 0,9 | **0,7** |
| 50 x 4 | **0,0** | **0,0** | **0,0** | **0,0** | **0,0** | 2,0 | 0,2 | **0,1** | **0,1** | **0,1** |
| 50 x 7 | 0,3 | **0,0** | **0,0** | **0,0** | **0,0** | 6,0 | 0,9 | 0,9 | 0,7 | **0,6** |
| 50 x 10 | 0,7 | **0,0** | **0,0** | **0,0** | **0,0** | 10,3 | 2,1 | 2,2 | 1,4 | **1,1** |
| Média | 0,2 | **0,0** | **0,0** | **0,0** | **0,0** | 8,5 | 0,5 | 0,5 | 0,4 | **0,3** |

Dado que o *valor-P* na ANOVA resultou em 0,00 e este valor é menor que 0,05, pode-se concluir que as diferenças são estatisticamente significantes.

A Figura 7.3 apresenta, para as instâncias de médio porte, o gráfico de médias resultante do teste HSD de Tukey com nível de confiança de 95%, o qual compara o algoritmo SA*2 e os algoritmos propostos: ILS, ILS-RVND, IG e IG-RVND. Observa-se que na Figura 7.3, a média do SA*2 é significativamente diferente das médias dos outros algoritmos, uma vez que o intervalo para o algoritmo SA*2 não intercepta os intervalos dos outros algoritmos. Isso nos levar a concluir que, as heurísticas propostas mostram desempenho superior ao algoritmo SA*2.

Para melhor analisar o desempenho dos algoritmos propostos entre si, é feita uma comparação apenas entre estes algoritmos. A Figura 7.4 apresenta o gráfico de médias resultantes do teste HSD de Tukey com nível de confiança de 95%. Nota-se, a partir desta figura, que o IG e IG-RVND tem desempenho significativamente melhor que os dois outros algoritmos (ILS e ILS-RVND), uma vez que não há sobreposição dos intervalos entre estes algoritmos. Além disso, percebe-se que não há diferenças significativas entre os algoritmos ILS e ILS-RVND, e entre IG e IG-RVND, já que estes gráficos se interceptam.

