UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS E TECNOLOGIAS PARA A SUSTENTABILIDADE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO - PPGEPS

PEDRO LUIS MIRANDA LUGO

# PROGRAMAÇÃO DA PRODUÇÃO EM SISTEMAS *FLOWSHOP* HÍBRIDO COM *BUFFERS* LIMITADOS

Sorocaba, SP
2013

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS E TECNOLOGIAS PARA A SUSTENTABILIDADE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO - PPGEPS

PEDRO LUIS MIRANDA LUGO

# PROGRAMAÇÃO DA PRODUÇÃO EM SISTEMAS *FLOWSHOP* HÍBRIDO COM *BUFFERS* LIMITADOS

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Produção, para a obtenção do título de Mestre em Engenharia de Produção.

**Orientador:** Prof. Dr. Rodolfo Florence Teixeira Jr.

Sorocaba, SP
2013

Lugo, Pedro Luis Miranda
L951p Programação da produção em sistemas *flowshop* híbrido com *buffers* limitados / Pedro Luis Miranda Lugo. – – Sorocaba, 2013.
140 f. : il. (color.) ; 28 cm

Dissertação (mestrado) - Universidade Federal de São Carlos, *Campus* Sorocaba, 2013.
Orientador: Rodolfo Florence Teixeira Jr
Banca examinadora: Vinícius Amaral Armentano, Eli Angela Vitor Toso
Bibliografia

1. Controle de produção. 2. Programação da produção. 3. Programação heurísitica. I. Título. II. Sorocaba - Universidade Federal de São Carlos.

CDD 658.5038

Ficha catalográfica elaborada pela Biblioteca do *Campus* de Sorocaba.

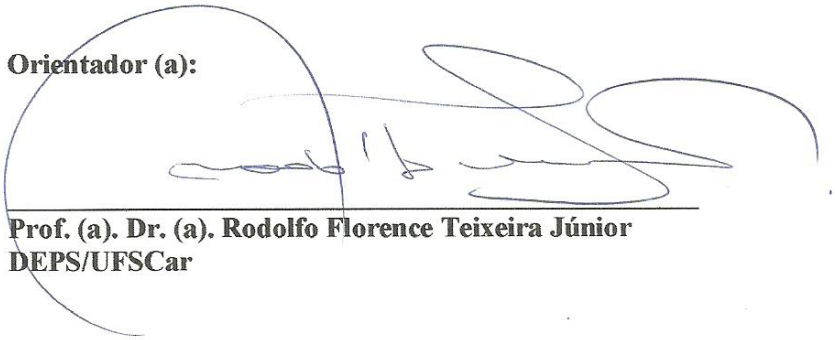
**PEDRO LUÍS MIRANDA LUGO**

# "PROGRAMAÇÃO DA PRODUÇÃO EM SISTEMAS *FLOWSHOP* HÍBRIDO COM *BUFFERS* LIMITADOS"

**Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção do Centro de Ciências e Tecnologias para a Sustentabilidade da Universidade Federal de São Carlos para obtenção do título de mestre em Engenharia de Produção, Área de Concentração: Gestão de Operações.**

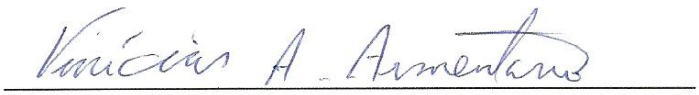
**Sorocaba, 12 de setembro de 2013**

**Orientador (a):**

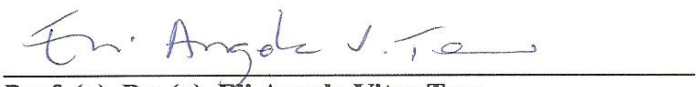


**Prof. (a). Dr. (a). Rodolfo Florence Teixeira Júnior**
**DEPS/UFSCar**

**Examinadores (as):**



**Prof. Dr. Vinícius Amaral Armentano**
**Densis/Unicamp**



**Prof. (a). Dr. (a). Eli Angela Vitor Toso**
**DEPS/UFSCar**

*Aos meus amados pais, Pedro e Yadira.*
*Aos meus irmãos Pedro Manuel, Pedro José e Mariarrosa.*
*Apesar da distância, o seu amor, apoio e compreensão estão sempre comigo.*
*Este sonho alcançado é dedicado a vocês, meu maior tesouro.*

## AGRADECIMENTOS

Agradeço a Deus por abençoar a minha família e todas as oportunidades oferecidas.

À minha família pelas suas incansáveis expressões de amor, carinho e confiança. Nos momentos difíceis, vocês foram a minha maior motivação para seguir em frente. Amo vocês com todas as minhas forças.

À Karim por todo o seu amor, força e compreensão. Esta longa caminhada, cheia de alegrias e tristezas, tem sido mais fácil com você ao meu lado. Obrigado por todas as suas expressões de amor e carinho, mas também pela sua compreensão e paciência em momentos difíceis. Obrigado por todo *more*!

Aos meus tios e primos, especialmente às minhas tias Fabiola, Norma e Inirida, que rezam, choram, riem e comemoram pelo seu sobrinho.

Ao prof. José e à sua esposa Rosane. Esta experiência não teria sido possível sem a sua ajuda. Para vocês sempre terei sentimentos de respeito, carinho, admiração e gratidão. Deus os abençoe sempre!

À Maritha, Nadya, Renata, Natália, Diego, Alessandro e Arthur pela amizade e os momentos compartilhados. Obrigado por fazer a minha vida longe de casa mais fácil e amena.

Ao meu grande amigo Lucas, o *makarrão*, pela amizade incondicional e o ensino do exótico português *makarroniano*. A sua atitude, sempre positiva e extrovertida, e seu jeito simples de ver a vida são admiráveis. Meus mais sinceros sentimentos de gratidão para você.

Aos meus amigos na Colômbia, especialmente aos meus amigos da vida inteira: Fanor, Jesus David, Samir e Yeimer. Obrigado por apoiar e compreender ao amigo ausente, mas principalmente, pela sua valiosa e eterna amizade.

Ao meu orientador, prof. Rodolfo Florence, pela extrema confiança e por todo o tempo dedicado.

A todos os professores, funcionários e amigos da UFSCar *campus* Sorocaba e do Programa de Pós-graduação em Engenharia de Produção, especialmente à Eli e à Erica, pela ajuda, colaboração e total disposição.

À CAPES pelo apoio financeiro, sem o qual esta dissertação não teria sido realizada.

## RESUMO

MIRANDA, P.L. **Programação da produção em sistemas *Flowshop* híbrido com *buffers* limitados.** 2013. 140f. Dissertação (Mestrado em Engenharia de Produção) – Centro de Ciências e Tecnologias para a Sustentabilidade, Universidade Federal de São Carlos, Sorocaba, 2013.

Este trabalho estuda o problema de programação da produção em sistemas *Flowshop* híbrido. Nesta configuração de produção há um conjunto de tarefas que deve ser processado em um conjunto de estações, nas quais um determinado número de máquinas paralelas encontra-se disponível para o processamento das tarefas. Todas as tarefas devem ser processadas seguindo o mesmo fluxo de produção, desde a primeira até a última estação. Cada tarefa deve ser processada em uma máquina de cada estação e cada máquina pode processar, no máximo, uma tarefa por vez. Algumas restrições comumente encontradas em sistemas de produção reais, como máquinas paralelas não relacionadas, *buffers* limitados, tempos de preparação dependentes da sequência (antecipatórios e não antecipatórios), elegibilidade de máquinas, tempos de transporte e tempos de liberação das máquinas, também são consideradas. O critério de otimização é o *makespan*, cuja minimização está diretamente relacionada com a utilização eficiente dos recursos de produção. Um modelo de programação inteira mista é proposto e resolvido através do *solver* comercial CPLEX. Os resultados da avaliação computacional indicam que o modelo é viável somente para resolver instâncias de até nove tarefas e cinco estações. Desta forma, para resolver instâncias de maior tamanho (50-100 tarefas), várias heurísticas e uma meta-heurística de busca local iterada (ILS, *Iterated Local Search*) são propostas e avaliadas computacionalmente. Os resultados indicam que o ILS é capaz de obter soluções de boa qualidade em curtos tempos computacionais.



**Palavras-chave:** *Flowshop* híbrido. Programação da produção. *Buffers* limitados. Heurísticas. Busca local iterada.

## ABSTRACT

This research studies the hybrid flowshop scheduling problem. In this production configuration, we have a set of jobs that has to be processed in a set of stages. At every stage we have a set of parallel machines available to process the jobs. All jobs have to be processed following the same production flow, from the first to the last stage. Every job has to be processed on one machine at each stage and each machine can process at most one job at a time. Some constraints commonly found in real production systems as unrelated parallel machines, limited buffers, sequence-dependent setup times (both anticipatory and non-anticipatory), machine eligibility, transportation times and release times for machines are also taken into account. The optimization criterion is the makespan, whose minimization is related to the efficient use of production resources. A mixed integer programming model is proposed and solved by the commercial solver CPLEX. The computational evaluation results indicate that the model is suitable just to solve instances up to nine jobs and five stages. Therefore, to solve larger instances (50-100 jobs), several heuristics and an iterated local search (ILS) algorithm are proposed and evaluated computationally. The results indicate that the ILS is able to obtain good quality solutions in short computation times.


**Keywords:** Hybrid flowshop. Scheduling. Limited buffers. Heuristics. Iterated local search.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| AI | *Adjacent Interchange* |
| ANOVA | *Analysis of variance* |
| APT | *Average Processing Time* |
| *Better* | Critério de aceitação *Better* |
| B&B | *Branch and Bound* |
| CDS | Heurística de Cambell, Dudek e Smith |
| CPU | *Central Processing Unit* |
| ECT | *Earliest Completion Time* |
| FAM | *First Available Machine* |
| GA | *Genetic Algorithm* |
| GAMS | *General Algebraic Modeling System* |
| *GeigerPert* | Operador de perturbação de Geiger |
| GL | Operador de perturbação *Giant Leap* |
| HFFL | *Hybrid Flexible Flowshop* |
| HFS | *Hybrid Flowshop* |
| IG | *Iterated Greedy Algorithm* |
| ILS | *Iterated Local Search* |
| LPT | *Longest Processing Time* |
| LSD | *Least Significant Difference* |
| *LSMC* | Critério de aceitação tipo recozimento simulado |
| *LSNaderi* | Algoritmo de busca local de Naderi |
| *LSUrlings* | Algoritmo de busca local de Urlings |
| MIP | *Mixed Integer Programming* |
| NEH | Heurística de Nawaz, Enscore e Ham |
| NEHKK1 | Heurística NEH modificada por Kalczynski e Kamburowski |
| PCB | *Printed Circuit Board* |
| PF | Heurística *Profile Fitting* |
| PFE | Combinação das heurísticas NEH e PF |
| PF-NEH | Combinação das heurísticas NEH e PF |
| RPD | *Relative Percentage Deviation* |
| *RW* | Critério de aceitação *Random Walk* |
| *RZ* | Algoritmo de busca local Rajendran e Ziegler |
| SA | *Simulated Annealing* |
| SPT | Shortest *processing time* |
| *T* | Parâmetro temperatura do critério LSMC |
| TAPT | *Total average processing time* |
| TS | *Tabu Search* |
| TSP | *Traveling Salesman Problem* |
| *TypeNEH* | Operador de perturbação do tipo NEH |

**SUMÁRIO**

# 1. Introdução

A programação da produção é um processo de tomada de decisões que tem um papel crucial em empresas de manufatura e de serviços. No ambiente competitivo no qual as empresas estão inseridas atualmente, a programação efetiva da produção tem se convertido em um requerimento indispensável para sobreviver no mercado, pois as empresas tem que atender prazos de produção e entrega prometidos aos clientes utilizando eficientemente os recursos de produção disponíveis.

Historicamente, a programação da produção passou a ser considerada a sério nos ambientes de manufatura no começo do século XX, com trabalhos de Henry Gantt e outros pioneiros. No entanto, passaram muitos anos até a aparição das primeiras publicações na literatura da pesquisa operacional e da engenharia industrial, algumas das quais apareceram na *Naval Research Logistic Quarterly* no início dos anos 50. Pouco depois, durante os anos 60, uma quantidade significativa de trabalho foi feita em programação dinâmica e formulações de programação inteira de problemas de programação da produção. Depois da aparição da teoria da complexidade, nos anos 70, as pesquisas focaram principalmente a hierarquia de complexidade dos problemas de programação da produção. Já nos anos 80, tanto a teoria quanto a prática seguiram direções diferentes, com uma notável atenção em problemas de programação da produção estocásticos. Finalmente, com a popularização dos computadores, sistemas de programação da produção começaram a serem desenvolvidos com o objetivo de obter programas de produção utilizáveis na prática.

Em termos gerais, a programação da produção refere-se à alocação de recursos a tarefas durante determinados períodos de tempo, com o objetivo de otimizar um, ou mais, objetivos específicos (PINEDO, 2008).

Dependendo da situação, os recursos e as tarefas podem tomar diversas formas. Assim, os recursos poderiam ser máquinas em um centro de processamento, unidades centrais de processamento (CPU, *Central Processing Unit*), pistas em um aeroporto, mecânicos em uma loja de reparação de automóveis, entre outros. As tarefas poderiam ser operações em um processo de manufatura, execuções de programas de computador, pousos e decolagens no aeroporto, reparação de automóveis e assim sucessivamente. Cada tarefa poderia ter certo nível de prioridade, um tempo de inicio mais cedo e uma data de entrega. Igualmente, o objetivo pode tomar diferentes formas, como minimizar o tempo de finalização da última tarefa ou minimizar o número de

tarefas que são finalizadas depois da sua data de entrega (LEUNG, 2004; PINEDO, 2005).

Dentro de qualquer organização, seja de manufatura ou serviços, a programação da produção deve interagir com muitas outras funções. Estas interações são dependentes do sistema e poderiam diferir significativamente de uma situação para outra. Por exemplo, em indústrias de manufatura, a programação da produção é afetada pelo processo de planejamento da produção, o qual define o planejamento no médio e longo prazo de toda a organização. Este planejamento tático e estratégico visa otimizar o *mix* de produção geral da empresa e a alocação de recursos no longo prazo baseado nos níveis de estoque, previsões de demanda e requerimentos de recursos de produção. Deste modo, a programação da produção pressupõe que o planejamento tático e estratégico já tenha sido feito e que os produtos e quantidades que devem ser fabricados são conhecidos, assim como disposição e quantidade de recursos disponíveis.

Igualmente, no nível do chão de fábrica, as tarefas frequentemente devem ser processadas pelas máquinas em um ou mais centros de trabalho em uma dada ordem ou sequência. As tarefas podem ter que esperar por máquinas que estão ocupadas e interrupções podem acontecer quando tarefas de alta prioridade chegam à máquina e devem ser processadas imediatamente. Eventos inesperados, como avarias de máquinas e tempos de processamento mais longos, também devem ser levados em conta, pois poderiam impactar significativamente o programa de produção. Em tais situações, um programa de produção detalhado ajuda a manter a eficiência e controle das operações.

Dependendo da configuração dos recursos de produção no chão de fábrica, diversos problemas de programação da produção podem ocorrer. Desta forma, tem-se:

- Problemas de uma máquina: somente existe uma máquina e cada tarefa só tem uma operação. Neste problema é preciso estabelecer a sequência na qual as tarefas deveriam ser processadas.
- Problemas de máquinas paralelas: existem duas ou mais máquinas arranjadas em paralelo e cada tarefa consiste de uma única operação. Em geral, deve-se designar e sequenciar as tarefas nas máquinas.
- Problemas do tipo *shop*: Nestes problemas existem várias estações que devem ser visitadas para completar as tarefas. Adicionalmente, para cada tarefa, existe uma relação de precedência entre as estações. Esta relação de precedência é denominada rota e dependendo de suas características os problemas podem ser classificados como *Flowshop*, no qual todas as tarefas

tem a mesma rota; *Job shop*, no qual cada tarefa tem sua própria rota a seguir pelo chão de fábrica; ou *Open shop*, no qual as operações podem realizar-se em qualquer ordem.

Esta dissertação estuda uma generalização do *Flowshop*, denominada *Flowshop* Híbrido (HFS, *Hybrid Flowshop*). Nesta configuração de produção há um conjunto de tarefas que deve ser processado em um conjunto de estações, nas quais um determinado número de máquinas paralelas encontra-se disponível para o processamento das tarefas. Todas as tarefas devem ser processadas seguindo o mesmo fluxo de produção, desde a primeira até a última estação. Cada tarefa deve ser processada por uma máquina de cada estação e cada máquina pode processar, no máximo, uma tarefa por vez.

Este tipo de sistema de produção pode ser encontrado em diversos setores de manufatura, como na indústria eletrônica, indústria de papel, industrial têxtil, indústria petroquímica, manufatura de filmes fotográficos, indústria de revestimentos cerâmicos, produção de concreto, entre muitas outras, o que denota sua ampla relevância prática. Uma descrição mais específica e detalhada do problema objeto de estudo nesta dissertação é apresentada no Capítulo 3.

## 1.1. Objetivos

O principal objetivo desta dissertação é propor um método de solução viável para o problema de programação da produção em sistemas *Flowshop* híbrido com máquinas paralelas não relacionadas, *buffers* limitados, tempos de preparação dependentes da sequência (antecipatórios e não antecipatórios), elegibilidade de máquinas, tempos de transporte e tempos de liberação das máquinas, visando minimizar o cumprimento total do programa de produção. Para isto, pretende-se propor um modelo de programação inteira mista, que permita a obtenção de soluções ótimas para o problema tratado, e também métodos heurísticos e meta-heurísticos, através dos quais possam ser obtidas soluções de boa qualidade em tempos computacionais aceitáveis.

## 1.2. Justificativa

O problema de programação da produção em *Flowshop* híbrido envolve aspectos relevantes tanto teóricos, quanto práticos. Do ponto de vista teórico, problemas de programação da produção apresentam dificuldades matemáticas similares àquelas encontradas em outros ramos da otimização combinatória e, portanto envolvem um grande desafio no desenvolvimento de métodos de solução exatos ou aproximados. Do

ponto de vista prático, como destacado anteriormente, sistemas de produção próprios de diferentes tipos de indústria podem ser modelados como um *Flowshop* híbrido, portanto o estudo do problema e a formulação de métodos de solução eficientes são de vital importância para a obtenção de programas de produção adequados, a utilização eficiente dos recursos de produção e o atendimento das necessidades do cliente no momento indicado.

Além disso, muitos pesquisadores tem indicado a existência de um *gap* entre teoria e prática da programação da produção. Recentes revisões da literatura têm destacado a necessidade de progredir no estudo de problemas mais realistas, considerando conjuntamente várias restrições e características próprias de ambientes de produção reais. Estes requerimentos da literatura motivam e incentivam o estudo de um *Flowshop* híbrido complexo e de relevância prática, com o objetivo de diminuir o *gap* entre teoria e prática da programação da produção.