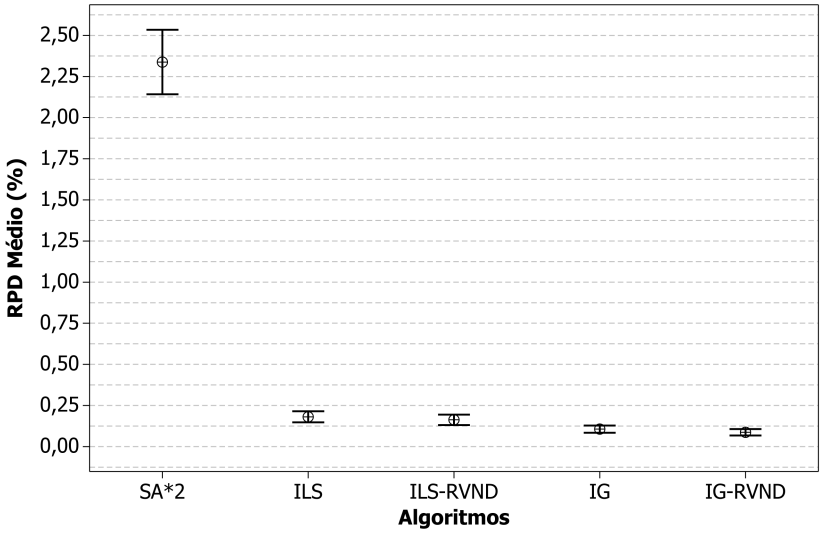


**Figura 7.3.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do algoritmo SA*2 e algoritmos propostos.

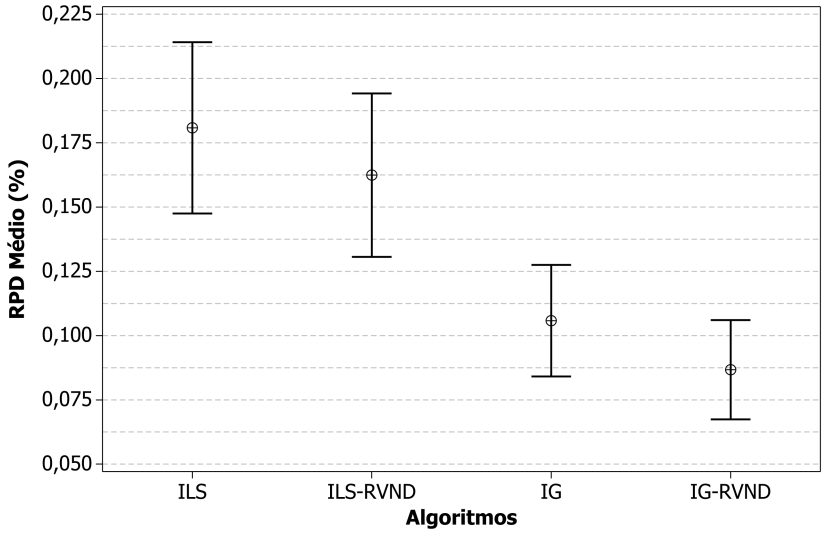


**Figura 7.4.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos algoritmos propostos.

## 7.3 Comparação dos algoritmos heurísticos nas instâncias de grande porte

Da mesma forma que na seção anterior, o algoritmo SA*2 e os algoritmos propostos neste trabalho foram executados 16 vezes. Os valores de RPDs foram calculados através da média, o melhor e o pior *makespan* das 16 execuções de cada instância.

A Tabela 7.8 mostra a comparação das soluções médias (de 16 execuções) encontrada pelos algoritmos para todas as instâncias de grande porte (geradas neste trabalho) e os tempos de CPU consumidos pelos algoritmos. As médias de RPDs são agrupadas pelo tamanho da instância, $n$ x $m$. Os valores em negrito representam os melhores resultados. Como pode-se ver, os algoritmos propostos apresentam as melhores médias para todas as instâncias. Observa-se ainda que somente o algoritmo IG-RVND produz os melhores resultados médios para todas os grupos de instâncias. No entanto, para os grupos 80x5, 100x5 e 150x5, os algoritmos IG e/ou ILS-RVND conseguem atingir os mesmos valores. O algoritmo IG-RVND apresenta a menor média RPD com o valor 0,8%. Os algoritmos SA*2, ILS, ILS-RVND e IG possuem a média global RPDs de 4,4%, 1,6%, 1,6% e 0,9%, respectivamente.

**Tabela 7.8.** Valores médios de RPDs (em 16 execuções) e tempos de CPU para os algoritmos comparados

| Instância $n$ x $m$ | SA*2 | ILS | ILS-RVND | IG | IG-RVND | Tempo de CPU (em segundos) |
|---|---|---|---|---|---|---|
| 80 x 5 | 1,0 | 0,3 | **0,2** | **0,2** | **0,2** | 16,0 |
| 80 x 10 | 3,7 | 1,3 | 1,5 | 0,8 | **0,7** | 8,0 |
| 80 x 20 | 8,1 | 2,6 | 2,7 | 1,3 | **1,2** | 4,0 |
| 100 x 5 | 1,1 | 0,4 | 0,3 | **0,2** | **0,2** | 20,0 |
| 100 x 10 | 3,8 | 1,5 | 1,6 | 0,9 | **0,7** | 10,0 |
| 100 x 20 | 8,7 | 3,1 | 3,2 | 1,7 | **1,5** | 5,0 |
| 150 x 5 | 1,3 | 0,6 | 0,5 | **0,4** | **0,4** | 30,0 |
| 150 x 10 | 4,1 | 1,8 | 1,5 | 1,1 | **0,9** | 15,0 |
| 150 x 20 | 7,7 | 2,9 | 3,0 | 1,7 | **1,4** | 7,5 |
| Média | 4,4 | 1,6 | 1,6 | 0,9 | **0,8** | 12,8 |

A Tabela 7.9 apresenta os RPDs do algoritmos que são obtidos considerando as melhores e piores soluções pelos algoritmos em 16 execuções, respectivamente. Para a comparação dos melhores resultados, é possível ver que o algoritmo IG-RVND apresenta os menores valores de *makespan* para todas as instâncias. Na comparação dos piores resultados, nota-se que o algoritmo de pior desempenho é o SA*2, com o valor de média global RPD igual a 8,1%. Os algoritmos propostos ILS, ILS-RVND, IG e IG-RVND possuem a média global RPDs de 3,1%, 3,0%, 2,0%, e 1,8%, respectivamente. Mais uma vez é fácil notar que os algoritmos propostos têm melhores desempenhos que o algoritmo SA*2, e que o IG-RVND obtém os melhores resultados em todos os grupos de instância.

De forma similar a análise feita para as instâncias de médio porte, nesta seção, é feita a análise do resultados da ANOVA para as instâncias de grande porte. A