## 1.3. Metodologia

Seguindo Bertrand e Fransoo (2002), o presente estudo pode ser considerado uma pesquisa quantitativa axiomática normativa.

A pesquisa é axiomática porque é orientada por um modelo idealizado devidamente definido para o problema abordado. Adicionalmente, a preocupação principal é obter soluções para esse modelo e, a partir de tais soluções, obter informação acerca da estrutura e comportamento do modelo.

Por outro lado, a pesquisa é normativa porque pretende desenvolver e comparar métodos e estratégias para encontrar a solução ótima, ou quase ótima, do problema definido.

## 1.4. Estrutura do trabalho

O trabalho está organizado da seguinte maneira: O Capítulo 2 apresenta a revisão da literatura, abordando restrições de *buffers* limitados, máquinas paralelas não relacionadas e tempos de preparação dependentes da sequência, as quais são consideradas importantes para o problema estudado. Antes disto, é definida a notação a ser utilizada ao longo do texto, assim como se apresentam brevemente as mais recentes revisões e classificações feitas acerca da programação da produção em *Flowshop* Híbridos.

O Capítulo 3 apresenta a descrição do problema de estudo. A fim de contextualizar o problema, o *Flowshop* híbrido em sua forma mais básica é discutido

inicialmente. Em seguida, o *Flowshop* híbrido com todas as características e restrições consideradas neste trabalho é formalmente definido. Este capítulo finaliza com a apresentação de um exemplo ilustrativo que permite visualizar as diferentes restrições estudadas.

No Capítulo 4 é proposto um modelo de programação inteira mista para o problema estudado. Os resultados da avaliação computacional sobre um extenso conjunto de instâncias são reportados e discutidos.

Métodos de solução para o problema de estudo são propostos nos capítulos 5 e 6. Em particular, no capítulo 5 são apresentadas e avaliadas computacionalmente diversas heurísticas para o *Flowshop* as quais são adaptadas e utilizadas na resolução do problema. O capítulo 6 aborda o desenvolvimento e avaliação computacional de um método meta-heurístico para resolver o problema. Resultados de extensivos estudos de parametrização do algoritmo, baseados em técnicas de delineamento experimental e análises estatísticas, são apresentados.

Finalmente, o capítulo 7 apresenta as principais conclusões do trabalho, assim como futuros tópicos de pesquisa.

## 2. Revisão da Literatura

Um problema de programação da produção é determinado pelo número de tarefas e operações a serem processadas, pelo número e tipo de máquinas disponíveis, pelo padrão de fluxo das operações nas máquinas e pelo critério de otimização com que se avalia uma solução (CONWAY; MAXWELL; MILLER, 1967). Comumente, esta informação é apresentada de forma compacta através da notação $\alpha|\beta|\gamma$, na qual $\alpha$ descreve o ambiente de máquinas, ou ambiente de produção, e só tem uma entrada; $\beta$ fornece detalhes das características e restrições de processamento, e poderia não ter entrada, uma única entrada ou múltiplas entradas; e $\gamma$ descreve o objetivo a ser minimizado e, comumente, tem uma única entrada. Esta notação foi inicialmente proposta por Graham et al. (1979), mas tem evoluído constantemente e pode ter algumas variações dependendo dos diferentes autores.

Vignier, Billaut e Proust (1999) propuseram uma variante na qual o parâmetro $\alpha$ é representado da forma $\alpha_1\alpha_2,\left(\alpha_3\alpha_4^{(1)},\alpha_3\alpha_4^{(2)},\ldots,\alpha_3\alpha_4^{(\alpha_2)}\right)$, tal que $\alpha_1$ define a estrutura do chão de fábrica; $\alpha_2$ representa o número de estações do sistema; e $\alpha_3\alpha_4^{(i)}$ indica que há $\alpha_4$ máquinas do tipo $\alpha_3$ na estação $i$, $i\in\{1,\ldots,\alpha_2\}$. A seguir descrevem-se detalhadamente as diversas alternativas que cada componente do parâmetro $\alpha$ pode assumir.

O parâmetro $\alpha_1\in\{\varnothing,P,Q,R,F,HF,J,HJ,O\}$, tal que:

- $\varnothing$ ou 1 (Máquina Única): é o caso mais simples de todos os possíveis ambientes de máquinas e é um caso especial de outros ambientes mais complicados. Aqui, uma única máquina está disponível para o processamento de $n$ tarefas. Embora pareça simples, sua resolução é de grande importância, pois seu estudo possibilita a resolução de ambientes de produção mais complicados.
- $P,Q,R$ (Máquinas Paralelas): dependendo da velocidade de processamento, as máquinas podem ser classificadas como idênticas ($P$), uniformes ($Q$) ou não relacionadas ($R$). Se todas as máquinas tem igual velocidade de processamento, então são chamadas de idênticas. Se as máquinas diferem em velocidade, mas a velocidade de cada máquina é constante e não depende das tarefas, então são chamadas de uniformes. Finalmente, se a velocidade das máquinas depende de cada tarefa, então são chamadas de não relacionadas. Cada tarefa precisa de uma única

operação, a qual pode ser realizada por qualquer uma das máquinas paralelas disponíveis.

- $F$ (*Flowshop)*: um conjunto de $n$ tarefas deve visitar um conjunto de $m$ máquinas arranjadas em série. Cada tarefa deve ser processada em cada uma das $m$ máquinas na mesma ordem, desde a primeira até a última máquina.
- $HF$ (*Flowshop* Híbrido): é uma combinação das duas configurações anteriores. Cada tarefa deve passar por um conjunto de estações $M = \{1,\ldots,m\}$, todas na mesma ordem. Em cada estação $i$, $i \in M$, um conjunto $M_i = \{1,\ldots,m_i\}$ de máquinas paralelas está disponível para processamento. Cada tarefa deve ser processada por uma das $m_i$ máquinas disponíveis em cada estação. O HFS também é conhecido na literatura como *Flowshop* Flexível ou *Flowshop* com múltiplos processadores. Uma generalização deste ambiente de produção acontece quando nem cada estação deve ser visitada por cada tarefa, ou seja, algumas tarefas podem "saltar" algumas estações nas quais não precisam ser processadas. Neste caso, o *Flowshop* Híbrido passa a ser chamado de *Flowshop* Híbrido Flexível (HFFL, *Hybrid Flexible Flowshop*).
- $J$ (*Job shop*): há $m$ máquinas disponíveis no chão de fábrica e cada tarefa tem sua própria rota predeterminada de processamento. O modelo *Job shop* mais simples considera que cada tarefa pode ser processada em uma máquina particular somente uma vez ao longo de seu percorrido pelo sistema. Note que no *Job shop* as tarefas não devem ser processadas nas máquinas seguindo a mesma ordem, tal como acontece no *Flowshop* e no *Flowshop* Híbrido.
- $HJ$ (*Job shop* Híbrido): este ambiente de produção é uma generalização dos ambientes *Job shop* e máquinas paralelas. Neste caso há $m$ estações e cada estação tem um número de máquinas paralelas. Cada tarefa tem sua rota predeterminada pelo chão de fábrica e requer processamento somente de uma das máquinas disponíveis em cada estação da sua rota.
- $O$ (*Open shop*): há $m$ máquinas disponíveis no chão de fábrica. As tarefas não tem rota fixa e podem utilizar as máquinas em qualquer ordem.

O parâmetro $\alpha_2$ indica o número de estações do sistema, tal como definido acima.

O parâmetro $\alpha_3 \in \{\varnothing, P, Q, R\}$, enquanto $\alpha_4 \in \{\varnothing, M^{(i)}, M^{(i)}(t)\}$, tal que $M^{(i)}$ indica o número de máquinas na estação $i$, e $M^{(i)}(t)$ é o número de máquinas na estação $i$ no instante de tempo $t$. Quando há várias estações consecutivas com o mesmo número e tipologia de máquinas, os termos $\alpha_3$ e $\alpha_4$ podem ser agrupados como $\left(\alpha_3 \alpha_4^{(i)}\right)_{i=s}^{k}$, sendo $s$ e $k$ os índices da primeira e última estação consecutiva, respectivamente.

Ilustremos a composição do parâmetro $\alpha$ através de um exemplo. Considere-se um *Flowshop* híbrido com cinco estações, no qual há duas máquinas paralelas idênticas na primeira e segunda estação e há três máquinas paralelas idênticas na terceira, quarta e quinta estação, respectivamente. Esta configuração de produção pode ser representada como $HF5, \left(P2^{(1)}, P2^{(2)}, P3^{(3)}, P3^{(4)}, P3^{(5)}\right)$ ou, de forma mais compacta, $HF5, \left(\left(P2^{(i)}\right)_{i=1}^{2}, \left(P3^{(i)}\right)_{i=3}^{5}\right)$.

Por outro lado, a solução de um problema de programação da produção deve satisfazer certas restrições, as quais devem ser definidas no campo $\beta$. Consideremos as mais comuns:

- $r_j$ (Datas de liberação das tarefas): representa o tempo no qual a tarefa $j$ chega ao sistema, ou seja, é o tempo mais cedo no qual a tarefa pode começar seu processamento. Em contraste, as datas de entrega não são explicitamente especificadas, pois a função objetivo usualmente indica a presença deste tipo de restrições.
- *prmp* (Preempção): indica que o processamento de uma tarefa pode ser interrompido em qualquer instante de tempo, de modo que outra tarefa seja processada. O processamento realizado sobre a tarefa interrompida não é perdido, isto é, quando a tarefa retorna à máquina só necessitará ser processada pelo tempo que restava para sua finalização.
- *prec* (Precedência): indica que uma ou mais tarefas devem ser finalizadas antes que outra tarefa possa começar seu processamento.
- *skip* (Flexibilidade): esta restrição aparece em sistemas de produção nos quais nem todas as estações devem ser visitadas por todas as tarefas, como acontece nos sistemas *Flowshop* híbrido e *Job shop* híbrido. Em algumas situações, como no

HFFL, esta restrição pode não ser incluída dentro do parâmetro $\beta$, pois a estrutura do chão de fábrica define esta restrição implicitamente.

- $lag$ (Tempos de transferência): define o tempo mínimo que deve transcorrer entre a finalização de uma tarefa na estação atual e o começo do processamento da mesma tarefa na seguinte estação.
- $S_{jk}$ (Tempos de preparação dependentes da sequência): o tempo requerido para organizar os recursos necessários (máquinas ou pessoas) para realizar uma tarefa é conhecido como tempo de preparação. Se o tempo de preparação depende somente da tarefa a ser processada, então é chamado de independente da sequência. Por outro lado, se o tempo de preparação depende tanto da tarefa a ser processada quanto da tarefa imediatamente anterior, então é chamado de dependente da sequência. Se tempos de preparação dependentes da sequência são considerados, então $S_{jk}$ representa a preparação que deve ser feita entre o processamento das tarefas $j$ e $k$. Adicionalmente, se o tempo de preparação também depende da máquina, então a notação $S_{ijk}$ é utilizada. Também é importante diferenciar entre preparação antecipatória e não antecipatória. Uma preparação é chamada de antecipatória se pode ser realizada enquanto a tarefa está sendo processada na estação ou máquina anterior. Entretanto, em algumas operações, como posicionamento e ajuste, é requerido que tarefa esteja presente na máquina para que a preparação possa ser feita. Neste caso, a preparação é chamada de não antecipatória.
- $fmls$ (Família ou grupo de produtos): indica que as $n$ tarefas a serem processadas pertencem a diferentes famílias ou grupos de tarefas. Tarefas da mesma família podem ter diferentes tempos de processamento, mas podem ser processadas em uma máquina, uma depois da outra, sem precisar de preparação entre elas. No entanto, se a máquina muda de uma família para outra, então preparação é requerida para a configuração da máquina. Tal como no caso anterior, os tempos de preparação podem ser independentes ou dependentes da sequência de famílias.
- $batch$ (Processamento em bateladas): denota que uma máquina pode processar um número determinado de tarefas simultaneamente, ou seja, a máquina é capaz de processar uma batelada de $b$ tarefas ao mesmo tempo. Os tempos de processamento das tarefas em uma batelada poderiam não ser iguais e a batelada inteira é finalizada somente quando a última tarefa da batelada tenha sido finalizada.

- *brkdwn* (Avarias): indica que as máquinas poderiam não estar disponíveis continuamente. Os períodos nos quais uma máquina não está disponível são considerados como fixos e conhecidos previamente (por exemplo, manutenção preventiva). Estas restrições também são frequentemente referidas como restrições de disponibilidade de máquinas
- $M_j$ (Elegibilidade de máquinas): é característico de ambientes de produção com máquinas paralelas e indica que nem todas as máquinas são capazes de processar a tarefa $j$. Assim, o conjunto $M_j$ contém as máquinas que são capazes de processar a tarefa $j$.
- *block* (Bloqueio): é um fenômeno que pode ocorrer em sistemas de produção nos quais há um espaço limitado de armazenagem (*buffer*) entre duas estações consecutivas. Quando o *buffer* entre duas estações estiver está totalmente ocupado, as tarefas deverão esperar na estação previa até que espaço suficiente no *buffer* seja liberado. Isto implica que uma tarefa finalizada terá que permanecer na máquina na qual foi processada, evitando que esta possa começar o processamento da próxima tarefa.
- *nwt* (sem espera): indica que as tarefas não podem esperar entre duas máquinas consecutivas. Isto implica que o processamento de uma tarefa na primeira máquina deve ser retrasado para garantir que esta possa atravessar o sistema sem ter que esperar por nenhuma máquina. Um caso mais geral se apresenta quando as tarefas podem esperar entre duas máquinas consecutivas, mas somente por um tempo determinado, isto é, existe um tempo de espera limitado. Restrições *nwt* são comuns na indústria de produção de aço, na qual o produto em processo não pode esperar entre estações porque poderia esfriar durante a espera.
- *recrc* (Recirculação): indica que uma tarefa pode passar por uma máquina ou estação mais de uma vez, ou seja, as tarefas podem ser processadas mais de uma vez na mesma máquina ou estação.
- *rm* (Data de liberação das máquinas): indica que as máquinas não necessariamente se encontram disponíveis desde o tempo zero, mas que tem datas de liberação individuais e predeterminadas. Esta restrição impede que as tarefas possam começar seu processamento em uma máquina antes que esta seja liberada.

As restrições descritas acima representam somente algumas das mais comuns e, portanto muitas outras podem ser facilmente encontradas na literatura.

O critério de otimização, definido no campo $\gamma$, é sempre uma função do tempo de finalização das tarefas (PINEDO, 2008). Antes de começar a definição dos diferentes critérios, consideremos algumas definições adicionais:

- $d_j$: denota a data de entrega da tarefa $j$ e representa a data de finalização ou envio prometida ao cliente final. É possível finalizar a tarefa depois desta data, porém uma penalização é incorrida.
- $\omega_j$: é um fator de prioridade que denota a importância da tarefa $j$ relativa às outras tarefas no sistema. Este peso poderia representar o custo de estocagem da tarefa ou o valor já agregado à tarefa ao longo do processo de produção.
- $C_j$: denota o tempo de finalização da tarefa $j$ na ultima estação do sistema.
- $F_j$: denota o tempo de fluxo da tarefa $j$, isto é, o tempo que a tarefa $j$ passa no sistema. Calcula-se como $F_j = C_j - r_j$.
- $L_j$: define o *Lateness* da tarefa $j$ e calcula-se como $L_j = C_j - d_j$. Note que este valor é positivo quando a tarefa $j$ é terminada após da data de entrega, e negativo quando é terminada antes da data de entrega. Portanto, considera-se uma medida da pontualidade da tarefa $j$.
- $T_j$: representa o atraso da tarefa $j$ e calcula-se como $T_j = \max\left(C_j - d_j, 0\right)$.
- $E_j$: indica o adiantamento da tarefa $j$ e calcula-se como $E_j = \max\left(d_j - C_j, 0\right)$.
- $U_j$: indica se a tarefa $j$ está atrasada ou não. Assume o valor de 1 se $C_j - d_j > 0$ ou 0 em caso contrário.

Por fim, alguns critérios de otimização comuns são:

- $C_{\max} = \max\left(C_1, \ldots C_n\right)$: tempo de finalização máxima ou *makespan*. Indica a duração total do programa de produção.
- $F_{\max} = \max\left(F_1, \ldots F_n\right)$: tempo de fluxo máximo. Note que quando $r_j = 0$, $j = 1, \ldots, n$, então $C_{\max} = F_{\max}$.
- $L_{\max} = \max\left(L_1 \ldots L_n\right)$: denominado *Lateness* máximo. Representa a maior violação das datas de entrega.
- $T_{\max} = \max\left(T_1, \ldots, T_n\right)$: define o atraso máximo incorrido.
- $E_{\max} = \max\left(E_1, \ldots, E_n\right)$: indica o adiantamento máximo incorrido.

- $\bar{C} = \sum_{j=1}^{n} C_j$ : indica o tempo de finalização total ou médio.
- $\bar{C}^{\omega} = \sum_{j=1}^{n} \omega_j C_j$ : indica o tempo de finalização total ponderado ou médio.
- $\bar{F} = \sum_{j=1}^{n} F_j$ : representa o tempo de fluxo total ou médio.
- $\bar{F}^{\omega} = \sum_{j=1}^{n} \omega_j F_j$ : representa o tempo de fluxo ponderado total ou médio.
- $\bar{T} = \sum_{j=1}^{n} T_j$ : representa o atraso total ou médio gerado pelo programa de produção.
- $\bar{T}^{\omega} = \sum_{j=1}^{n} \omega_j T_j$ : indica o atraso ponderado total ou médio do programa de produção.
- $\bar{U} = \sum_{j=1}^{n} U_j$ : número total ou médio de tarefas atrasadas.
- $\bar{U}^{\omega} = \sum_{j=1}^{n} \omega_j U_j$ : número ponderado total ou médio de tarefas atrasadas.
- $\bar{E} = \sum_{j=1}^{n} E_j$ : representa o adiantamento total ou médio do programa de produção.
- $\bar{E}^{\omega} = \sum_{j=1}^{n} \omega_j E_j$ : representa o adiantamento ponderado total ou médio do programa de produção.

Note que alguns critérios de otimização são equivalentes, isto é, a minimização de um ou outro conduz à mesma solução ótima, inclusive se o valor do critério não é o mesmo nos dois casos (T´KINDT; BILLAUT, 2006). Em decorrência, tais critérios de otimização são comumente estudados de forma intercambiável. Exemplos destes critérios são o tempo de finalização total e o tempo de finalização médio, tempo de fluxo total e tempo de fluxo médio, atraso total e atraso médio, entre outros. As respectivas versões ponderadas também são equivalentes.