**Tabela 7.9.** RPDs dos melhores e piores resultados obtidos pelos algoritmos

| | Comparação dos melhores resultados | | | | | Comparação dos piores resultados | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instância $n$ x $m$ | SA*2 | ILS | ILS RVND | IG | IG RVND | SA*2 | ILS | ILS RVND | IG | IG RVND |
| 80 x 5 | 0,2 | **0,0** | **0,0** | **0,0** | **0,0** | 2,6 | 0,8 | 0,6 | 0,6 | **0,5** |
| 80 x 10 | 1,7 | 0,3 | 0,5 | **0,1** | **0,1** | 7,3 | 2,8 | 2,9 | 1,8 | **1,6** |
| 80 x 20 | 4,4 | 0,7 | 0,8 | **0,1** | **0,1** | 15,4 | 5,0 | 5,4 | 3,1 | **2,9** |
| 100 x 5 | 0,3 | **0,0** | **0,0** | **0,0** | **0,0** | 2,8 | 1,0 | 0,8 | 0,7 | **0,6** |
| 100 x 10 | 1,7 | 0,4 | 0,5 | 0,2 | **0,1** | 6,9 | 3,1 | 2,9 | 1,9 | **1,6** |
| 100 x 20 | 5,1 | 1,2 | 1,2 | 0,3 | **0,2** | 14,0 | 5,5 | 5,5 | 3,4 | **3,2** |
| 150 x 5 | 0,4 | 0,1 | 0,1 | 0,1 | **0,0** | 3,7 | 1,2 | 1,0 | **0,9** | **0,9** |
| 150 x 10 | 1,3 | 0,5 | 0,5 | 0,2 | **0,1** | 7,7 | 3,4 | 3,1 | 2,5 | **2,1** |
| 150 x 20 | 3,6 | 1,2 | 1,2 | 0,3 | **0,1** | 12,4 | 5,0 | 5,1 | 3,4 | **2,9** |
| Média | 2,1 | 0,5 | 0,5 | 0,2 | **0,1** | 8,1 | 3,1 | 3,0 | 2,0 | **1,8** |

Figura 7.5 apresenta o gráfico de médias resultante do teste HSD de Tukey com nível de confiança de 95%, o qual compara o algoritmo SA*2 e os algoritmos propostos. Como se pode ver, os intervalos de confiança dos algoritmos propostos não são interceptados pelo intervalo do algoritmo SA*2. Isto significa que os algoritmos propostos são estatisticamente melhores que o SA*2.

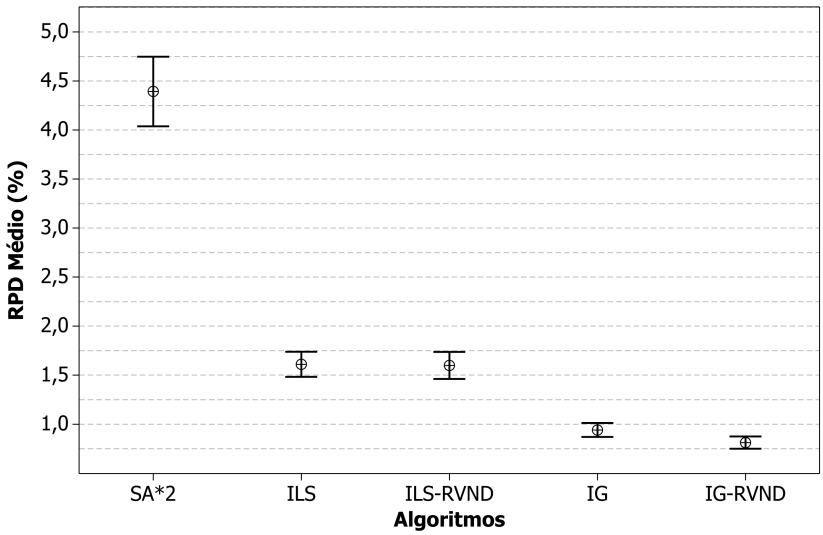


**Figura 7.5.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação do algoritmo SA*2 e algoritmos propostos.

Buscando avaliar de forma mais clara o desempenho apenas dos algoritmos propostos, a Figura 7.6 apresenta o gráfico de médias resultantes do teste HSD de

Tukey com nível de confiança de 95% do resultados da ANOVA destes algoritmos. Nesta figura, é possível perceber que os algoritmos IG e IG-RVND apresentam uma diferença estatisticamente significativa em relação aos algoritmos ILS e ILS-RVND , visto que os gráficos não se interceptam. Além disso, é possível observar que a média obtida pelo IG-RVND é estatisticamente melhor que a média obtida pelo IG, uma vez que os intervalos destes algoritmos não se sobrepõem. Este resultado mostra que a busca local RVND obtém uma melhora considerável na qualidade da solução.

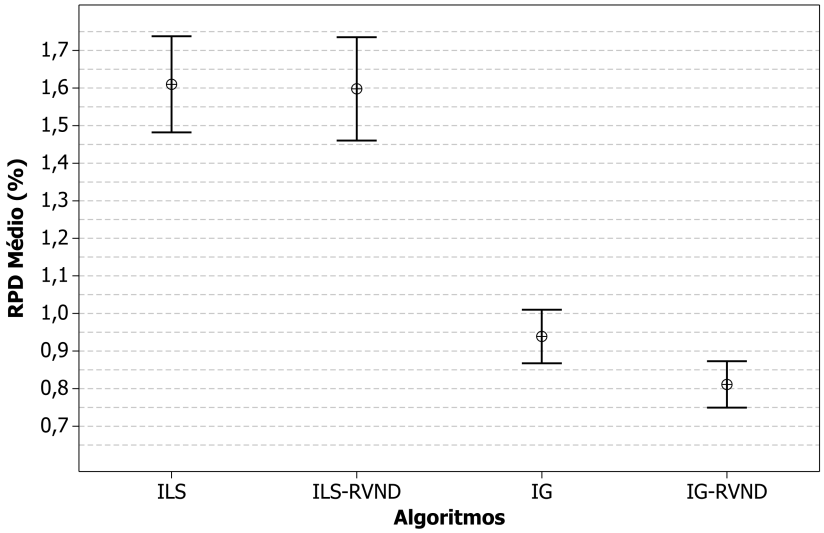


**Figura 7.6.** Gráfico de Médias e intervalos HSD de Tukey com nível de confiança de 95% para comparação dos algoritmos propostos.

Para melhor comparar o desempenho dos algoritmos propostos e do algoritmo SA*2, é feita uma análise entre os melhores soluções obtidas por estes algoritmos. O gráfico da Figura 7.7 apresenta os melhores valores obtidos no teste HSD de Tukey com nível de confiança de 95%. Nota-se que o comportamento dos algoritmos permanece basicamente o mesmo. O algoritmo IG-RVND obtém os melhores valores e o SA*2 tem o pior desempenho. Observa-se também que o algoritmo ILS-RVND apresenta uma pequena piora em relação ao ILS. Contudo, não é uma diferença estatisticamente significante pois os intervalos destes gráficos se sobrepõem.

## 7.4 Análise de convergência dos algoritmos heurísticos

Os experimentos descritos nas Seções 7.2 e 7.3 foram feitos focando na qualidade das soluções em relação ao valor da função objetivo. No entanto, é importante considerar

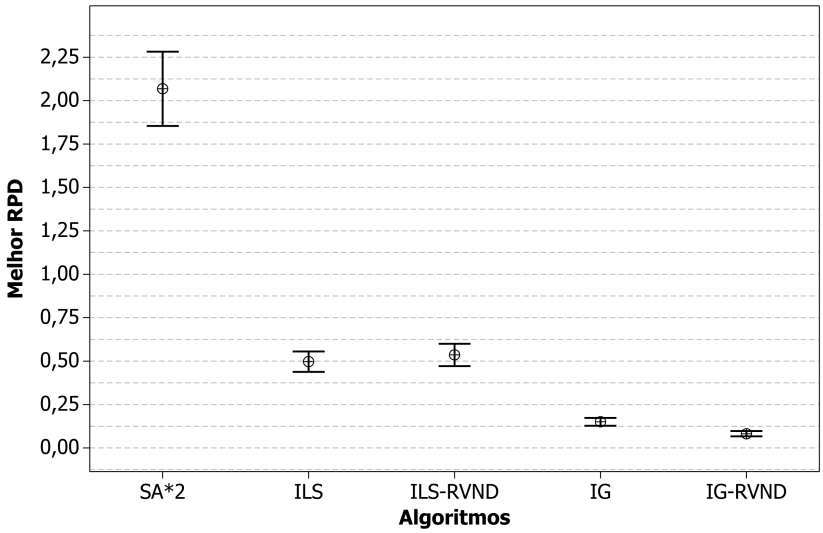


**Figura 7.7.** Gráfico de melhores valores e intervalos HSD de Tukey com nível de confiança de 95% para comparação do algoritmo SA*2 e algoritmos propostos.

também o desempenho dos algoritmos quanto ao tempo de execução gasto.

Para verificar em quanto tempo cada algoritmo atinge a melhor solução, isto é, como se dá a convergência dos métodos, foram selecionadas duas instâncias com diferentes quantidades de tarefas e máquinas. A primeira instância possui 35 tarefas e 7 máquinas e a segunda instância escolhida possui 150 tarefas e 10 máquinas. No experimento, a partir do *makespan* da solução inicial, a cada melhora obtida pelo algoritmo foi armazenado o tempo gasto para obter a solução. Ao final, foi gerado o gráfico: *tempo computacional (segundos) versus makespan* para cada instância. Os resultados são apresentados nas Figuras 7.8 e 7.9.

Ao analisar o gráfico da Figura 7.8, é fácil observar que o algoritmo IG-RVND encontra a melhor solução mais rapidamente que os outros algoritmos propostos. O IG-RVND obtém a melhor solução em menos de 0,05 segundos (50 milissegundos), enquanto os algoritmos ILS-RVND, IG e ILS gastam aproximadamente 0,1; 0,2 e 0,4 segundos, respectivamente. Nota-se, também, que embora o algoritmo IG apresenta uma convergência maior nos primeiro instantes de execução, este algoritmo demorou mais tempo para encontrar a melhor solução do que o algoritmo ILS-RVND. Sendo assim, neste caso, entende-se que o ILS-RVND tem a possibilidade de encontrar a melhor solução mais rápido.