Funções objetivo envolvendo tempos de finalização das tarefas, $C_j$, estão preocupadas com a utilização da capacidade, enquanto funções objetivo considerando o tempo de fluxo, $F_j$, estão relacionadas à minimização de custos de estocagem de produto em processo. Por outro lado, funções relacionadas com datas de entrega, como $T_j$, $L_j$ e $U_j$, penalizam falhas para atender a demanda dos clientes oportunamente. Por fim, funções relacionadas com o adiantamento das tarefas, $E_j$, estão associadas com custos de estocagem de produtos finalizados.

Assim como no caso das restrições, estes são somente alguns dos critérios de otimização mais comuns e, portanto não representam o universo dos possíveis critérios que podem ser considerados em problemas de programação da produção.

Uma vez introduzida a notação seguida na literatura da programação da produção, apresentamos diversas pesquisas envolvendo sistemas de produção do tipo HFS. Esta revisão segue uma estrutura similar à proposta por Ribas, Leisten e Framiñan (2010), na qual os artigos são classificados de acordo às restrições de processamento e características das máquinas. Especificamente, os artigos são classificados segundo a consideração de *buffers* limitados ou zero, máquinas paralelas não relacionadas e tempos de preparação dependentes da sequência.

Esta classificação permite implicitamente a revisão de outras restrições consideradas nesta dissertação, como elegibilidade de máquinas, tempos de liberação das máquinas e tempos de transporte, as quais são escassamente consideradas na literatura. Todas estas características e restrições são consideradas em decorrência de recentes pedidos da comunidade acadêmica para começar o estudo de problemas mais realistas, com o objetivo de diminuir o *gap* existente entre teoria e prática da programação da produção (RIBAS; LEISTEN; FRAMIÑAN, 2010; RUIZ; VÁZQUEZ-RODRÍGUEZ, 2010).

Note que um artigo pode estudar simultaneamente dois ou mais das restrições anteriores e, portanto pode ser classificado em vários tópicos. Quando isto acontecer o artigo será descrito somente em um dos tópicos, enquanto nos demais somente será mencionado sem detalhes. O método de solução utilizado não será abordado, toda vez que nosso foco é a apresentação dos artigos do ponto de vista da configuração produtiva abordada neles. No entanto, quando for estritamente requerido o método de solução será mencionado sem maiores detalhes de implementação ou resultados computacionais.

A revisão de literatura está estruturada da seguinte forma: a seção 2.1 descreve brevemente algumas das revisões de literatura sobre HFS apresentadas recentemente. A seção 2.2 apresenta artigos envolvendo *buffers* limitados ou zero. Na seção 2.3 são apresentadas pesquisas considerando máquinas paralelas não relacionadas e, finalmente, a seção 2.4 apresenta artigos com tempos de preparação dependentes da sequência.

## 2.1. Revisões e classificações recentes

Em anos recentes varias revisões de literatura e classificações de problemas de programação da produção em sistemas de produção HFS têm sido publicadas. Por exemplo, Wang (2005) faz uma revisão de literatura e classifica as abordagens de solução como ótimas, heurísticas e de inteligência artificial.

Kis e Pesch (2005) realizam uma exaustiva revisão de métodos de solução exatos para $HF_m, \left( \left( PM^{(i)} \right)_{i=1}^{m} \right) \mid \mid C_{\max}$ e $HF_m, \left( \left( PM^{(i)} \right)_{i=1}^{m} \right) \mid \mid \bar{F}$. Diferentes limitantes inferiores e estratégias de ramificação são apresentados e discutidos.

Quadt e Kuhn (2007) apresentam uma classificação dos procedimentos de solução para HFS com $m$ estações de processamento, focando principalmente em métodos heurísticos. Estes métodos heurísticos são divididos em abordagens holísticas, as quais consideram o problema de forma integrada, e abordagens de decomposição, as quais dividem o problema de acordo as estações de processamento, as tarefa individuais ou os subproblemas a serem resolvidos.

As mais recentes revisões de literatura sobre programação da produção em HFS foram publicadas por Ruiz e Vázquez-Rodríguez (2010) e Ribas, Leisten e Framiñan (2010). A revisão de Ruiz e Vázquez-Rodríguez (2010) descreve o problema HFS e classifica as diferentes abordagens de solução que têm sido propostas para sua solução, incluindo métodos exatos, heurísticos e meta-heurísticos. Esta revisão não estuda uma variante particular do HFS, mas faz uma classificação ampla e geral do problema, estudando de forma mais detalhada o problema mais comum, $HF_m, \left( \left( PM^{(i)} \right)_{i=1}^{m} \right) \mid \mid C_{\max}$.

Por outro lado, a revisão de Ribas, Leisten e Framiñan (2010) classifica artigos publicados depois de 1.995 utilizando duas abordagens diferentes. Na primeira delas, os artigos são classificados do ponto de vista da produção, isto e, considerando as diferentes restrições de processamento, características das máquinas e critérios de otimização. Na segunda abordagem, os artigos são classificados segundo o método de solução proposto para resolver o problema. Assim, os artigos são classificados em procedimentos exatos, heurísticos, híbridos e simulação/sistemas de suporte à decisão.

## 2.2. *Buffer* limitado ou zero

Problemas de programação da produção com *buffers* limitados e *buffers* zero podem ser encontrados em vários ambientes de manufatura moderna, tais como

sistemas de produção *Just-in-time* e linhas de montagem flexíveis (SAWIK, 2000). Este fenômeno também é comum nas indústrias químicas e de fabricação de aço, onde há uma limitação no *buffer* disponível para a armazenagem de produto em processo devido a requerimentos de espaço físico e investimento (LIU; KARIMI, 2008; TANG; XUAN, 2005). No entanto, e apesar de ser comum na indústria, a literatura do HFS não apresenta muitas pesquisas nas quais restrições de *buffers* limitados ou zero sejam consideradas. A seguir, apresentam-se brevemente algumas delas.

O problema de programação da produção em HFS com *buffer* limitados com máquinas paralelas idênticas, visando minimizar o *makespan* foi abordado por Wardono e Fathi (2004). Alguns autores adotam a estratégia de transformar o problema de *buffers* limitados em um problema com *buffers* zero, tal como feito por Thornton e Hunsucker (2004) e Wang e Tang (2009). Sawik (2000) estuda a versão flexível com *buffers* zero através de modelos de programação inteira mista, apresentando alguns exemplos numéricos. Algumas extensões para considerar restrições de recirculação e rotas alternativas de processamento também são propostas.

Posteriormente, novas extensões para considerar processamento em batelada são feitas por Sawik (2002). Tavakkoli-Moghaddam, Safaei e Sassani (2009) também estudam a versão flexível do problema e utilizam o limitante inferior proposto por Sawik (2000) para analisar a efetividade do método de solução proposto. Critérios de otimização diferentes do *makespan*, como tempo de fluxo ponderado total e atraso ponderado total, são considerados por Tran e Ng (2011), enquanto a minimização do tempo de finalização ponderado total é estudada por Tang e Xuan (2005).

Alguns problemas mais complexos também são estudados na literatura. Por exemplo, a pesquisa de Kochhar e Morris (1987) considera *buffers* limitados, máquinas idênticas, tempos de preparação não dependentes da sequência, ociosidade e tempos de indisponibilidade das máquinas por avaria e manutenção. O objetivo é a minimização dos custos de produção associados ao tempo de fluxo médio. Kochhar, Morris e Wong (1988) estudam o mesmo problema e propõem vários operadores de busca local para resolvê-lo. Liu e Karimi (2008) propõem vários modelos de programação inteira mista para programar a produção de HFFL com *buffers* zero e restrições de processamento em bateladas, elegibilidade de máquinas, tempos de liberação das bateladas e tempos de liberação das máquinas. Os modelos são estendidos para considerar máquinas paralelas não relacionadas e critérios de otimização diferentes do *makespan*, como atraso mínimo, adiantamento mínimo e soma ponderada de atraso e adiantamento. O caso especial sem

tempos de espera, no qual o processamento das bateladas deve ser continuo, também é abordado.

Um HFS com *buffer* zero, máquinas paralelas não relacionadas e tempos de preparação dependentes da sequência é estudado por Rashidi, Jahandar e Zandieh (2010). O problema, inicialmente de múltiplos objetivos, é transformado em um problema com um único objetivo usando ponderações que refletem a importância do cada objetivo individual. Os objetivos a serem minimizados são o *makespan* e o atraso máximo, respectivamente. Um problema similar é estudado por Hakimzadeh Abyaneh e Zandieh (2012). Desta vez as máquinas paralelas são idênticas e como critério de otimização são considerados o *makespan* e o atraso total.

Algumas aplicações industriais são reportadas. Por exemplo, Deal, Yang e Hallquist (1994) estudam o problema de programação da produção em uma indústria petroquímica formada por duas estações e *buffers* limitados, visando à minimização do *makespan.* Neste sistema, a matéria prima entra de forma continua na primeira estação, onde é processada pelas diferentes máquinas paralelas disponíveis. O produto em processo resultante é armazenado em tanques de capacidade finita alocados entre as duas estações, onde deve permanecer até que seu processamento na segunda estação possa começar. Finalmente, na segunda estação o produto em processo é mais processado ou misturado com outros produtos, obtendo-se os diferentes produtos finais prontos para a entrega ao cliente final.

Yaurima, Burtseva e Tchernykh (2009) consideram uma linha de montagem de televisores com três seções bem definidas: inserção automática, montagem manual e controle de qualidade, embalagem e expedição. A pesquisa é limitada à seção de inserção automática, a qual representa um HFS de seis estações. Cada estação consiste de várias máquinas de inserção em paralelo, as quais são dedicadas ao processamento de certos tipos de placa de circuito impresso (PCB, *Printed Circuit Board*). As máquinas em cada estação são de diferentes marcas, capacidades e funcionalidades. Ajustes nas máquinas são requeridos quando o tipo de PCB é trocado e o tempo requerido para o ajuste depende essencialmente do tipo de PCB previamente processado na máquina. Finalmente, cada máquina tem um *buffer* de capacidade limitada para a armazenagem de produto em processo. Dadas todas estas características, o problema estudado é definido como um HFS com máquinas paralelas não relacionadas, elegibilidade de máquinas, tempos de preparação dependentes da sequência e *buffers* limitados. O critério de otimização adotado é o *makespan.*

## 2.3. Máquinas paralelas não relacionadas

A coexistência de várias máquinas paralelas em uma mesma estação se deve, entre outras razões, pela necessidade de incrementar a capacidade do chão de fábrica, nivelar a capacidade entre estações e, inclusive, dedicar algumas destas máquinas à fabricação de produtos especiais ou customizados (RIBAS; LEISTEN; FRAMIÑAN, 2010). Em situações reais é comum encontrar máquinas mais novas e modernas funcionando ao lado de máquinas mais antigas e menos eficientes, as quais poderiam executar as mesmas operações que às mais novas, mas geralmente precisando de um maior tempo de processamento (JUNGWATTANAKIT et al., 2009). Apesar de esta situação ser muito comum na indústria, a maioria da literatura reportada tem-se focado em problemas com máquinas idênticas em cada estação. A seguir serão apresentados alguns artigos reportados na literatura do HFS que consideram máquinas paralelas não relacionadas.

Oğuz, Lin e Cheng (1997) estudam um caso particular do HFS com duas estações. A primeira estação tem duas máquinas paralelas não relacionadas, enquanto a segunda estação tem somente uma máquina. Adicionalmente, há dois tipos de tarefas e cada máquina na primeira estação pode processar somente um tipo de tarefa. O critério de otimização adoptado é a minimização do *makespan.* Uma generalização deste problema é estudada por Low, Hsu e Su (2008). Neste caso o HFS tem um número qualquer de máquinas paralelas não relacionadas na primeira estação e uma única máquina na segunda. Restrições de elegibilidade de máquinas são consideradas, pois as máquinas na primeira estação diferem em funcionalidade, pelo qual nem todas as tarefas podem ser processadas por todas as máquinas. O HFS com máquinas paralelas não relacionadas e critério de otimização *makespan*, denotado como $HF_m, \left( \left( RM^{(i)} \right)_{i=1}^{m} \right) \mid \mid C_{\max}$, foi estudado por Chen e Chen (2009). Note que neste caso há $m > 2$ estações.

Algumas pesquisas que consideram HFS com máquinas paralelas não relacionadas em conjunto com algumas restrições adicionais também são reportadas. Por exemplo, Low (2005) estuda um HFS com máquinas paralelas não relacionadas, tempos de preparação independentes da sequência e tempos de remoção dependente da sequência, visando minimizar o tempo de fluxo total. Ruiz e Maroto (2006) estudam um problema inspirado na indústria de produção de revestimentos cerâmicos, com máquinas paralelas não relacionadas, tempos de preparação dependentes da sequência e

elegibilidade máquinas, com o objetivo de minimizar o *makespan.* Uma generalização deste problema, considerando flexibilidade e tempos de liberação das máquinas, é estudada por Zandieh, Mozaffari e Gholami (2009). O HFS com máquinas paralelas não relacionadas, tempos de espera limitado e tempos de liberação das tarefas, visando minimizar o *makespan* foi estudado recentemente por Attar, Mohammadi e Tavakkoli-Moghaddam (2013).

Jungwattanakit et al. (2008) propõem um modelo de programação inteira mista e vários algoritmos heurísticos para um HFS com máquinas paralelas não relacionadas, tempos de preparação dependentes da sequência e tempos de liberação das máquinas, visando minimizar a soma convexa do *makespan* e o número de tarefas atrasadas. Este mesmo problema é novamente estudado por Jungwattanakit et al. (2009), desta vez propondo métodos meta-heurísticos para resolvê-lo. Behnamian e Fatemi Ghomi (2011) consideram o caso com máquinas paralelas não relacionadas e tempos de preparação dependentes da sequência. Adicionalmente, o tempo de processamento das tarefas pode ser reduzido alocando a cada máquina uma quantidade adicional de recursos com um custo associado. O objetivo é a minimização dos custos associados ao *makespan* e a alocação de recursos adicionais.

Ruiz, Şerifoğlu e Urlings (2008) apresentam e analisam através de avançadas técnicas estatísticas um complexo modelo de programação inteira mista para resolver um HFS flexível com máquinas paralelas não relacionadas, tempos de preparação dependentes da sequência, elegibilidade de máquinas, restrições de precedência, tempos de liberação das máquinas e tempos de transferência entre estações, visando minimizar o *makespan.* Posteriormente, devido à dificuldade de resolver instâncias de maior tamanho através deste modelo, vários métodos meta-heurísticos foram propostos por Urlings, Ruiz e Şerifoğlu (2010) e Urlings, Ruiz e Stützle (2010).

Outras pesquisas envolvendo máquinas paralelas não relacionadas, descritas em seções anteriores, são aquelas feitas por Hakimzadeh Abyaneh e Zandieh (2012); Liu e Karimi (2008); Rashidi, Jahandar e Zandieh (2010) e Yaurima, Burtseva e Tchernykh (2009).

## 2.4. Tempos de preparação dependentes da sequência

Em termos gerais, o tempo de preparação é o tempo requerido para organizar os recursos necessários para executar uma determinada tarefa. Estas

atividades de preparação podem incluir, por exemplo, a obtenção de ferramentas, posicionamento do material em processo, retorno de ferramentas, limpeza, ajuste de peças e acessórios necessários, ajuste de ferramentas e inspeção do material (ALLAHVERDI; GUPTA; ALDOWAISAN, 1999).

Segundo Allahverdi e Soroush (2008), a programação da produção com tempos ou custos de preparação tem um papel importante nos ambiente de manufatura e serviços atuais, onde produtos e serviços confiáveis devem ser entregues pontualmente e o uso eficiente dos recursos disponíveis é indispensável. Os mesmos autores indicam os diversos benefícios que podem ser atingidos através da redução de custos e tempos de preparação, dentre os quais se destacam o incremento da velocidade de produção, redução de tempos de entrega, redução de despesas, trocas mais rápidas, diminuição de tamanhos de lote mínimo, redução de estoques, entre outros.

Desde a publicação do trabalho de Allahverdi, Gupta e Aldowaisan (1999) a literatura da programação da produção tem apresentado um incremento notório de pesquisas envolvendo explicitamente tempos ou custos de preparação, com uma média de 40 artigos publicados por ano (ALLAHVERDI et al., 2008). Nesta seção apresentamos uma revisão de artigos que estudam problemas de programação da produção em HFS envolvendo explicitamente tempos de preparação dependentes da sequência. Nosso interesse é também diferenciar entre tempos de preparação antecipatórios e não antecipatórios, portanto quando possível definiremos explicitamente se o tempo de preparação considerado no artigo é antecipatório ou não.

Recentemente, Gómez-Gasquet, Andrés e Lario (2012) estudaram o problema com tempos de preparação dependentes da sequência, máquinas paralelas idênticas e *makespan* como critério de otimização. O HFS com máquinas paralelas idênticas e tempos de preparação não antecipatórios dependentes da sequência é estudado por Naderi, Zandieh e Roshanaei (2008). Neste trabalho os autores consideram duas funções objetivo diferentes: minimização do *makespan* e minimização do atraso máximo. Este mesmo problema foi também estudado por Naderi et al. (2009), considerando adicionalmente tempo de transporte entre estações, de modo a minimizar o tempo de finalização total e o atraso total. Outra variante, na qual os tempos de preparação são antecipatórios, é estudada por Naderi, Zandieh e Shirazi (2009). Neste caso, o critério de otimização é somente a minimização do atraso total ponderado.

O HFS flexível com máquinas paralelas idênticas e tempos de preparação não antecipatórios dependentes da sequência, visando à minimização do *makespan* tem

sido estudado largamente por Behnamian, Fatemi Ghomi e Zandieh (2012); Kurz e Askin (2003,2004); Naderi, Ruiz e Zandieh (2010) e Zandieh, Fatemi Ghomi e Moattar Husseini (2006).

Algumas aplicações industriais envolvendo tempos de preparação dependentes da sequência também são reportadas na literatura. Por exemplo, Liu e Chang (2000) propõem um modelo de programação inteira mista e um método de solução baseado em relaxação lagrangeana para o HFS com máquinas idênticas e tempos de preparação dependentes da sequência, visando minimizar os custos totais de produção, os quais são representados por custos de atraso, estoque em processo e preparação das máquinas. O método proposto é utilizado para programar as máquinas do processo de teste de circuitos integrados, os quais devem ser examinados depois da sua fabricação para garantir sua conformidade com especificações operacionais.