Como pode ser observado na Figura7.9, o IG-RVND consegue encontrar soluções boas com o menor tempo de execução se comparadas ao outros algoritmos

propostos. Mas neste caso, o IG foi o primeiro a encontrar a melhor solução com aproximadamente 40 segundos. Em seguida, foram os algoritmos IG-RVND e ILS-RVND que atingiram a melhor solução com aproximadamente 60 segundos. E em último lugar, o ILS que gastou aproximadamente 130 segundos para encontrar a melhor solução.

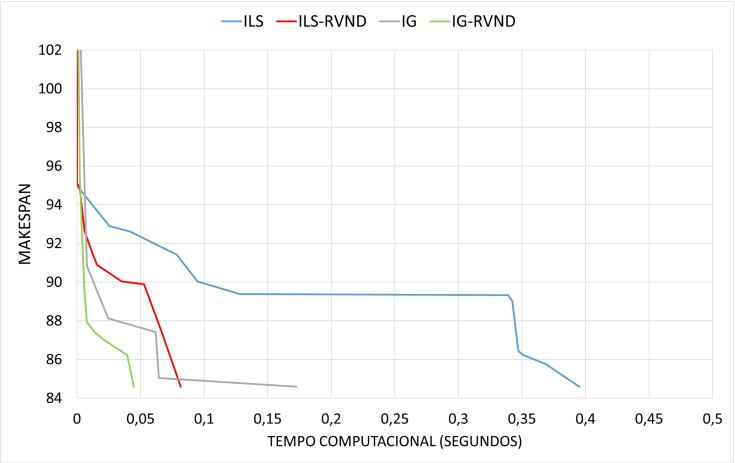


**Figura 7.8.** Convergência das soluções da instância *Ni_35_7-1_1_5* ao decorrer do tempo usando os algoritmos ILS, ILS-RVND, IG, IG-RVND.

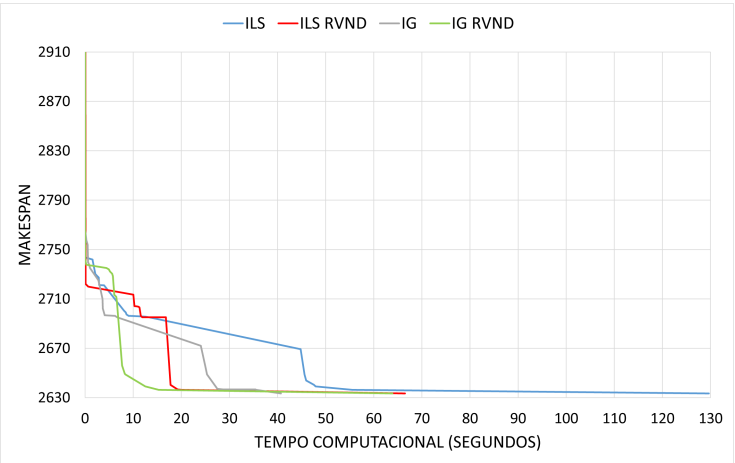


**Figura 7.9.** Convergência das soluções da instância *Ni_150_10_2_2_2* ao decorrer do tempo usando os algoritmos ILS, ILS-RVND, IG, IG-RVND.

# Capítulo 8

# Conclusões

Este trabalho aborda o problema de sequenciamento de tarefas em máquinas paralelas não-relacionadas, proposto por Ruiz-Torres et al. (2013). O problema em questão considera a minimização do máximo tempo de conclusão do processamento das tarefas, levando em conta que as tarefas provocam certos desgastes nas máquinas. Estes desgastes dependem da sequência de tarefas previamente processadas nas máquinas. Este é um importante problema que é encontrado em uma ampla variedade de situações práticas.

Uma vez que a formulação matemática do problema $Rm|Sdd|C_{max}$, proposta por Ruiz-Torres et al. (2013) é um modelo de programação inteiro não-linear, foi proposto neste trabalho um novo modelo de programação inteira mista (MPIM) baseado na geração de padrões (conjunto de tarefas). Utilizando o software IBM CPLEX, resolveu-se o MPIM para 500 instâncias de um total de 900 instâncias de médio porte. Para as outras 400 instâncias restantes, foi elaborado uma outra abordagem que utiliza padrões pré definidos, usados como dados de entrada do solver. Esta última abordagem não garante que a solução encontrada é ótima, mas cria a possibilidade de encontrar uma combinação melhor entre os padrões não avaliados pelo algoritmo. Os resultados para as 400 instâncias foram satisfatórios, uma vez que a abordagem encontra soluções de qualidade.

O problema $Rm|Sdd|C_{max}$ é classificado como NP-difícil, razão esta que torna justificável o estudo de algoritmos heurísticos que obtenham soluções próximas da ótima em tempo computacional razoável. Foram propostos quatro algoritmos heurísticos. O primeiro deles é baseado na meta-heurística *Iterated Local Search* (ILS). O segundo é um método híbrido que usa *Variable Neighborhood Descent* (VND) com a ordenação da vizinhança randômica (RVND) na fase da busca local. O terceiro é baseado na meta-heurística *Iterated Greedy* (IG). E o último, também é um método

híbrido que combina o IG com o RVND. Os algoritmos ILS e ILS-RVND utilizam um eficiente esquema de controle dinâmico do tamanho da perturbação. Da mesma forma, os algoritmos IG e IG-RVND usam, nas fases de destruição e reconstrução, um eficiente controle dinâmico da quantidade de tarefa que devem ser removidas e reinseridas. Estas abordagens são relativamente simples e fáceis de implementar.

A meta-heurística ILS usa o método de busca local que faz modificações da solução corrente até que um ótimo local seja alcançado. O ILS utiliza também o procedimento de perturbação, o qual busca escapar do mínimo local atual alterando parte da solução, mas mantendo as boas características da solução em questão para encontrar um novo mínimo local possivelmente melhor. Na meta-heurística IG, em cada iteração, são aplicados em sequência um procedimento de destruição e reconstrução. A fase de destruição remove aleatoriamente alguns componentes da solução atual, levando a uma solução parcial. Em seguida, uma nova solução completa é reconstruída de forma gulosa a partir da solução parcial. Nos algoritmos híbridos (ILS-RVND e IG-RVND), a busca local RVND utiliza uma ordenação randômica da vizinhança, em que o movimento para gerar soluções vizinhas é selecionado aleatoriamente.

O algoritmo *Simulated Annealing* (SA*) proposto por Ruiz-Torres et al. (2013) foi reimplementado e a versão usada para comparação é denominada SA*2. Vale ressaltar que o SA*2 apresenta desempenho superior ao SA*. Tal divergência é explicada pela diferença entre hardware e linguagem de programação utilizados no artigo do Ruiz-Torres et al. (2013) e neste trabalho. Os algoritmos foram testados em 900 instâncias de médio porte, encontradas na literatura (Ruiz-Torres et al., 2013), e 900 instâncias de grande porte, propostas nestes trabalho. Os resultados obtidos foram validados através de testes estatísticos.

Os resultados obtidos indicam claramente que os algoritmos propostos são altamente eficientes quando comparados com o algoritmo SA*2. Cada algoritmo produz um melhoramento significativo das soluções encontradas, mostrado que as heurísticas propostas são mais efetivas que o algoritmo SA*2. Além disso, os resultados mostram que os algoritmos híbridos geraram melhores resultados que os algoritmos baseados em apenas uma meta-heurística, mostrando a eficiência da busca local RVND.

É importante destacar que o algoritmo SA*2 baseia-se na aceitação de soluções de qualidade inferior com uma certa probabilidade. Já na calibração dos algoritmos propostos, ficou bem claro que não se deve considerar soluções de qualidade inferior a solução corrente no critério de aceitação. Logo, os testes mostraram que estes algoritmos encontram soluções melhores quando há uma intensificação da solução

muito forte, ou seja, se apenas soluções melhores que a atual são aceitas. Entende-se, assim, que uma das razões pelo algoritmo SA*2 apresentar um desempenho inferior aos demais algoritmos é fato de aceitar soluções de baixa qualidade. Já que, neste caso, ao aceitar uma solução ruim pode-se perder as características boas da solução tornando-se difícil readquiri-las.

Além disso, no método de busca local do SA*2 cada estrutura de vizinhança é aplicada apenas uma vez para obter o mínimo local. Por outro lado, no método de busca local dos algoritmos propostos, o procedimento repete-se enquanto obtiver melhoria na função objetivo da solução. Compreende-se, então, que esta estratégia de intensificar a procura pelo ótimo local a cada iteração do algoritmo é relevante para obter o aperfeiçoamento das soluções.

Pode-se concluir que os algoritmos heurísticos propostos neste trabalho são os métodos do *estado-da-arte* para resolver este tipo de problema de sequenciamento, como pode ser visto através da comparação dos resultados obtidos com a melhor meta-heurística disponível na literatura em um amplo conjunto de instâncias. Da mesma forma, acredita-se que os resultados apresentados são os melhores resultados que existem até o momento na literatura para o problema em estudo.

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

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.

Assis, L. P. D. (2012). *Investigação de metaheurísticas aplicadas aos problemas de roteamento de veículos multiobjetivo com serviços de coleta e entrega*. PhD thesis, Universidade Federal de Minas Gerais.