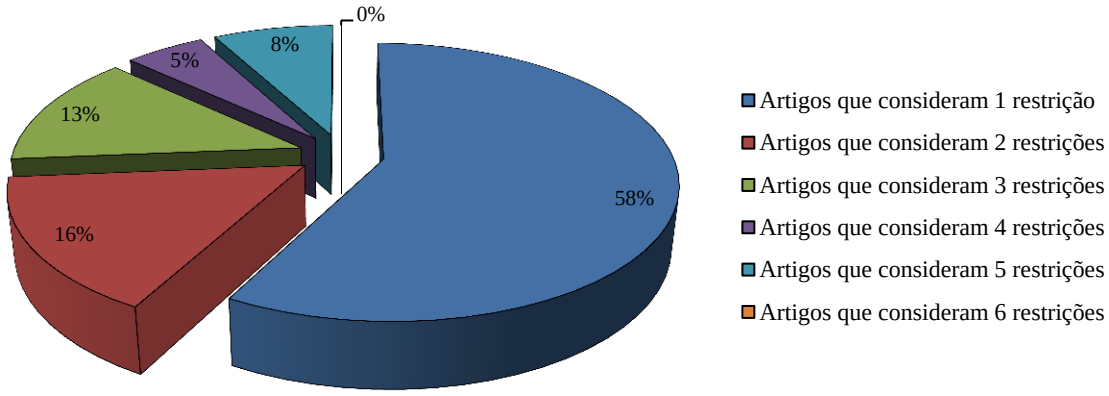
Um estudo de caso na indústria de manufatura de etiquetas autoadesivas é apresentado por Lin e Liao (2003). O sistema estudado é modelado como um HFS de duas estações com tempos de preparação dependentes da sequência e elegibilidade de máquinas. As atividades de preparação ocorrem na primeira estação, onde somente há uma máquina disponível para o processamento das tarefas. Estas atividades são basicamente a troca da base adesiva utilizada para produzir as etiquetas e o ajuste da temperatura de operação da máquina. Na segunda estação há dois tipos de máquinas de corte, cada um consistindo em duas máquinas paralelas idênticas. Dependendo dos requerimentos do cliente, cada tarefa pode ser processada somente em um tipo de máquina de corte. O objetivo é programar a produção das etiquetas, minimizando o atraso máximo ponderado.

Outros artigos incluindo tempos de preparação dependentes da sequência, já descritos em seções anteriores, são aqueles apresentados por Behnamian e Fatemi Ghomi (2011); Jungwattanakit et al. (2008, 2009); Kochhar, Morris e Wong (1988); Kochhar e Morris (1987); Rashidi, Jahandar e Zandieh (2010); Ruiz e Maroto (2006); Yaurima, Burtseva e Tchernykh (2009) e Zandieh, Mozaffari e Gholami (2009). Adicionalmente, pesquisas que explicitamente consideram tempos de preparação dependentes da sequência tanto antecipatórios quanto não antecipatórios são aquelas apresentadas por Ruiz, Şerifoğlu e Urlings (2008); Urlings, Ruiz e Şerifoğlu (2010) e Urlings, Ruiz e Stützle (2010).

A anterior revisão da literatura evidencia e confirma algumas das conclusões expostas recentemente por Ribas, Leisten e Framiñan (2010):

1) A maioria das pesquisas em HFS consideram máquinas paralelas idênticas em cada estação. Isto é provavelmente devido ao fato de que problemas com máquinas idênticas são mais fáceis de tratar que problemas com máquinas uniformes e não relacionadas. No entanto, pesquisas envolvendo máquinas paralelas uniformes e não relacionadas são requeridas, pois situações reais são melhor representadas através deste tipo de máquinas.
2) Há relativamente poucas pesquisas que consideram restrições como tempos de preparação dependentes da sequência, bloqueio, elegibilidade de máquinas, tempos de liberação das máquinas ou tempos de transporte. Adicionalmente, estas restrições geralmente são estudadas uma por vez, havendo poucos artigos nos quais todas, ou algumas delas, sejam estudadas conjuntamente. A Figura 2.1 apresenta a percentagem de artigos revisados que estudam um determinado número de restrições de forma conjunta e permite confirmar a conclusão exposta anteriormente. Por exemplo, 58% dos artigos revisados estudam somente uma restrição por vez. Entretanto, os artigos que consideram duas restrições por vez representam somente 16%. Portanto, é evidente a escassez de pesquisas envolvendo várias restrições conjuntamente e visando resolver problemas mais próximos da realidade industrial.

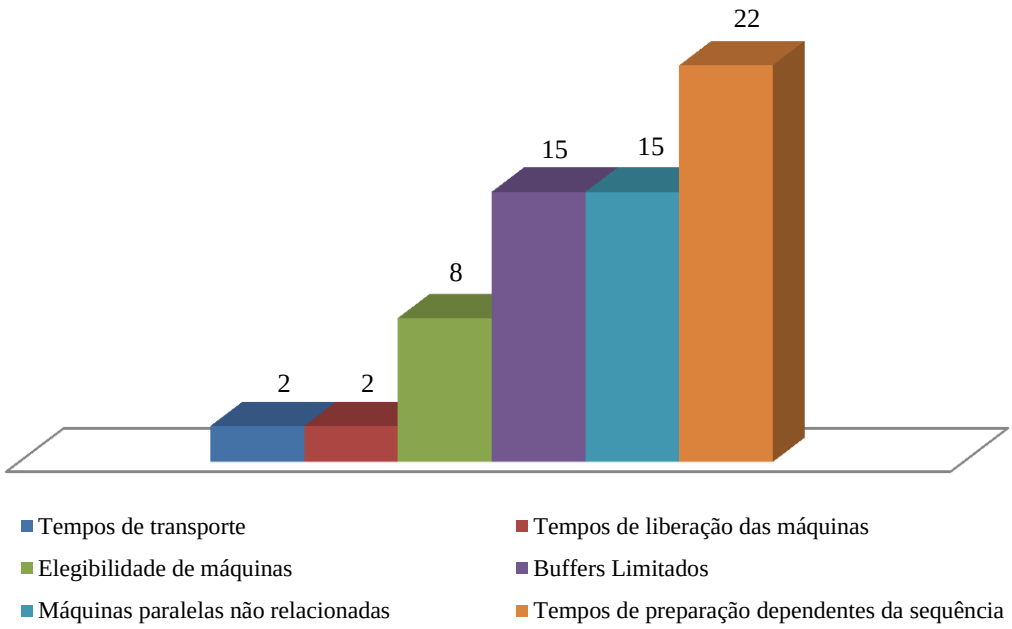
**Figura 2.1 – Percentagem de artigos revisados que consideram várias restrições simultaneamente.**




3) Há poucos artigos que consideram explicitamente tempos de preparação dependentes da sequência antecipatórios ou não antecipatórios. A maioria dos artigos revisados não define claramente se os tempos de preparação considerados pertencem a um tipo ou outro. Neste aspecto, a literatura sugere que quando não houver declaração explícita do tipo de preparação estudada, esta deve ser considerada como antecipatória. Em qualquer caso, a revisão feita indica que pesquisas envolvendo ambos os tipos preparação são desejáveis. A Figura 2.2

mostra o total de artigos que consideram uma determinada restrição. Note que neste caso os artigos que consideram mais de uma restrição são contabilizados várias vezes e, portanto a soma de cada categoria não é igual ao número de artigos revisados. Evidentemente, restrições de tempos de preparação dependentes da sequência são mais estudadas que qualquer outra restrição, o qual confirma o crescente interesse da comunidade acadêmica por estudar esta restrição. Analisando o tipo de preparação considerada, encontramos que 12 artigos consideram tempos de preparação antecipatória, 7 consideram tempos de preparação não antecipatória e somente 3 artigos consideram ambos os tipos de forma conjunta.

**Figura 2.2 – Número total de artigos revisados que consideram cada restrição**




Todas estas conclusões motivam o estudo de um HFS complexo, que considere conjuntamente várias restrições realistas, visando diminuir o *gap* entre a teoria e prática da programação da produção neste tipo de sistema. Portanto, nesta dissertação será estudado um HFS com máquinas paralelas não relacionadas, *buffers* limitados, tempos de preparação dependentes da sequência e da máquina, tanto antecipatórios quanto não antecipatórios, elegibilidade de máquinas, tempos de transporte e tempos de liberação das máquinas. Como critério de otimização será considerado o *makespan*, cuja minimização está diretamente relacionada com a utilização eficiente dos recursos de produção. Este problema será formalmente definido na seção 3.2.

Um problema com todas estas restrições ainda não tem sido considerado na literatura. Assim, este estudo começa com sua modelagem como um problema de programação inteira mista e, em seguida, propõem-se métodos de solução capazes de obter soluções de boa qualidade em curtos tempos computacionais.

Para finalizar, apresenta-se a Tabela 2.1, na qual são resumidos todos os artigos revisados. A primeira coluna mostra as referências bibliográficas em ordem cronológica, a segunda coluna apresenta a descrição do problema segundo a notação $\alpha|\beta|\gamma$ e, por fim, a última coluna indica a abordagem utilizada para resolver o problema.

**Tabela 2.1 – Resumo da revisão da literatura**

| Referência | Descrição do problema | Abordagem de solução |
|---|---|---|
| (KOCHHAR; MORRIS, 1987) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid block, brkdwn, skip \mid \sum F_j$ | Regras de liberação |
| (KOCHHAR; MORRIS; WONG, 1988) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid block, brkdwn \mid \sum F_j$ | Regras de liberação |
| (DEAL; YANG; HALLQUIST, 1994) | $FH2, \left(\left(PM^{(1)}, PM^{(2)}\right)\right) \mid buffer \mid C_{\max}$ | Heurísticas |
| (SAWIK, 2000) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid skip, block, recrc \mid C_{\max}$ | MIP |
| (LIU; CHANG, 2000) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid S_{jk}, block \mid \sum \alpha_j E_j + \omega_j T_j$ | Heurísticas baseadas em relaxação lagrangeana |
| (SAWIK, 2002) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid batch, skip, block, recrc \mid C_{\max}$ | MIP |
| (KURZ; ASKIN, 2003) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid S_{jk}, skip \mid C_{\max}$ | Heurísticas |
| (LIN; LIAO, 2003) | $FH2, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \mid S_{jk}^{(1)}, M_j^{(2)} \mid \omega T_{\max}$ | Heurísticas |
| (THORNTON; HUNSUCKER, 2004) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid block \mid C_{\max}$ | Heurísticas |
| (WARDONO; FATHI, 2004) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid buffer \mid C_{\max}$ | Meta-heurísticas |
| (KURZ; ASKIN, 2004) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid S_{jk}, skip \mid C_{\max}$ | MIP, heurísticas, meta-heurísticas |
| (LOW, 2005) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \mid R_{jk} \mid \sum F_j$ | Meta-heurísticas |
| (TANG; XUAN, 2005) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid buffers \mid \sum \omega_j C_j$ | Heurísticas baseadas em relaxação lagrangeana |
| (ZANDIEH; FATEMI GHOMI; MOATTAR HUSSEINI, 2006) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \mid S_{jk}, skip \mid C_{\max}$ | Meta-heurísticas |
| (RUIZ; MAROTO, 2006) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \mid S_{jk}, M_j \mid C_{\max}$ | Meta-heurísticas |

| | | |
|---|---|---|
| (RUIZ; ŞERIFOĞLU; URLINGS, 2008) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| skip, rm, lag, S_{ijk}, M_j, prec \right\| C_{\max}$ | MIP, heurísticas |
| (LOW; HSU; SU, 2008) | $FH2, \left(\left(RM^{(1)}, \varnothing^{(2)}\right)\right) \left\| M_j \right\| C_{\max}$ | Meta-heurísticas |
| (JUNGWATTANAKIT et al., 2008) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk}, r_j \right\| \alpha \cdot C_{\max} + (1-\alpha) \cdot \sum U_j$ | MIP, heurísticas |
| (LIU; KARIMI, 2008) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| block, skip, batch, M_j, r_j, rm \right\| vários$ | MIP |
| (NADERI; ZANDIEH; ROSHANAEI, 2008) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk} \right\| \{C_{\max}, T_{\max}\}$ | Meta-heurísticas |
| (CHEN; CHEN, 2009) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\|  \right\| C_{\max}$ | Heurísticas |
| (JUNGWATTANAKIT et al., 2009) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk}, r_j \right\| \alpha \cdot C_{\max} + (1-\alpha) \cdot \sum U_j$ | MIP, heurísticas, meta-heurísticas |
| (NADERI et al., 2009) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk}, transport \right\| \{\sum C_j, \sum T_j\}$ | Meta-heurísticas |
| (NADERI; ZANDIEH; SHIRAZI, 2009) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk}, transport \right\| \sum \omega_j T_j$ | MIP, meta-heurísticas |
| (WANG; TANG, 2009) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| buffers \right\| \sum \omega_j C_j$ | Meta-heurísticas |
| (TAVAKKOLI-MOGHADDAM; SAFAEI; SASSANI, 2009) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| skip, block, recrc \right\| C_{\max}$ | Meta-heurísticas |
| (YAURIMA; BURTSEVA; TCHERNYKH, 2009) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk}, M_j, buffers \right\| C_{\max}$ | Meta-heurísticas |
| (ZANDIEH; MOZAFFARI; GHOLAMI, 2009) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| skip, S_{jk}, M_j, rm \right\| C_{\max}$ | Meta-heurísticas |
| (URLINGS; RUIZ; ŞERIFOĞLU, 2010) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| skip, rm, lag, S_{ijk}, M_j, prec \right\| C_{\max}$ | Meta-heurísticas |
| (URLINGS; RUIZ; STÜTZLE, 2010) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| skip, rm, lag, S_{ijk}, M_j, prec \right\| C_{\max}$ | Meta-heurísticas |
| (RASHIDI; JAHANDAR; ZANDIEH, 2010) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| block, S_{jk} \right\| \{C_{\max}, T_{\max}\}$ | Meta-heurísticas |
| (NADERI; RUIZ; ZANDIEH, 2010) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk}, skip \right\| C_{\max}$ | Meta-heurísticas |
| (TRAN; NG, 2011) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| block, skip \right\| \{\sum \omega_j F_j, \sum \omega_j T_j,\}$ | Meta-heurísticas |
| (BEHNAMIAN; FATEMI GHOMI, 2011) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk} \right\| CustoTotal$ | Meta-heurísticas |
| (HAKIMZADEH ABYANEH; ZANDIEH, 2012) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| block, S_{jk} \right\| \{C_{\max}, \sum T_j\}$ | Meta-heurísticas |
| (GÓMEZ-GASQUET; ANDRÉS; LARIO, 2012) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk} \right\| C_{\max}$ | Meta-heurísticas |
| (BEHNAMIAN; FATEMI GHOMI; ZANDIEH, 2012) | $FH_m, \left(\left(PM^{(i)}\right)_{i=1}^{m}\right) \left\| S_{jk}, skip \right\| C_{\max}$ | Meta-heurísticas |
| (ATTAR; MOHAMMADI; TAVAKKOLI-MOGHADDAM, 2013) | $FH_m, \left(\left(RM^{(i)}\right)_{i=1}^{m}\right) \left\| r_j, skip \right\| \{C_{\max}, \sum T_j\}$ | Meta-heurísticas |

# 3. Descrição do Problema

Este capítulo apresenta uma descrição do HFS estudado nesta dissertação. A fim de contextualizar o problema, inicialmente uma descrição do HFS na forma padrão é apresentada na seção 3.1. A seção 3.2 apresenta a definição formal do HFS considerado nesta pesquisa, detalhando as principais características e restrições de processamento a serem levadas em conta. Finalmente, na seção 3.3 é apresentado um exemplo ilustrativo do problema estudado.

## 3.1. *Flowshop* Híbrido (HFS)

Como descrito no capítulo 2, em um ambiente *Flowshop*, um conjunto de tarefas deve ser processado através de múltiplas estações na mesma ordem, desde a primeira até a última estação, e cada estação tem unicamente uma máquina. Porém, como destacado por Ribas, Leisten e Framiñan (2010), em algumas indústrias a necessidade de incrementar ou equilibrar a capacidade das estações tem levado à duplicação de máquinas. Em outras, a crescente demanda de produtos customizados tem desencadeado a necessidade de comprar máquinas adicionais para algumas estações com o objetivo de dedicá-las à produção destes produtos. Em qualquer caso, a aquisição das novas máquinas não implica a substituição ou eliminação do equipamento existente, o que leva à coexistência de várias máquinas, possivelmente diferentes, em várias estações do processo de produção.

Do ponto de vista teórico, esta nova configuração de produção, conhecida como *Flowshop* Híbrido (HFS), pode ser considerada como a combinação de dois problemas particulares de programação da produção: o problema de máquinas paralelas e o problema do *Flowshop*. No problema de máquinas paralelas a decisão-chave é a alocação de tarefas a máquinas, enquanto no *Flowshop* a decisão-chave é o sequenciamento das tarefas através do chão de fábrica. Portanto, no HFS as decisões-chave são designar e programar as tarefas às máquinas em cada estação, isto é, determinar a ordem na qual as tarefas serão processadas nas diferentes máquinas de cada estação, com o objetivo de minimizar um ou vários critérios determinados.

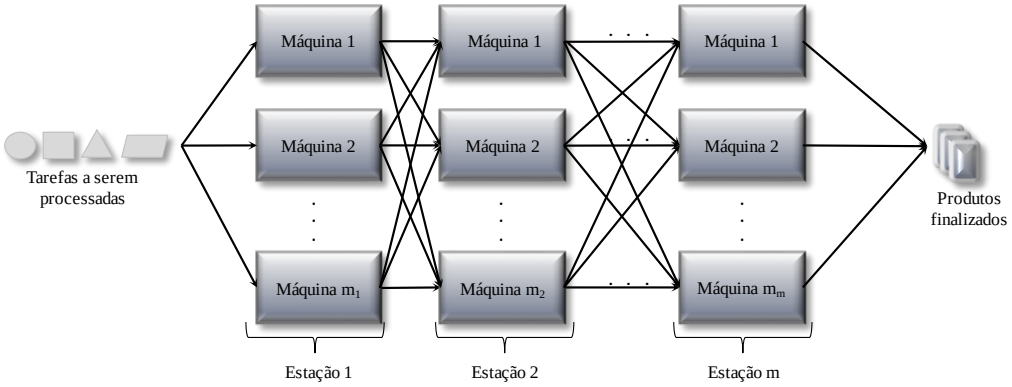
Na forma padrão, este sistema de produção consiste de $m$ estações de máquinas, tal que cada estação $i$ tem $m_i > 1$ máquinas paralelas idênticas. O conjunto de estações é representado por $M = \{1,\ldots,m\}$ e o conjunto de máquinas na estação $i$ é denotado por $M_i = \{1,\ldots,m_i\}$, $i \in M$. Um conjunto de $n$ tarefas, representado por

$N = \{1, \ldots, n\}$, deve ser processado no sistema. Todas as tarefas passam pelas estações na mesma ordem $1, 2, \ldots, m$. A tarefa $j$, $j \in N$, consiste de uma sequência de $m$ operações, cada uma delas correspondendo ao seu processamento na estação $i$ durante um tempo de processamento denotado por $p_{ij}$. Em cada estação $i$, a tarefa $j$ pode ser processada em qualquer das $m_i$ máquinas. Cada máquina pode processar, no máximo, uma tarefa por vez, e cada tarefa é processada, no máximo, em uma máquina por vez. As tarefas estão disponíveis para processamento no tempo zero, os tempos de preparação são insignificantes, interrupções não são permitidas, a capacidade de armazenagem entre estações é considerada ilimitada e os dados do problema são determinísticos e conhecidos antecipadamente. O problema é gerar um programa de produção que minimize o *makespan.* A definição anterior é geral, mas representa a base para o estudo de outras variantes do problema, as quais podem ser obtidas acrescentando ou removendo algumas restrições ou suposições.

O estudo deste problema é de grande relevância prática, dado que este tipo de configuração produtiva pode ser encontrado em diversos setores de manufatura, como na indústria petroquímica (DEAL; YANG; HALLQUIST, 1994), manufatura de filmes fotográficos (TSUBONE et al., 1993), indústria automotiva (AGNETIS et al., 1997), indústria eletrônica (JIN et al., 2002; LIU; CHANG, 2000), indústria têxtil (GRABOWSKI; PEMPERA, 2000; KARACAPILIDIS; PAPPIS, 1996), manufatura de etiquetas autoadesivas (LIN; LIAO, 2003), indústria de revestimentos cerâmicos (RUIZ; MAROTO, 2006; RUIZ; ŞERIFOĞLU; URLINGS, 2008), indústria de serviços (BERTEL; BILLAUT, 2004), entre outras.

A Figura 3.1 ilustra esquematicamente a estrutura de um HFS. As tarefas a serem processadas entram ao sistema na estação 1, onde um conjunto de máquinas paralelas está disponível para o processamento. As setas indicam o fluxo unidirecional que devem seguir as tarefas, desde a estação 1 até a estação $m$. Finalmente, depois de passar por todas as estações, os produtos finalizados saem do sistema, prontos para ser entregues ao cliente final. Note que o número de máquinas paralelas não é necessariamente o mesmo para cada estação, isto é, diferentes estações podem ter diferente número de máquinas arranjadas em paralelo.

**Figura 3.1 – Representação HFS**




## 3.2. Definição do problema de estudo

As conclusões expostas no Capítulo 2, assim como recentes revisões de literatura, mostram que o estudo de problemas de programação da produção em HFS está largamente concentrado na forma padrão do problema, e consequentemente a comunidade acadêmica tem começado o estudo de problemas mais realistas, os quais possam representar de forma mais adequada os diversos sistemas de produção encontrados na prática. A seguir são descritas as diferentes características e restrições do HFS estudado.

Considera-se que as máquinas paralelas são não relacionadas, o que significa que o tempo de processamento das tarefas é diferente para cada máquina e cada estação. Esta situação, tal como descrito anteriormente, pode acontecer devido à compra de máquinas mais novas e eficientes, às diferentes atividades de manutenção feitas sobre cada máquina ou, inclusive, ao fato de que certas máquinas são, física e tecnologicamente, mais adequadas para algumas tarefas que outras.

Igualmente, devido a sua importância e relevância prática, tempos de preparação dependentes da sequência são explicitamente considerados. Note que, devido ao fato das máquinas serem não relacionadas, esses tempos de preparação também são dependentes das máquinas. Assim, o tempo de preparação requerido entre qualquer par de tarefas também depende da máquina na qual forem processados. Esses tempos de preparação podem ser tanto antecipatórios quanto não antecipatórios, dado que ambos os tipos são comumente encontrados na prática.

Adicionalmente, devido a restrições tecnológicas e físicas, nem todas as máquinas em cada estação são capazes de processar todas as tarefas. Isto é, para cada tarefa há um conjunto de máquinas elegíveis em cada estação que podem processá-la.

Uma característica importante em ambientes de produção reais é que as máquinas estão processando tarefas prévias no momento em que o novo programa de produção está sendo elaborado. Portanto nenhuma tarefa pode começar seu processamento em uma máquina até que esta termine todas as tarefas pertencentes a programas de produção anteriores. Deste modo, as máquinas não necessariamente estão disponíveis no tempo zero e cada uma delas tem um tempo de liberação, antes do qual nenhuma tarefa do novo programa de produção pode ser processada.

Em geral, problemas de programação da produção consideram espaços de armazenagem intermediária ilimitada. Porém, na prática há limitações de espaço físico e investimento, pelas quais o espaço disponível para armazenagem de produto em processo é considerado limitado ou nulo, dependendo das características do processo de produção. O caso no qual a capacidade dos *buffers* intermediários é limitada será considerado.

Finalmente, tempo de transporte entre estações também é considerado. Isto implica que, uma vez finalizado seu processamento em uma dada estação, a tarefa não está imediatamente disponível para processamento na seguinte estação, pois requer ser transportada de uma estação para outra.

O critério de otimização adotado é a minimização da duração do programa de produção, $C_{\max}$.

Formalmente, um conjunto $N$ de tarefas, $N = \{1,...,n\}$, deve ser processado em um conjunto $M$ de estações, $M = \{1,...,m\}$. Em cada estação $i$, $i \in M$, há um conjunto $M_i = \{1,...,m_i\}$ de processadores paralelos não relacionados. Os processadores podem ser máquinas ou *buffers*, sendo os últimos considerados máquinas especiais com tempos de processamento e preparação iguais a zero. $E_{ij}$ denota o conjunto de processadores capazes de processar a tarefa $j$, $j \in N$, na estação $i$. O parâmetro $rm_{il}$ indica o tempo de liberação do processador $l$, $l \in M_i$, na estação $i$. Cada tarefa $j$ deve ser processada por exatamente um dos processadores paralelos em cada estação $i$. O tempo de processamento da tarefa $j$ no processador $l$ da estação $i$ é denotado como $p_{ilj}$. O parâmetro $S_{iljk}$ denota o tempo de preparação entre as tarefas $j$ e $k$, $k \in N$, no processador $l$ da estação $i$. Adicionalmente, o parâmetro binário $A_{iljk}$ indica se a preparação entre as tarefas $j$ e $k$ no processador $l$ da estação $i$ é antecipatória ou não

antecipatória. Operações de transporte entre estações devem ser feitas antes de iniciar o processamento de qualquer tarefa, assim $t_{iq}$ indica o tempo de transporte entre as estações $i$ e $q, q \in M$. O critério de otimização é o $C_{\max}$, tal que $C_{\max} = \max_{j \in N} \{C_j\}$ e $C_j$ indica o tempo de finalização da tarefa $j$ na última estação.

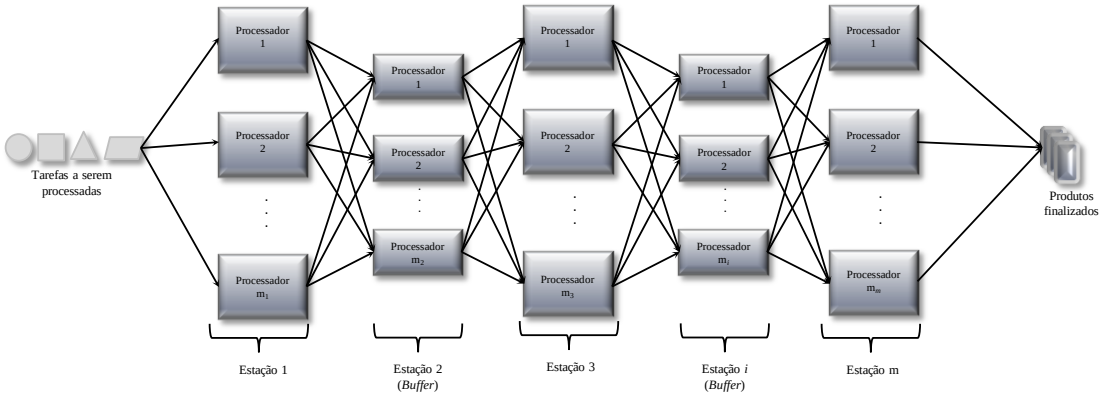
Sem perda de generalidade, assume-se que:

- O tempo de processamento das tarefas nos processadores que não são capazes de processá-la é considerado igual a zero, ou seja, $p_{ilj} = 0$ se $l \notin E_{ij}$, $j \in N$, $i \in M$.
- Em cada estação cada tarefa tem, no mínimo, um processador capaz de processá-la, ou seja, $|E_{ij}| \geq 1$, $j \in N$, $i \in M$.
- O tempo de preparação da primeira tarefa programada em cada processador é denotado como $S_{il0k}$, tal que 0 é uma tarefa fictícia que precede a primeira tarefa de cada processador. Esta estratégia permite considerar a preparação inicial do processador, a qual depende da tarefa a ser processada.
- O tempo de preparação entre as tarefas $j$ e $k$ no processador $l$ da estação $i$ é considerado zero se $l \notin E_{ij} \vee l \notin E_{ik}$. Isto é, uma preparação entre as tarefas $j$ e $k$ no processador $l$ da estação $i$ pode acontecer se e somente se o processador $l$ é capaz de processar tanto à tarefa $j$ quanto à tarefa $k$.

A abordagem unificada de chamar conjuntamente máquinas e *buffers* como processadores foi utilizada, por exemplo, por Sawik (2000, 2002), Thornton e Hunsucker (2004) e Wang e Tang (2009). Como resultado, o problema de programação da produção com *buffers* limitados é convertido em um problema sem *buffers*, mas com bloqueio. O tempo de bloqueio de um processador com tempo de processamento zero denota o tempo de espera da correspondente tarefa no *buffer* representado por esse processador. Se esse tempo for zero, a tarefa não precisa esperar no *buffer*.

A Figura 3.2 ilustra um HFS no qual os *buffers* e as máquinas são conjuntamente denominados como processadores. Nesta configuração, as tarefas devem ser processadas em todas as estações, incluindo as estações *buffer*, nas quais somente operações de armazenagem podem ser feitas.

**Figura 3.2 – Representação HFS com *buffers* limitados**




Seguindo a notação de Vignier, Billaut e Proust (1999), o problema estudado pode ser descrito como:

$$FH_m, \left( \left( RM^{(i)} \right)_{i=1}^{m} \right) \left| block, S_{iljk}, A_{iljk}, M_j, rm, transpor \right| C_{\max}$$

## 3.3. Exemplo Ilustrativo

De forma ilustrativa, apresentamos um exemplo com cinco 5 tarefas e 3 estações. As estações 1 e 3 são estações de processamento, enquanto a estação 2 é uma estação de armazenagem intermediária. O número de processadores paralelos não relacionados é 2, 1 e 2, para a primeira, segunda e terceira estação respectivamente. Os tempos de liberação dos processadores na estação 1 são 163 e 182, respectivamente; 26 para o processador na estação 2 e, finalmente, 183 e 127 para os processadores na estação 3. O tempo de transporte entre as estações 1 e 2 é 7, enquanto o tempo entre as estações 2 e 3 é 8. Todas as preparações são consideradas não antecipatórias. Informações de tempos de processamentos e tempos de preparação são apresentadas na Tabela 3.1 e Tabela 3.2, respectivamente. O símbolo "-" indica que o processador não é capaz de processar a correspondente tarefa.

**Tabela 3.1 – Tempos de processamento do exemplo ilustrativo**

| Estação | 1 | | 2 | 3 | |
|---|---|---|---|---|---|
| Processador | 1 | 2 | 3 | 4 | 5 |
| Tarefa | | | | | |
| 1 | 10 | 16 | 0 | - | 76 |
| 2 | - | 97 | 0 | - | 74 |
| 3 | - | 95 | 0 | 85 | 39 |
| 4 | - | 49 | 0 | 93 | - |
| 5 | 96 | - | 0 | 68 | - |

**Tabela 3.2 – Tempos de preparação dependentes da sequência do exemplo ilustrativo**

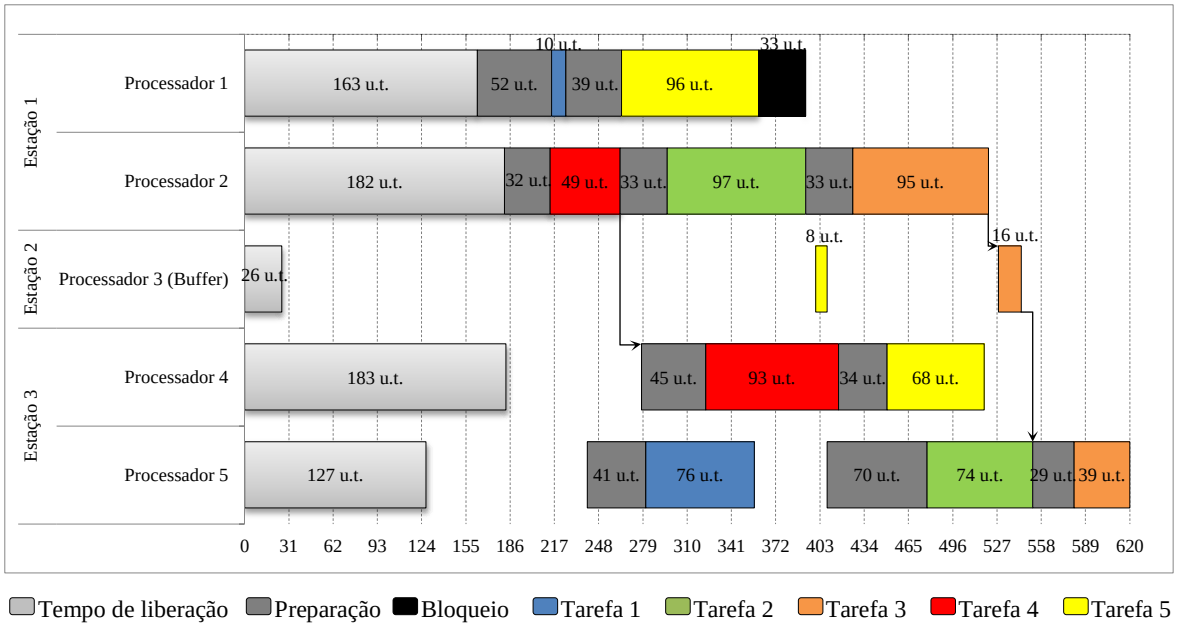
| | Processador 1 | | | | | Processador 2 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Tarefa | **1** | **2** | **3** | **4** | **5** | **1** | **2** | **3** | **4** | **5** |
| **1** | - | - | - | - | 39 | - | 48 | 51 | 62 | - |
| **2** | - | - | - | - | - | 51 | - | 33 | 47 | - |
| **3** | - | - | - | - | - | 63 | 41 | - | 29 | - |
| **4** | - | - | - | - | - | 71 | 33 | 38 | - | - |
| **5** | 72 | - | - | - | - | - | - | - | - | - |
| **0** | 52 | - | - | - | 28 | 53 | 40 | 59 | 32 | - |

| | Processador 4 | | | | | Processador 5 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | - | - | - | - | - | - | 70 | 45 | - | - |
| **2** | - | - | - | - | - | 70 | - | 29 | - | - |
| **3** | - | - | - | 65 | 31 | 72 | 30 | - | - | - |
| **4** | - | - | 68 | - | 34 | - | - | - | - | - |
| **5** | - | - | 53 | 50 | - | - | - | - | - | - |
| **0** | - | - | 52 | 45 | 45 | 41 | 37 | 53 | - | - |

O *makespan* ótimo desta instância é 620. A Figura 3.3 mostra um programa de produção ótimo para este problema. Note que depois de terminar seu processamento no processador 1, a tarefa cinco permanece na estação 1 bloqueando o processador, mesmo quando o *buffer* está disponível. No entanto, transferir a tarefa cinco para o *buffer* assim que terminar seu processamento no processador 1 não terá efeito sobre o valor do *makespan*, dado que o *buffer* está ocioso desde o momento da sua liberação. O resultado desta mudança no programa de produção será a liberação mais cedo do processador 1 e o aumento do tempo de espera da tarefa 5 no *buffer*. Note também que as tarefas 1, 2 e 4 passaram diretamente da estação 1 para a estação 3, pois os processadores elegíveis desta estação estavam disponíveis quando as tarefas finalizaram seu processamento na estação 1, fazendo desnecessária a armazenagem intermediária das tarefas. Por outro lado, as tarefas 3 e 5 tiveram um tempo de espera no *buffer* de 16 e 8 unidades de tempo, respetivamente. As setas representam tempos de transporte entre estações.

Por razões estritamente estéticas nem todas as setas são apresentadas no gráfico. Finalmente, note que a preparação de cada processador somente começa depois do transporte da tarefa designada desde a estação anterior.

**Figura 3.3 – Gráfico de Gantt do exemplo ilustrativo**



# 4. Modelo Matemático

Uma metodologia clássica para a solução de problemas de programação da produção, e problemas combinatórios em geral, consiste na definição de modelos matemáticos e resolução dos mesmos utilizando *solver* comerciais disponíveis, com o objetivo de encontrar soluções ótimas. Esta metodologia é especialmente adequada para problemas pequenos e relativamente fáceis, devido a limitações de tempo e memória que dificultam sua utilização na resolução de problemas grandes e complexos (URLINGS, 2010). Da mesma forma, Ruiz e Vázquez-Rodríguez (2010) indicam que algoritmos *Branch and Bound* (*B&B*) e modelos de programação matemática representam 25% dos artigos revisados (mais de duzentos), todos eles estudando problemas pequenos, simplificados e/ou configurações específicas.

Neste capítulo apresenta-se um modelo de programação inteira mista para o problema *Flowshop* Híbrido descrito na seção 3.2. O modelo proposto está baseado no modelo de programação inteira mista apresentado por Ruiz, Şerifoğlu e Urlings (2008), o qual representa um HFS flexível com máquinas paralelas não relacionadas, tempos de preparação dependentes da sequência, elegibilidade de máquinas, restrições de precedência, tempos de liberação das máquinas e tempos de transferência entre estações. O modelo apresentado a seguir não considera flexibilidade, restrições de precedência, nem tempos de transferência, mas inclui bloqueio devido ao espaço limitado de armazenagem intermediária e tempo de transporte entre estações. Portanto, o modelo proposto não é uma extensão, mas uma variação do modelo de Ruiz, Şerifoğlu e Urlings (2008).

O modelo é testado em um amplo conjunto de instâncias e os resultados são utilizados para determinar a viabilidade e desempenho do mesmo como ferramenta de solução.

## 4.1. Formulação do Modelo de Programação Inteira Mista

Para a apresentação do modelo de programação inteira mista considere-se a seguinte notação:

**Índice e Conjuntos:**

$j,k,h = \text{Tarefas}$;

$i,q = \text{Estações}$;

$l = \text{Processadores (Máquinas ou } \textit{buffers}\text{)}$;

$N = \text{Conjunto de tarefas}$;

$M$ = Conjunto de estações **;**

$M_i$ = Conjunto de processadores paralelos na estação $i$ **;**

$E_{ij}$ = Conjunto de processadores elegíveis para a tarefa $j$ na estação $i$ **;**

$G_{il}$ = Conjunto de tarefas que podem ser processadas no processador $l$ da estação $i$ **:**

$$G_{il} = \{ j | l \in E_{ij} \}$$

**Parâmetros:**

$rm_{il}$ = Tempo de liberação do processador $l$ da estação $i$ **;**

$p_{ilj}$ = Tempo de processamento da tarefa $j$ no processador $l$ da estação $i$;

$S_{iljk}$ = Tempo de preparação entre as tarefas $j$ e $k$ no processador $l$ da estação $i$;

$t_{iq}$ = Tempo de transporte entre as estações $i$ e $q$;

$$A_{iljk} = \begin{cases} 1, \text{ se a preparação entre as tarefas } j \text{ e } k \text{ no processador } l \text{ da estação } i \\ \quad \text{é anticipatória} \\ 0, \text{ caso contrário} \end{cases}$$

$FS(LS)$ = Primeira (última) estação do sistema.