Browne, S. & Yechiali, U. (1990). Scheduling deteriorating jobs on a single processor. *Operations Research*, 38(3):495--498.

Brucker, P. (1998). Parallel machines. Em *Scheduling Algorithms*, pp. 101--144. Springer.

Chase, R.; Jacobs, F. & Aquilano, N. (2006). *Administração da produção e operações para vantagens competitivas*. McGraw-Hill.

Cheng, C.; Feiring, B. & Cheng, T. (1994). The cutting stock problem a survey. *International Journal of Production Economics*, 36(3):291--305.

Cheng, W.; Guo, P.; Zhang, Z.; Zeng, M. & Liang, J. (2012). Variable neighborhood search for parallel machines scheduling problem with step deteriorating jobs. *Mathematical Problems in Engineering*, 2012.

Della Croce, F.; Garaix, T. & Grosso, A. (2012). Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem. *Computers & Operations Research*, 39(6):1213--1217.

Dong, X.; Huang, H. & Chen, P. (2009). An iterated local search algorithm for the permutation flow-shop problem with total flowtime criterion. *Computers and Operations Research*, 36:1664--1669.

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145--159.

Entringer, T. C. & Arica, G. G. M. d. (2013). Introdução ao cplex©: Otimização de modelos matemáticos. *Confict*.

Gara-Ali, A.; Espinouse, M.-L. & Finke, G. (2014). Unrelated parallel-machine scheduling with deterioration effects and multi-maintenance activities. Em *CIE-44-44th International Conference on Computers & Industrial Engineering*.

Garey, M. R. & Johnson, D. S. (1985). A 71/60 theorem for bin packing. *Journal of Complexity*, 1(1):65--106.

Gilmore, P. C. & Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849--859.

Góes, A. R. T. (2005). Otimização na distribuição da carga horária de professores: método exato, método heurístico, método misto e interface. *Universidade Federal do Paraná, Curitiba, Paraná*.

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.

Guo, P.; Cheng, W. & Wang, Y. (2015). Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm. *Engineering Optimization*, 47(11):1564--1585.

Gupta, J. N. & Gupta, S. K. (1988). Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering*, 14(4):387--393.

Gupta, S.; Kunnathur, A. & Dandapani, K. (1987). Optimal repayment policies for multiple loans. *Omega*, 15(4):323--330.

Hansen, P. & Mladenović, N. (1999). An introduction to variable neighborhood search. Em *Meta-heuristics*, pp. 433--458. Springer.

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

Hansen, P.; Mladenović, N. & Pérez, J. A. M. (2008). Variable neighbourhood search: methods and applications. *4OR*, 6(4):319--360.

Hsu, C.-J.; Ji, M.; Guo, J.-Y. & Yang, D.-L. (2013). Unrelated parallel-machine scheduling problems with aging effects and deteriorating maintenance activities. *Information Sciences*, 253:163--169.

Hsu, C.-J. & Yang, D.-L. (2014). Unrelated parallel-machine scheduling with position-dependent deteriorating jobs and resource-dependent processing time. *Optimization Letters*, 8(2):519--531.

Huang, X. & Wang, M.-Z. (2011). Parallel identical machines scheduling with deteriorating jobs and total absolute differences penalties. *Applied Mathematical Modelling*, 35(3):1349--1353.

Huang, X.; Wang, M.-Z. & Ji, P. (2014). Parallel machines scheduling with deteriorating and learning effects. *Optimization Letters*, 8(2):493--500.

Jacob, V. V. et al. (2014). Aplicação de metaheurísticas para problemas de sequenciamento com lotes de tarefas. Master's thesis, Universidade Federal de Viçosa.

Ji, M. & Cheng, T. E. (2009). Parallel-machine scheduling of simple linear deteriorating jobs. *Theoretical Computer Science*, 410(38):3761--3768.

Jiang, S. (2011). Lagrangian relaxation for parallel machine batch scheduling with deteriorating jobs. Em *Management of e-Commerce and e-Government (ICMeCG), 2011 Fifth International Conference on*, pp. 109--112. IEEE.

Johnson, D. S. (1974). Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272--314.

Joo, C. & Kim, B. (2015). Machine scheduling of time-dependent deteriorating jobs with determining the optimal number of rate modifying activities and the position of the activities. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 9(1):JAMDSM0007–JAMDSM0007.

Kunnathur, A. & Gupta, S. (1990). Minimizing the makespan with late start penalties added to processing times in a single facility scheduling problem. *European Journal of Operational Research*, 47:56--64.

Lee, H.-T.; Yang, D.-L. & Yang, S.-J. (2013). Multi-machine scheduling with deterioration effects and maintenance activities for minimizing the total earliness and

tardiness costs. *The International Journal of Advanced Manufacturing Technology*, 66(1-4):547--554.