**Variáveis de Decisão:**

$$X_{iljk} = \begin{cases} 1, \text{ se a tarefa } j \text{ precede a tarefa } k \text{ no processador } l \text{ da estação } i \\ 0, \text{ caso contrário} \end{cases}$$

$C_{ij}$ = Tempo de finalização da tarefa $j$ na estação $i$;

$d_{ij}$ = Tempo de saída da tarefa $j$ da estação $i$;

$C_{\max}$ = *Makespan.*

Note que não é necessário definir uma variável para o tempo de bloqueio da tarefa $j$ na estação $i$, pois este valor obtém-se simplesmente como $d_{ij} - C_{ij}$.

Finalmente, o modelo de programação inteira mista é apresentado a seguir:

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

$$\sum_{\substack{j \in \{N,0\} \\ j \neq k}} \sum_{l \in E_{ij} \cap E_{ik}} X_{iljk} = 1, \qquad k \in N, i \in M \tag{4.2}$$

$$\sum_{\substack{j \in N \\ j \neq k}} \sum_{l \in E_{ij} \cap E_{ik}} X_{ilkj} \leq 1, \qquad k \in N, i \in M \tag{4.3}$$

$$\sum_{\substack{h \in \{G_{il},0\} \\ h \neq j, h \neq k}} X_{ilhj} \geq X_{iljk}, \qquad j,k \in N,\ j \neq k, i \in M, l \in E_{ij} \cap E_{ik} \tag{4.4}$$

$$\sum_{l \in E_{ij} \cap E_{ik}} (X_{iljk} + X_{ilkj}) \leq 1, \qquad j \in N, k = j+1, ..., n, i \in M \tag{4.5}$$

$$\sum_{k \in G_{il}} X_{il0k} \leq 1, \qquad i \in M, l \in M_i \tag{4.6}$$

$$C_{i0} = 0, \qquad i \in M \tag{4.7}$$

$$C_{ik} + B(1 - X_{iljk}) \geq rm_{il} + (1 - A_{iljk}) S_{iljk} + p_{ilk}, \quad k \in N, i \in M, l \in E_{ik}, \; j \in \{G_{il}, 0\}, j \neq k \tag{4.8}$$

$$C_{ik} + B(1 - X_{iljk}) \geq d_{ij} + S_{iljk} + p_{ilk}, \quad k \in N, i \in M, l \in E_{ik}, \; j \in \{G_{il}, 0\}, j \neq k \tag{4.9}$$

$$C_{ik} + B(1 - X_{iljk}) \geq C_{i-1,k} + t_{i-1,i} + (1 - A_{iljk}) S_{iljk} + p_{ilk}, \quad k \in N, i \in \{M \setminus FS\}, \; l \in E_{ik}, j \in \{G_{il}, 0\}, j \neq k \tag{4.10}$$

$$d_{i-1,k} = C_{ik} - \left( \sum_{\substack{j \in \{N,0\} \\ j \neq k}} \sum_{l \in E_{ij} \cap E_{ik}} X_{iljk} \cdot \left( p_{ilk} - \left(1 - A_{iljk}\right) \cdot S_{iljk} \right) \right) - t_{i-1,i}, \qquad k \in N, i \in \{M \setminus FS\} \tag{4.11}$$

$$d_{ik} \geq C_{ik}, \qquad k \in N, i \in \{M \setminus LS\} \tag{4.12}$$

$$d_{ik} = C_{ik}, \qquad k \in N, i = LS \tag{4.13}$$

$$C_{\max} \geq C_{ik}, \qquad k \in N, i = LS \tag{4.14}$$

$$X_{iljk} \in \{0,1\}, \qquad j \in \{N, 0\}, k \in N, j \neq k, i \in M, l \in E_{ij} \cap E_{ik} \tag{4.15}$$

$$C_{ij}, d_{ij} \geq 0, \qquad j \in N, i \in M \tag{4.16}$$

A função objetivo (4.1) define a minimização do *makespan*. O conjunto de restrições (4.2) assegura que cada tarefa $k$ deve ser precedida por uma e somente uma tarefa $j$ em apenas um processador $l$ em cada estação $i$. Note a introdução da tarefa fictícia zero, a qual precede a primeira tarefa em cada processador e permite a consideração da preparação inicial de cada máquina. Note também que somente processadores elegíveis tanto para a tarefa $k$ quanto para a tarefa $j$ são considerados. Isto é necessário porque não é possível que a tarefa $j$ seja predecessora da tarefa $k$ em um processador $l$ que não é capaz de processar alguma das tarefas.

As restrições (4.3) indicam que cada tarefa $k$ teve ter, no máximo, um sucessor $j$. Observe que uma solução factível pode permitir que um processador $l$ processe somente uma tarefa $k$, a qual não teria uma tarefa sucessora. De igual modo, a última tarefa designada a um processador não tem tarefa sucessora.

O conjunto de restrições (4.4) garante que se a tarefa $j$ é predecessora de alguma tarefa $k$ em algum processador $l$ da estação $i$, então a tarefa $j$ deve ter um predecessor $h$ no mesmo processador $l$. Note que a tarefa $h$ deve ser processável pelo processador $l$, o qual é forçado pela condição $h \in \{G_{il}, 0\}$.

O conjunto de restrições (4.5) evita a ocorrência de precedência cruzada. Isto é, para quaisquer duas tarefas $j$ e $k$ processadas na estação $i$, no máximo uma das seguintes situações pode acontecer:

- A tarefa $j$ precede a tarefa $k$ em um processador $l$ da estação $i$;
- A tarefa $k$ precede a tarefa $j$ em um processador $l$ da estação $i$;
- Nem a tarefa $j$ precede a tarefa $k$ no processador $l$ da estação $i$, nem a tarefa $k$ precede a tarefa $j$ em nenhum processador $l$ da estação $i$. Isto implica que as tarefas $j$ e $k$ são processadas em processadores diferentes ou que ambas as tarefas são designadas ao mesmo processador, mas não são processadas uma depois da outra.

Observe-se que as restrições (4.5) envolvem $j,k \in N$. No entanto, a condição $j \in N,\ k = j+1,\ldots,n$ abrange todo o domínio das restrições, impedindo a geração de restrições redundantes. Considere-se, por exemplo, o conjunto de tarefas $N = \{1,2,3\}$ para as restrições (4.5). Se considerarmos $j,k \in N$, para qualquer processador $l$ na estação $i$, o seguinte conjunto de restrições é gerado:

$$\sum_{l \in E_{ij} \cap E_{ik}} (X_{il12} + X_{il21}) \leq 1, \qquad j = 1, k = 2, i \in M$$

$$\sum_{l \in E_{ij} \cap E_{ik}} (X_{il13} + X_{il31}) \leq 1, \qquad j = 1, k = 3, i \in M$$

$$\sum_{l \in E_{ij} \cap E_{ik}} (X_{il21} + X_{il12}) \leq 1, \qquad j = 2, k = 1, i \in M$$

$$\sum_{l \in E_{ij} \cap E_{ik}} (X_{il23} + X_{il32}) \leq 1, \qquad j = 2, k = 3, i \in M$$

$$\sum_{l \in E_{ij} \cap E_{ik}} (X_{il31} + X_{il13}) \leq 1, \qquad j = 3, k = 1, i \in M$$

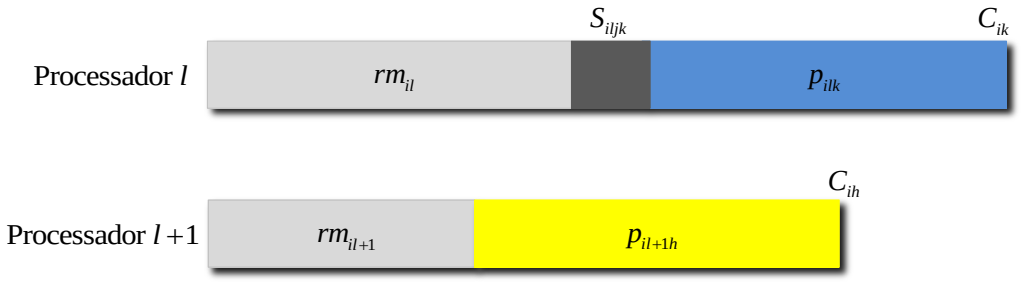
$$\sum_{l \in E_{ij} \cap E_{ik}} (X_{il32} + X_{il23}) \leq 1, \qquad j = 3, k = 2, i \in M$$

Observe que desta forma três das seis expressões anteriores são redundantes. Assim, as restrições (4.5) devem ser consideradas unicamente para $j \in N,\ k = j+1,\ldots,n,\ i \in M$.

O conjunto de restrições (4.6) indica que a tarefa fictícia zero pode ser predecessor de, no máximo, uma tarefa $k$ em cada processador $l$ em cada estação $i$. Note que a restrição (4.6) não força a utilização de todos os processadores de cada estação, permitindo ao modelo maior autonomia acerca da forma de utilizar os recursos disponíveis no sistema. Por outro lado, o conjunto de restrições (4.7) simplesmente garante que a tarefa fictícia zero seja finalizada no tempo zero em todas as estações $i \in M$.

O conjunto de restrições (4.8) indica que se a tarefa $k$ é designada ao processador $l$ da estação $i$, então seu processamento não pode começar até que o processador $l$ seja liberado pelo programa de produção anterior. Assim, as restrições (4.8) impõem um limitante inferior sobre o tempo de finalização da tarefa $k$ na estação $i$. Note que $B$ representa um número grande utilizado para tornar redundante a restrição quando a variável de designação é zero ($X_{iljk} = 0$). Observe que ambos os tipos de tempos de preparação dependentes da sequência (antecipatório e não antecipatório), também são considerados. A Figura 4.1 ilustra a restrição (4.8) considerando dois processadores, $l$ e $l+1$, os quais processam as tarefas $k$ e $h$, respectivamente.

**Figura 4.1 – Ilustração da Restrição (4.8)**

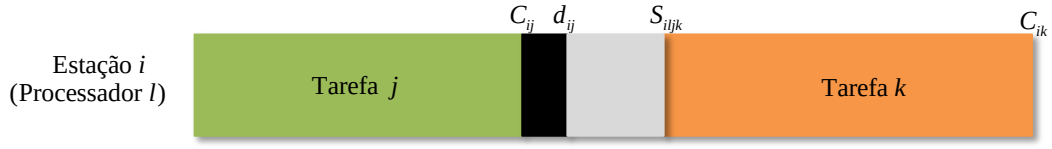



O tempo de preparação entre a tarefa $j$ e a tarefa $k$ no processador $l$ é não antecipatório. Portanto, depois da liberação do processador $l$ e antes do processamento da tarefa $k$ um tempo de preparação entre as tarefas $j$ e $k$ no processador $l$ é requerido. A situação contrária é ilustrada no processador $l+1$, no qual a preparação entre as tarefas $j$ e $h$ é antecipatória e, portanto o processamento da tarefa $h$ pode começar imediatamente depois da liberação do processador $l$, denotado por $rm_{il+1}$. Os

valores $C_{ik}$ e $C_{ih}$ representam o tempo mínimo no qual as tarefas $k$ e $h$ poderiam ser finalizadas, respectivamente.

O conjunto de restrições (4.9) impõe outro limitante inferior sobre o tempo de finalização da tarefa $k$ na estação $i$. Do mesmo modo que a restrição (4.8), a restrição (4.9) é ativa quando a variável de designação assume o valor de 1 ($X_{iljk} = 1$) e indica que o processamento da tarefa $k$ no processador $l$ da estação $i$ não pode começar até que a tarefa predecessora $j$ tenha saído da estação $i$ e a correspondente preparação do processador $l$ tenha sido feita. Note que $X_{iljk} = 1$ indica que a tarefa $j$ também é processada no processador $l$. Deste modo, a restrição (4.9) garante que o tempo de finalização da tarefa $k$ na estação $i$ não seja menor que o tempo de saída de sua tarefa predecessora $j$ na estação $i$. Observe-se que a restrição (4.9) não considera a possibilidade de preparação antecipatória ou não antecipatória, pois a preparação entre as tarefas $j$ e $k$ no processador $l$ da estação $i$ somente pode acontecer quando a tarefa $j$ tenha saído do processador $l$. A Figura 4.2 ilustra a restrição (4.9):

**Figura 4.2 – Ilustração da Restrição (4.9)**




A tarefa $j$ é processada no processador $l$ da estação $i$ e seu tempo de finalização é denotado por $C_{ij}$. Depois de ser finalizada, a tarefa $j$ permanece bloqueada durante o intervalo $d_{ij} - C_{ij}$ e sai da estação $i$ no tempo $d_{ij}$, quando algum processador da estação $i+1$ está disponível para processá-la. Após da saída da tarefa $j$, o processador $l$ deve ser preparado para o processamento da seguinte tarefa da sequência, $k$. Assim, antes de começar o processamento da tarefa $k$, $S_{iljk}$ unidades de tempo devem ser utilizadas para a preparação do processador $l$. Finalmente, a tarefa $k$ é processada durante $p_{ilk}$ unidades de tempo e seu tempo de finalização na estação $i$ é denotado por $C_{ik}$, tal que $C_{ik} \geq d_{ij} + S_{iljk} + p_{ilk}$.

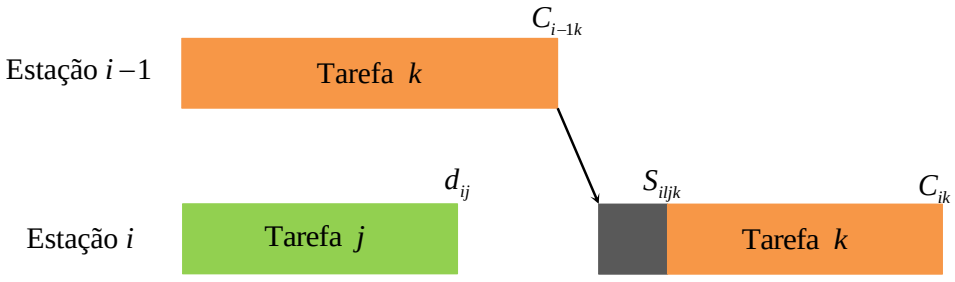
O conjunto de restrições (4.10) garante que cada tarefa $k$ seja processada subsequentemente em cada estação $i$. Assim, a restrição (4.10) estabelece que o

processamento da tarefa $k$ na estação $i$ não pode começar até que seu processamento na estação $i-1$ tenha finalizado e o transporte entre as estações $i-1$ e $i$ seja realizado, evitando que a tarefa $k$ seja processada simultaneamente em estações consecutivas.

O termo $C_{i-1,k}$ denota o tempo de finalização da tarefa $k$ na estação $i-1$, enquanto o termo $t_{i-1,i}$ denota o tempo de transporte entre as estações $i-1$ e $i$, respectivamente. Portanto, $C_{i-1,k}+t_{i-1,i}$ denota o tempo mais cedo no qual a tarefa $k$ poderia chegar à estação $i$, depois de ter sido processada na estação $i-1$. Note que o termo $\left(1-A_{iljk}\right)\cdot S_{iljk}$ considera a possibilidade de preparação antecipatória e não antecipatória. Assim, se $A_{iljk}=0$, a preparação do processador $l$ somente pode começar no tempo $C_{i-1,k}+t_{i-1,i}$, quando a tarefa $k$ chega à estação $i$, e o tempo de finalização da tarefa $k$ na estação $i$ é definido como $C_{ik}\geq C_{i-1,k}+t_{i-1,i}+S_{iljk}+p_{ilk}$. Caso contrário, se $A_{iljk}=1$, a preparação do processador $l$ pode ser feita antes da chegada da tarefa $k$ e, portanto o tempo de finalização da tarefa $k$ na estação $i$ é definido como $C_{ik}\geq C_{i-1,k}+t_{i-1,i}+p_{ilk}$, o que implica que o processamento da tarefa $k$ começa assim que ela chegar à estação $i$, pois a preparação é feita de forma antecipada.

Similarmente às restrições (4.8) e (4.9), a restrição (4.10) é ativa somente se $X_{iljk}=1$, o que implica que a tarefa $j$ precede a tarefa $k$ no processador $l$ da estação $i$ e permite identificar valores adequados dos parâmetros $A_{iljk}$ e $S_{iljk}$. A Figura 4.3 representa graficamente a restrição (4.10).

**Figura 4.3 – Ilustração da Restrição (4.10)**




As restrições (4.11) calculam o tempo de saída da tarefa $k$ da estação $i$. Lembre-se que preempção não é permitida, portanto o processamento da tarefa $k$ em qualquer processador $l$ da estação $i$ é continuo e sem interrupções. Isto implica que a

tarefa $k$ finalizada na estação $i$ no tempo $C_{ik}$ deveu começar seu processamento na estação $i$ no tempo $C_{ik} - p_{ilk}$. De igual forma, se subtrairmos o tempo de preparação, $S_{iljk}$, do tempo de início, $C_{ik} - p_{ilk}$, teremos o tempo no qual começa a preparação do processador $l$, o qual é exatamente o mesmo no qual a tarefa $k$ chega à estação $i$. Finalmente, subtraindo o tempo de transporte entre as estações $i$ e $i-1$ obtemos o tempo no qual a tarefa $k$ sai da estação $i-1$, denotada por $d_{i-1,k}$.

Note que é necessário considerar se a preparação é antecipatória ou não. Assim, se $A_{iljk} = 0$, o tempo de saída da tarefa $k$ da estação $i-1$ é definida como $d_{i-1,k} = C_{ik} - p_{ilk} - S_{iljk} - t_{i-1,i}$. Caso contrário, se $A_{iljk} = 1$, o tempo de saída da tarefa $k$ da estação $i-1$ é definida através da expressão $d_{i-1,k} = C_{ik} - p_{ilk} - t_{i-1,i}$, na qual o tempo de preparação não é considerado, pois esta é feita de forma antecipada.

Observe-se que para calcular $d_{i-1,k}$ devemos conhecer o processador $l$ da estação $i$ no qual a tarefa $k$ é processada, assim como a tarefa $j$ que precede a tarefa $k$ no processador $l$ da estação $i$. Estas informações são capturadas pela expressão $\sum_{\substack{j \in \{N,0\} \\ j \neq k}} \sum_{l \in E_{ij} \cap E_{ik}} X_{iljk} \cdot \left( p_{ilk} - \left(1 - A_{iljk}\right) \cdot S_{iljk} \right)$, a qual assume o valor $p_{ilk} - \left(1 - A_{iljk}\right) \cdot S_{iljk}$ se $X_{iljk} = 1$, ou 0 se $X_{iljk} = 0$.

A Figura 4.4 ilustra a restrição (4.11) quando a preparação entre as tarefas $j$ e $k$ no processador $l$ da estação $i$ é não antecipatória, enquanto a Figura 4.5 ilustra o caso quando a preparação é antecipatória.

**Figura 4.4 – Ilustração da Restrição (4.11) com preparação não antecipatória**

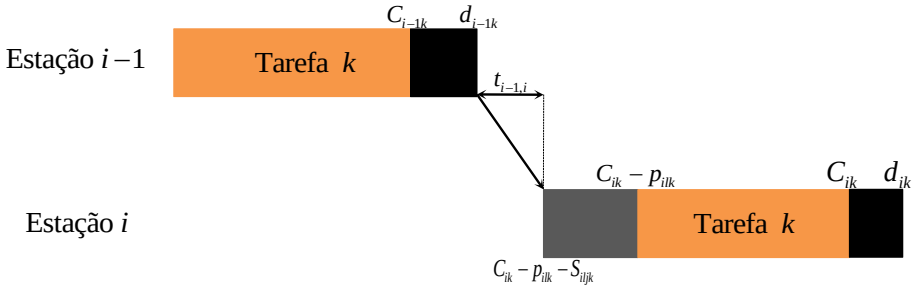
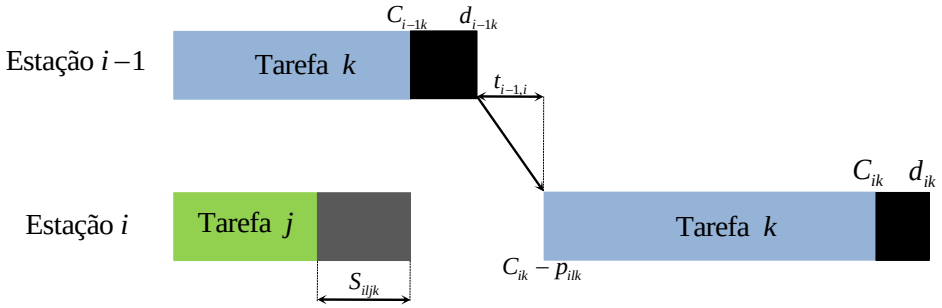


**Figura 4.5 – Ilustração da Restrição (4.11) com preparação antecipatória**




As restrições (4.12) indicam que o tempo de saída da tarefa $k$ da estação $i$ não pode ser menor que seu tempo de finalização nesta estação. Isto garante que a tarefa $k$ não possa sair da estação $i$ até que seu processamento nesta estação seja finalizado. Note que a expressão $d_{ik} - C_{ik} \geq 0$ representa o tempo de bloqueio da tarefa $k$ na estação $i$. O conjunto de restrições (4.13) garante que a tarefa $k$ sai do sistema, assim que finalizar seu processamento na última estação. A restrição (4.14) define o *makespan* como o máximo tempo de finalização na última estação do sistema e, finalmente, as restrições (4.15) e (4.16) definem o domínio das variáveis de decisão.

O modelo apresentado foi utilizado para resolver algumas instâncias geradas especificamente para avaliar se este representa adequadamente o problema estudado. O exemplo ilustrativo da seção 3.3 foi resolvido utilizando o modelo de programação inteira descrito nesta seção.

## 4.2. Avaliação Computacional

Um conjunto de instâncias foi gerado com o objetivo de testar e avaliar o desempenho do modelo de programação inteira mista proposto na seção 4.1. Este conjunto foi gerado a partir da combinação de diferentes fatores que representam as diversas características do problema estudado. A Tabela 4.1 apresenta os fatores e valores considerados para a geração de instâncias de teste do modelo de programação inteira mista.

**Tabela 4.1 – Fatores utilizados na geração de instâncias para o modelo de programação inteira mista**

| Fatores | Símbolo | Valores |
|---|---|---|
| Número de tarefas | $n$ | 5, 7, 9, 11 |
| Número de estações | $m$ | 3, 5, 7 |
| Número de processadores paralelos em estações de processamento | $m_i$ | 2, 3 |
| Número de processadores paralelos em estações *buffer* | $b_i$ | $m_i/2, m_i$ |
| Probabilidade de um processador ser elegível | $PE_{ij}$ | 0.50, 1 |
| Distribuição dos tempos de preparação | $DS_{iljk}$ | $U[25,74], U[75,125]$ |
| Probabilidade do tempo de preparação ser antecipatório | $PA_{iljk}$ | 0, 0.50, 1 |

Os valores dos parâmetros mostrados na Tabela 4.1 foram escolhidos com o objetivo de obter instâncias de pequeno porte que representem diferentes cenários para o problema estudado. Por exemplo, uma instância com três estações representa em realidade um problema com duas estações de processamento e uma de armazenagem intermediária, assim como uma instância com cinco estações representa um problema com três estações de processamento e duas de armazenagem intermediária. Note que os valores considerados para os parâmetros $m_i$ e $b_i$ permitem gerar instâncias nas quais há menos *buffers* que máquinas paralelas, $b_i = m_i/2$, e instâncias nas quais há igual número de *buffers* que máquinas paralelas, $b_i = m_i$. Assim, espera-se ter instâncias com diferentes níveis de bloqueio e ociosidade dos processadores.

O parâmetro $PE_{ij} \in \{0.5,1\}$ permite a consideração de dois cenários diferentes. Por exemplo, instâncias nas quais $PE_{ij} = 0.5$ representam problemas nos quais somente alguns processadores são elegíveis para o processamento das tarefas. Por outro lado, instâncias nas quais $PE_{ij} = 1$ refletem a situação ideal na qual todos os processadores são capazes de processar todas as tarefas a serem programadas. Similarmente, o parâmetro $PA_{iljk} \in \{0, 0.5, 1\}$ permite gerar instâncias com diferentes tipos de preparação. Por exemplo, quando $PA_{iljk} = 0$ todas as preparações são não antecipatórias. O caso contrário é representado por $PA_{iljk} = 1$, no qual todas as preparações são antecipatórias. Finalmente, o caso intermediário, no qual há preparações tanto antecipatórias quanto não antecipatórias, é representado através de $PA_{iljk} = 0.5$.

Como realizado comumente na literatura da programação da produção, o tempo de processamento das tarefas é gerado seguindo uma distribuição uniforme $U[1,99]$ (TAILLARD, 1993). Igualmente, os tempos de preparação são gerados a partir da distribuição uniforme $U[25,74]$ e $U[75,125]$, respectivamente. Assim, os tempos de preparação correspondem a 25-74% e 75-125% dos tempos de processamento, respectivamente (RUIZ; MAROTO, 2006; RUIZ; ŞERIFOĞLU; URLINGS, 2008; YAURIMA; BURTSEVA; TCHERNYKH, 2009).

Esta estratégia permite gerar instâncias com tempos de preparação menores que os tempos de processamento, assim como instâncias com tempos de preparação maiores que os tempos de processamento. O tempo de liberação dos processadores é

fixado no intervalo $U[1,200]$ (RUIZ; ŞERIFOĞLU; URLINGS, 2008; URLINGS; RUIZ; ŞERIFOĞLU, 2010; URLINGS; RUIZ; STÜTZLE, 2010). Finalmente o tempo de transporte entre estações é fixado no intervalo $U[1,10]$.

Observe que o conjunto de processadores paralelos na estação $i$, $M_i$, é composto por $m_i$ máquinas paralelas se $i$ é uma estação de processamento. Caso contrário, se $i$ é uma estação de armazenagem intermediária, $M_i$ é composto por $b_i$ *buffers* paralelos. O número total de combinações é $4 \times 3^2 \times 2^4 = 576$. Para cada combinação três replicas são geradas, portanto há 1.728 instâncias. É importante destacar que quando as instâncias foram geradas, foi garantido que em cada estação de processamento cada tarefa $j$ tivesse, no mínimo, um processador elegível, isto é, $|E_{ij}| \geq 1$. No caso de estações de armazenagem intermediária, considerou-se que todos os processadores são elegíveis para cada tarefa $j$, assim $|E_{ij}| = b_i$.

O modelo de programação inteira mista da seção 4.1 foi implementado no sistema de modelagem para programação matemática e otimização *General Algebraic Modeling System* (*GAMS*) e cada instância foi resolvida utilizando o *solver* comercial *CPLEX* 12 em um *notebook* com processador Intel Core i5 2.67 GHz com 4 Gigabytes de memória *RAM*. Devido à quantidade instâncias geradas, um tempo limite de execução de 3.600 segundos foi imposto para cada instância.

Para $n = 5$, independente dos demais fatores, todas as instâncias são resolvidas até otimalidade e, portanto pouca informação interessante em relação ao comportamento do modelo pode ser extraída. Desta forma os resultados de estas instâncias não serão analisados.

A Tabela 4.2 apresenta os resultados gerais da avaliação do modelo para $n = 7$ e $m = 7$. Cada célula representa a média das três réplicas geradas para cada combinação de fatores. Assim, são apresentadas a percentagem de instâncias cuja solução ótima foi encontrada (%Ótimo); a percentagem de instâncias para as quais uma solução inteira factível foi encontrada (%IntSol); a percentagem de instâncias para as quais nenhuma solução inteira factível foi encontrada (%NãoSol) e, finalmente, o tempo computacional médio requerido para encontrar a solução ótima (Tem. Médio). Para os casos nos quais não foi possível encontrar uma solução ótima, o símbolo "-" é utilizado como entrada na célula **Tem. Médio.** Note que a análise dos resultados considerando

conjuntamente todos os fatores é complexa e dispendiosa e, portanto a informação deveria ser agregada com o objetivo de simplificar a análise dos resultados.

**Tabela 4.2 – Resultados do modelo de programação inteira mista para *n*=7 e *m*=7**

| | | | $PE_{ij}$ | 0,5 | | | 1 | | |
|---|---|---|---|---|---|---|---|---|---|
| $m_i$ | $b_i$ | $DS_{iljk}$ | $PA_{iljk}$ | 0 | 0,5 | 1 | 0 | 0,5 | 1 |
| 2 | 1 | *U*[25,74] | %Ótimo | 66,67 | 33,33 | 100,00 | 66,67 | 66,67 | 100,00 |
| | | | %IntSol | 33,33 | 66,67 | 0,00 | 0,00 | 33,33 | 0,00 |
| | | | %NãoSol | 0,00 | 0,00 | 0,00 | 33,33 | 0,00 | 0,00 |
| | | | Tem. Médio | 1095,55 | 339,12 | 128,46 | 429,76 | 910,52 | 22,83 |
| | | *U*[75,125] | %Ótimo | 33,33 | 33,33 | 100,00 | 66,67 | 66,67 | 100,00 |
| | | | %IntSol | 0,00 | 66,67 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | %NãoSol | 66,67 | 0,00 | 0,00 | 33,33 | 33,33 | 0,00 |
| | | | Tem. Médio | 540,70 | 133,13 | 525,36 | 50,68 | 57,86 | 46,96 |
| | 2 | *U*[25,74] | %Ótimo | 100,00 | 100,00 | 100,00 | 66,67 | 66,67 | 100,00 |
| | | | %IntSol | 0,00 | 0,00 | 0,00 | 33,33 | 33,33 | 0,00 |
| | | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | Tem. Médio | 274,75 | 152,26 | 76,95 | 42,32 | 25,64 | 74,02 |
| | | *U*[75,125] | %Ótimo | 100,00 | 100,00 | 100,00 | 66,67 | 66,67 | 66,67 |
| | | | %IntSol | 0,00 | 0,00 | 0,00 | 33,33 | 33,33 | 33,33 |
| | | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | Tem. Médio | 2011,06 | 1187,05 | 219,16 | 647,29 | 1150,04 | 26,57 |
| 3 | 2 | *U*[25,74] | %Ótimo | 100,00 | 100,00 | 100,00 | 66,67 | 100,00 | 100,00 |
| | | | %IntSol | 0,00 | 0,00 | 0,00 | 33,33 | 0,00 | 0,00 |
| | | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | Tem. Médio | 125,13 | 444,49 | 833,59 | 10,23 | 1157,24 | 39,72 |
| | | *U*[75,125] | %Ótimo | 100,00 | 100,00 | 100,00 | 33,33 | 100,00 | 100,00 |
| | | | %IntSol | 0,00 | 0,00 | 0,00 | 66,67 | 0,00 | 0,00 |
| | | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | Tem. Médio | 116,49 | 761,67 | 41,18 | 4,32 | 453,46 | 267,35 |
| | 3 | *U*[25,74] | %Ótimo | 100,00 | 100,00 | 100,00 | 100,00 | 100,00 | 100,00 |
| | | | %IntSol | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | Tem. Médio | 145,66 | 87,31 | 19,11 | 12,74 | 50,05 | 4,75 |
| | | *U*[75,125] | %Ótimo | 100,00 | 100,00 | 100,00 | 100,00 | 100,00 | 100,00 |
| | | | %IntSol | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| | | | Tem. Médio | 889,80 | 492,59 | 21,21 | 189,08 | 78,87 | 19,09 |

A Tabela 4.3 apresenta resultados considerando somente a interação entre $m_i$ e $b_i$, tomando a média sobre os demais fatores. Como observado, a percentagem de instâncias resolvidas até otimalidade é sensível à interação entre $m_i$ e $b_i$, mostrando um incremento quando $b_i$ aumenta. Note, por exemplo, que o subconjunto de instâncias com $m_i = 2$ e $b_i = 1$ é mais difícil de resolver, com a maior percentagem de instâncias sem solução inteira factível. Adicionalmente, observe o incremento de instâncias resolvidas até otimalidade quando $m_i = 2$ e $b_i = 2$, assim como a redução da percentagem de instâncias sem solução inteira factível.

**Tabela 4.3 – Resultados $m_i \times b_i$ para $n = 7; m = 7$**

| $m_i \times b_i$ | %Ótimo | %IntSol | %NoSol |
|---|---|---|---|
| $2 \times 1$ | 69,44 | 16,67 | 13,89 |
| $2 \times 2$ | 86,11 | 13,89 | 0,00 |
| $3 \times 2$ | 91,67 | 8,33 | 0,00 |
| $3 \times 3$ | 100,00 | 0,00 | 0,00 |

Este mesmo comportamento é observado em instâncias com $m_i = 3$, nas quais há um aumento na percentagem de instâncias resolvidas até otimalidade quando $b_i$ passa de dois para três. Contrário ao esperado, instâncias com $m_i = 3$ são mais fáceis de resolver que instâncias com $m_i = 2$. Note que a percentagem média de instâncias com solução ótima é de 95,83% quando $m_i = 3$, enquanto este valor diminui até 77,78% quando $m_i = 2$. Em principio, este resultado é inesperado, pois com mais processadores paralelos por estação maior é o número de variáveis do modelo e, portanto maior a dificuldade de encontrar uma solução ótima.

Ruiz, Şerifoğlu e Urlings (2008) indicam que esta situação é explicada pelo fato de que o número de tarefas $n$ é o fator mais influente em problemas de programação da produção e, portanto as decisões de sequenciamento das tarefas são mais relevantes. Porém, quando o número de processadores paralelos aumenta a designação de tarefas a processadores torna-se importante, ajudando a atenuar o efeito puro do número de tarefas na dificuldade das instâncias.

Também é interessante analisar o impacto do parâmetro $PE_{ij}$, o qual está diretamente relacionado com o número de processadores capazes de processar uma determinada tarefa. Quando $PE_{ij} = 0,5$ a percentagem de instâncias com solução ótima é 90,28%, enquanto este valor diminui até 83,33% quando $PE_{ij} = 1$. Note que este resultado é contraditório com as conclusões expostas acerca do parâmetro $m_i$, no entanto uma análise conjunta dos dois parâmetros permite validar ambas as conclusões. A Tabela 4.4 apresenta os resultados da interação entre $m_i$ e $PE_{ij}$.

**Tabela 4.4 – Resultados $PE_{ij} \times m_i$ para $n = 7; m = 7$**

| $PE_{ij}$ | $m_i$ | %Ótimo | %IntSol | %NoSol |
|---|---|---|---|---|
| 0,5 | 2 | 80,56 | 13,89 | 5,56 |
| 0,5 | 3 | 100,00 | 0,00 | 0,00 |
| 1 | 2 | 75,00 | 16,67 | 8,33 |
| 1 | 3 | 91,67 | 8,33 | 0,00 |

Note que quando $PE_{ij} = 0,5$ e $m_i = 2$, a percentagem de instâncias com solução ótima é 80,56%, enquanto esta percentagem diminui até 75% no caso $PE_{ij} = 1$ e $m_i = 2$. Este mesmo padrão é observado quando compararmos $PE_{ij} = 0,5$ e $m_i = 3$ com $PE_{ij} = 1$ e $m_i = 3$, cujas percentagens são 100% e 91,67%, respectivamente. Esta situação evidencia que a dificuldade na resolução das instâncias tende a ser maior quando há mais processadores elegíveis por estação ($PE_{ij} = 1$). Note também que os valores para $m_i = 3$ são maiores que os valores para $m_i = 2$. Portanto, para cada valor de $PE_{ij}$, as instâncias são mais fáceis de resolver quando $m_i = 3$.

É importante também analisar o comportamento do modelo em relação à duração e tipo dos tempos de preparação. A Tabela 4.5 sintetiza os resultados para $DS_{iljk}$ e $PA_{iljk}$.

**Tabela 4.5 – Resultado $DS_{iljk} \times PA_{iljk}$ para $n = 7; m = 7$**

| $DS_{iljk}$ | $PA_{iljk}$ | %Ótimo | %IntSol | %NãoSol |
|---|---|---|---|---|
| *U*[25,74] | 0 | 83,33 | 12,50 | 4,17 |
| *U*[25,74] | 0,5 | 83,33 | 16,67 | 0,00 |
| *U*[25,74] | 1 | 100,00 | 0,00 | 0,00 |
| *U*[75,125] | 0 | 75,00 | 12,50 | 12,50 |
| *U*[75,125] | 0,5 | 83,33 | 12,50 | 4,17 |
| *U*[75,125] | 1 | 95,83 | 4,17 | 0,00 |

Os resultados da Tabela 4.5 mostram que, em geral, a percentagem de instâncias com solução ótima tende a aumentar quando a probabilidade dos tempos de preparação ser antecipatórios, $PA_{iljk}$, também aumenta. Por exemplo, quando $DS_{iljk} = U[25,74]$ e $PA_{iljk} = 0$, a percentagem de instâncias com solução ótima é 83,33%, enquanto este valor aumenta até 100% quando $PA_{iljk} = 1$. O mesmo comportamento é observado quando $DS_{iljk} = U[75,125]$. Neste caso, as percentagens de instâncias com solução ótima são 75% e 95,83% para $PA_{iljk} = 0$ e $PA_{iljk} = 1$, respectivamente. Este resultado é esperado, pois quando a preparação das tarefas pode ser antecipada, a solução final depende principalmente da designação e sequenciamento das tarefas nos processadores que podem finalizá-las no tempo mais cedo. Note que a duração do tempo de preparação é irrelevante, pois esta é feita de forma antecipada.