Lee, W.-C.; Wang, W.-J.; Shiau, Y.-R. & Wu, C.-C. (2010). A single-machine scheduling problem with two-agent and deteriorating jobs. *Applied Mathematical Modelling*, 34(10):3098--3107.

Lopes, H. S.; Rodrigues, L. C. A. & Steiner, M. T. A. (2013). *Meta-Heurísticas em Pesquisa Operacional*. Omnipax, Curitiba, PR, 1 edição.

Lourenço, H. R.; Martin, O. C. & Stützle, T. (2003). *Handbook of Metaheuristics*, chapter Iterated local search, pp. 321--353. F. Glover Eds. Kluwer Academic.

Lourenço, H. R.; Martin, O. C. & Stützle, T. (2010). Iterated local search: Framework and applications. Em *Handbook of Metaheuristics*, pp. 363--397. Springer.

Lustosa, L.; Mesquita, M. A.; Quelhas, O. & Oliveira, R. (2011). *Planejamento e Controle da Produção (Pcp)*. Elsevier Brasil.

Ma, W.-M.; Sun, L.; Liu, S. & Wu, T. (2015). Parallel-machine scheduling with delivery times and deteriorating maintenance. *Asia-Pacific Journal of Operational Research*, 32(04):1550029.

Martello, S. & Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc.

Mazdeh, M. M.; Zaerpour, F.; Zareei, A. & Hajinezhad, A. (2010). Parallel machines scheduling to minimize job tardiness and machine deteriorating cost with deteriorating jobs. *Applied Mathematical Modelling*, 34(6):1498 – 1510.

Melián, B.; Pérez, J. A. M. & Vega, J. M. M. (2003). Metaheuristics: A global view. *Iberoamericana de Inteligencia Artificial*, 19(19):7--28.

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

Mosheiov, G. (2012). A note: Multi-machine scheduling with general position-based deterioration to minimize total load. *International Journal of Production Economics*, 135(1):523--525.

Nelder, J. A. & Mead, R. (1965). A simplex method for function minimization. *The computer journal*, 7(4):308--313.

Papadimitriou, C. H. (2003). *Computational complexity*. John Wiley and Sons Ltd.

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

Ribas, I.; Companys, R. & Tort-Martorell, X. (2013). An efficient iterated local search algorithm for the total tardiness blocking flow shop problem. *International Journal of Production Research*, 51(17):5238--5252.

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.

Ruiz-Torres, A.; Paletta, G. & Pérez, E. (2013). Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects. *Computers & Operations Research*, 40(8):2051--2061.

Russomano, V. (2000). *PCP, planejamento e controle da produção*. Biblioteca Pioneira de administração e negocios. Pioneira.

Shen, W.; Wang, L. & Hao, Q. (2006). Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 36(4):563--577.

Stützle, T. (2006). Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519--1539.

Subramanian, A.; Penna, P. H. V.; Ochi, L. S. & Souza, M. J. F. (2013a). Um algoritmo heurístico baseado em iterated local search para problemas de roteamento de veículos. *Lopes, HS; Rodrigues, LCDA; Steiner, MTA Meta-Heurísticas em Pesquisa Operacional. Omnipax*, pp. 165--180.

Subramanian, A.; Uchoa, E. & Ochi, L. S. (2013b). A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519--2531.

Talbi, E. (2009). *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley.

Toksarı, M. D. & Güner, E. (2009). Parallel machine earliness/tardiness scheduling problem under the effects of position based learning and linear/nonlinear deterioration. *Computers & Operations Research*, 36(8):2394--2417.

Vallada, E.; Ruiz, R. & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350--1373.

Vansteenwegen, P. & Mateo, M. (2014). An iterated local search algorithm for the single-vehicle cyclic inventory routing problem. *European Journal of Operational Research*, 237(3):802--813.

Vansteenwegen, P.; Souffriau, W.; Vanden Berghe, G. & Van Oudheusden, D. (2009). Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281--3290.

Wang, J.-B. & Wei, C.-M. (2011). Parallel machine scheduling with a deteriorating maintenance activity and total absolute differences penalties. *Applied Mathematics and Computation*, 217(20):8093--8099.

Wang, J.-J.; Wang, J.-B. & Liu, F. (2011). Parallel machines scheduling with a deteriorating maintenance activity. *Journal of the Operational Research Society*, 62(10):1898--1902.

Yang, D.-L.; Cheng, T.; Yang, S.-J. & Hsu, C.-J. (2012). Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities. *Computers & Operations Research*, 39(7):1458--1464.

Yang, S.-J. (2011). Parallel machines scheduling with simultaneous considerations of position-dependent deterioration effects and maintenance activities. *Journal of the Chinese Institute of Industrial Engineers*, 28(4):270--280.

Zaccarelli, S. (1979). *Programação e controle da produção*. Pioneira.