Por outro lado, quando as preparações são dependentes da sequência e não antecipatórias, não é suficiente designar e sequenciar no processador mais veloz, mas é preciso considerar informações acerca de tarefas predecessoras e duração da preparação entre tarefas para garantir a designação e sequenciamento mais adequados. Lembre-se que com a presença de preparações dependentes da sequência, o processador mais veloz não necessariamente finaliza a tarefa $k$ no tempo mais cedo, pois o tempo de preparação entre a tarefa prévia $j$ e a tarefa $k$ poderia ser longo, resultando em um tempo de finalização maior.

Observe que a Tabela 4.5 indica que a percentagem de instâncias com solução ótima é pouco sensível às variações do fator $DS_{iljk}$, pois a percentagem média de instâncias resolvidas até otimalidade é 88,89% e 84,72% para $DS_{iljk} = U[25,74]$ e $DS_{iljk} = U[75,125]$, respectivamente. Assim, o fator $DS_{iljk}$ parece ter pouca, ou nenhuma, relevância na complexidade do problema.

A Tabela 4.6 apresenta os resultados consolidados do modelo de programação inteira mista em termos do número de tarefas, número de estações, número de processadores paralelos por estação de processamento e número de *buffers* de armazenagem intermediária. Deste modo, cada célula da Tabela 4.6 representa a média do subconjunto de instâncias com $n$ tarefas, $m$ estações, $m_i$ processadores paralelos e $b_i$ *buffers*, incluindo os demais fatores do problema ($PE_{ij}$, $PA_{iljk}$, $DS_{iljk}$).

Note que todas as instâncias com $n = 5$ são resolvidas até otimalidade em tempos computacionais curtos. Como esperado, a dificuldade na resolução das instâncias aumenta na medida em que o número de tarefas e estações aumenta. Note-se também que aumentar o número de tarefas reduz em maior proporção a percentagem de instâncias resolvidas até otimalidade em comparação com a redução causada quando aumentarmos o número de estações. Este comportamento é um indício da maior relevância do número de tarefas na dificuldade das instâncias e, por tanto, na dificuldade para encontrar uma solução ótima. A Figura 4.6 ilustra esta situação.

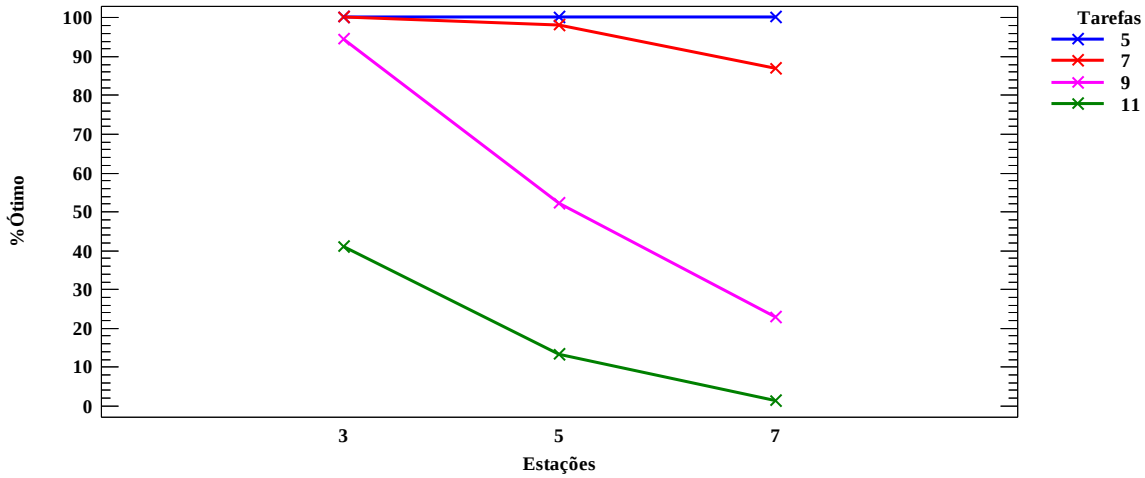
Os resultados consolidados do modelo de programação inteira mista indicam que a percentagem de instâncias com solução ótima foi 62,85% quando $m_i = 2$, enquanto este valor aumenta para 72,11% quando $m_i = 3$, o que confirma as conclusões expostas anteriormente. Por outro lado, a percentagem de instâncias resolvidas até otimalidade não apresenta variações significativas quando o parâmetro $b_i$ é modificado.

Especificamente, as percentagens de instâncias com solução ótima são 68,98% e 65,97% para $b_i = m_i$ e $b_i = m_i/2$, respectivamente. Em termos gerais, isto implica que o tamanho do *buffer* não influencia na dificuldade para resolver as instâncias através do *solver* CPLEX.

**Tabela 4.6 – Resultados consolidados do modelo de programação inteira mista**

| | | $m_i$ | 2 | | 3 | |
|---|---|---|---|---|---|---|
| $n$ | $m$ | $b_i$ | $m_i/2$ | $m_i$ | $m_i/2$ | $m_i$ |
| 5 | 3 | %Ótimo | 100,00 | 100,00 | 100,00 | 100,00 |
| | | %IntSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | Tem. Médio | 0,38 | 0,46 | 0,34 | 0,36 |
| | 5 | %Ótimo | 100,00 | 100,00 | 100,00 | 100,00 |
| | | %IntSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | Tem. Médio | 1,21 | 1,64 | 1,05 | 0,99 |
| | 7 | %Ótimo | 100,00 | 100,00 | 100,00 | 100,00 |
| | | %IntSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | Tem. Médio | 42,05 | 7,80 | 1,97 | 2,38 |
| 7 | 3 | %Ótimo | 100,00 | 100,00 | 100,00 | 100,00 |
| | | %IntSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | Tem. Médio | 11,49 | 5,67 | 3,90 | 4,33 |
| | 5 | %Ótimo | 94,44 | 97,22 | 100,00 | 100,00 |
| | | %IntSol | 5,56 | 2,78 | 0,00 | 0,00 |
| | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | Tem. Médio | 137,84 | 172,44 | 40,61 | 39,41 |
| | 7 | %Ótimo | 69,44 | 86,11 | 91,67 | 100,00 |
| | | %IntSol | 16,67 | 13,89 | 8,33 | 0,00 |
| | | %NãoSol | 13,89 | 0,00 | 0,00 | 0,00 |
| | | Tem. Médio | 330,90 | 508,69 | 386,23 | 167,52 |
| 9 | 3 | %Ótimo | 86,11 | 97,22 | 100,00 | 94,44 |
| | | %IntSol | 13,89 | 2,78 | 0,00 | 5,56 |
| | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | Tem. Médio | 305,15 | 468,70 | 141,35 | 252,21 |
| | 5 | %Ótimo | 25,00 | 47,22 | 61,11 | 75,00 |
| | | %IntSol | 72,22 | 50,00 | 38,89 | 25,00 |
| | | %NãoSol | 2,78 | 2,78 | 0,00 | 0,00 |
| | | Tem. Médio | 780,41 | 663,19 | 550,46 | 429,63 |
| | 7 | %Ótimo | 8,33 | 27,78 | 33,33 | 22,22 |
| | | %IntSol | 83,33 | 66,67 | 58,33 | 75,00 |
| | | %NãoSol | 8,33 | 5,56 | 8,33 | 2,78 |
| | | Tem. Médio | 1411,75 | 1533,70 | 932,09 | 1193,35 |
| 11 | 3 | %Ótimo | 25,00 | 33,33 | 58,33 | 47,22 |
| | | %IntSol | 75,00 | 66,67 | 41,67 | 52,78 |
| | | %NãoSol | 0,00 | 0,00 | 0,00 | 0,00 |
| | | Tem. Médio | 1348,72 | 348,30 | 469,16 | 548,50 |
| | 5 | %Ótimo | 2,78 | 5,56 | 25,00 | 19,44 |
| | | %IntSol | 94,44 | 94,44 | 63,89 | 75,00 |
| | | %NãoSol | 2,78 | 0,00 | 11,11 | 5,56 |
| | | Tem. Médio | 2086,37 | 1183,19 | 543,81 | 1042,76 |
| | 7 | %Ótimo | 0,00 | 2,78 | 2,78 | 0,00 |
| | | %IntSol | 97,22 | 80,56 | 72,22 | 66,67 |
| | | %NãoSol | 2,78 | 16,67 | 25,00 | 33,33 |
| | | Tem. Médio | - | 1504,36 | 930,62 | - |

**Figura 4.6 – Percentagem de instâncias com solução ótima: Tarefas vs. Estações**




A informação da Tabela 4.7 resume o *gap* médio de todas as instâncias para as quais o *solver* encontrou uma solução inteira factível, o qual oscila entre 22,25% e 53,77% e tende a ser maior com o aumento do número de tarefas e estações. Este comportamento era esperado, pois quanto maior número de tarefas e estações, maior dificuldade teve o *solver* para resolver as instâncias (ver Figura 4.6).

**Tabela 4.7 – *Gap* médio de instâncias com solução inteira factível**

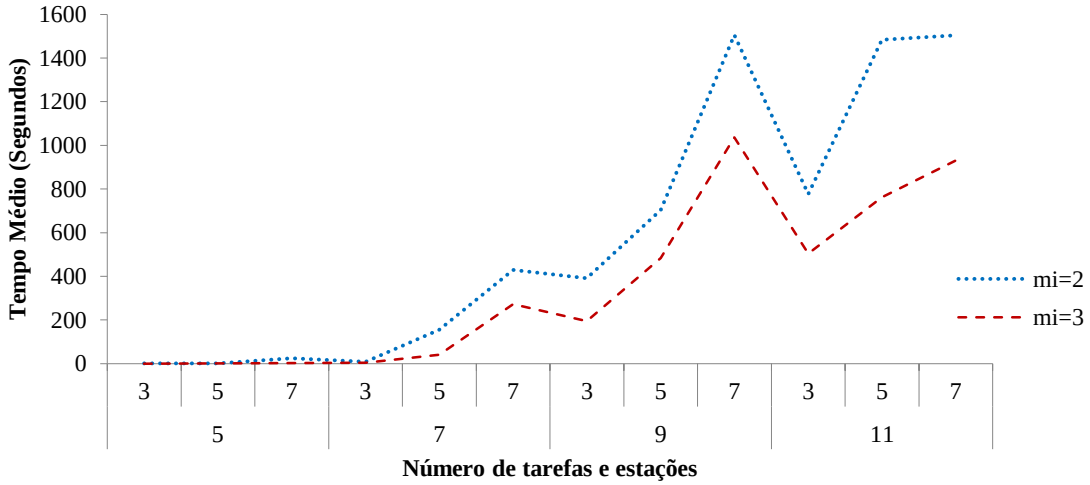
| $n \times m$ | $m_i = 2$ | $m_i = 3$ |
|---|---|---|
| 7×5 | 31,86 | - |
| 7×7 | 29,84 | 31,56 |
| 9×3 | 28,97 | 22,25 |
| 9×5 | 33,96 | 24,99 |
| 9×7 | 39,03 | 31,43 |
| 11×3 | 36,56 | 30,23 |
| 11×5 | 49,19 | 42,87 |
| 11×7 | 53,77 | 43,73 |

Em relação ao tempo computacional, aumentar o número de tarefas e estações aumenta consideravelmente o tempo requerido para encontrar uma solução ótima, tal como esperado. A análise do tempo médio permite identificar certos valores para o número de tarefas e o número de estações a partir dos quais o tempo utilizado para solucionar o problema torna-se proibitivo e, portanto algum método alternativo para encontrar uma solução deve ser considerado. Em particular, para instâncias além de nove tarefas e cinco estações, atingir uma solução ótima é inviável do ponto de vista prático, dado a quantidade de tempo requerida.

A Figura 4.7 ilustra melhor o comportamento de tempo médio utilizado pelo *solver* como função do número de tarefas e estações. Nesta figura, é possível observar como o *solver* requer mais tempo na medida em que o número de tarefas e estações

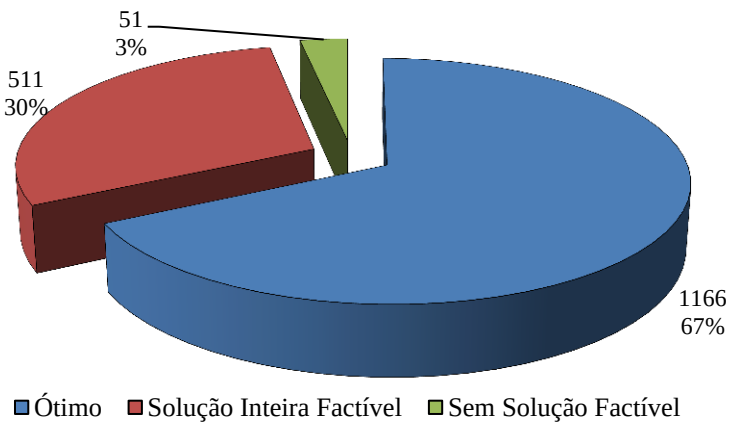
aumenta. Igualmente, observe o menor tempo requerido para resolver instâncias com maior número de máquinas paralelas, o qual indica uma menor dificuldade para resolver tais instâncias. Assim, instâncias com $m_i = 3$ são mais fáceis de resolver que instâncias com $m_i = 2$, tanto pela maior percentagem de instâncias com solução ótima quanto pelo menor esforço computacional requerido.

**Figura 4.7 – Tempo médio utilizado pelo *solver* para encontrar uma solução ótima**




Finalmente, um resumo geral dos resultados do modelo de programação inteira mista é apresentado na Figura 4.8. Como observado, o *solver* encontrou ao menos uma solução ótima em 1.166 instâncias, as quais representam o 67% do total de instâncias resolvidas. Para 511 instâncias, equivalentes a 30% do total de instâncias, ao menos uma solução inteira factível foi encontrada. Finalmente, para as 51 instâncias restantes, representando o 3% do total, nenhuma solução inteira factível foi encontrada depois de 3.600 segundos de execução.

**Figura 4.8 – Resumo geral do modelo de programação inteira mista**



Os resultados apresentados neste capítulo indicam que o modelo de programação inteira mista é útil para a resolução de instâncias de até nove tarefas e cinco estações. Por outro lado, considerando os recursos de tempo e memória disponibilizados para a avaliação do modelo, o *gap* médio é elevado e, portanto pode-se concluir que para este conjunto de instâncias as solução obtidas não foram boas. De igual forma, o alto custo computacional requerido torna inviável a utilização do modelo para a resolução de problemas de maior tamanho. Desta forma, métodos heurísticos e meta-heurísticos devem ser utilizados para obter soluções de boa qualidade em curtos tempos computacionais. Estes métodos serão abordados nos capítulos 5 e 6, respectivamente.

# 5. Heurísticas

Heurísticas são métodos de resolução de problemas de otimização discreta, que não garantem a obtenção de uma solução factível ou ótima. Nicholson (1971) propôs uma definição que expressa muito bem as características de uma heurística: é um procedimento para resolver problemas através de um enfoque "intuitivo", em geral racional, no qual a estrutura do problema possa ser interpretada e explorada inteligentemente para se obter uma solução razoável.

Problemas complexos requerem a avaliação de um imenso número de possibilidades para determinar uma solução exata e, frequentemente, o tempo requerido para encontrar tal solução é muito elevado. Portanto, métodos heurísticos são frequentemente utilizados para identificar boas soluções aproximadas em menor tempo que o requerido para encontrar a solução ótima através de um algoritmo exato (BARR et al., 1995).

Como descrito na seção 4.2, o tempo computacional do modelo de programação inteira mista, para o problema tratado neste trabalho, tende a incrementar consideravelmente quando o número de tarefas e estações aumenta. Assim, encontrar a solução ótima para problemas além de nove tarefas e cinco estações torna-se inviável do ponto de vista prático, dado o alto custo computacional que é requerido. Portanto, neste capítulo serão apresentados métodos heurísticos tradicionalmente utilizados na resolução do *Flowshop* clássico, devidamente adaptados para resolver o HFS aqui considerado.

A estrutura deste capítulo é a seguinte: a seção 5.1 descreve as regras utilizadas para designar tarefas a processadores. A seção 5.2 apresenta a heurística de Campbell, Dudek e Smith (1970) (CDS, pelas iniciais dos sobrenomes dos autores), a qual é considerada para propósitos comparativos e devido a sua relevância teórica, pois a literatura reporta várias heurísticas superiores. A seção 5.3 descreve a heurística proposta por Nawaz, Enscore e Ham (1983) (NEH, pelas iniciais dos sobrenomes dos autores), a qual é considerada a melhor heurística construtiva para o *Flowshop* clássico. Na seção 5.4 é apresentada a heurística proposta recentemente por Kalczynski e Kamburowski (2008) (NEHKK1, pelas iniciais dos sobrenomes dos autores), a qual utiliza conceitos do algoritmo de Johnson para melhorar o desempenho da heurística NEH. A seção 5.5 descreve a heurística proposta por Ronconi (2004), denominada PFE, a qual combina a heurística NEH com a heurística PF (*Profile Fitting*), explorando características específicas do *Flowshop* clássico com bloqueio. A seção 5.6 apresenta a

heurística PF-NEH, proposta por Pan e Wang (2012), a qual utiliza outro método para combinar as heurísticas NEH e PF com o objetivo de superar aparentes desvantagens da heurística PFE. Finalmente na seção 5.7 são apresentados os resultados da avaliação computacional.

## 5.1. Regras de Designação

Todas as heurísticas utilizadas neste capítulo geram como saída uma sequência, ou permutação, na qual as tarefas devem ser programadas na primeira estação do sistema. Note que em um HFS uma permutação não é suficiente para elaborar um programa de produção completo, pois as tarefas devem ser designadas a processadores elegíveis em cada estação. Portanto, para gerar um programa de produção completo, algum método de designação de tarefas aos processadores deve ser considerado. Nesta dissertação, as regras de designação primeira máquina (processador) disponível (FAM, *First Available Machine*) e tempo de finalização mais cedo (ECT, *Earliest Completion Time*) são consideradas. Estas regras de designação utilizam informação sobre a tarefa $j$ a ser programada e os $m_i$ processadores disponíveis na estação atual $i$. Por outro lado, fazendo suposições sobre decisões futuras, ou simplesmente usando médias, também é possível usar informação acerca de tarefas ainda não programadas e processadores em estações subsequentes. Regras de designação que levam em conta este tipo de informação são denominadas *Look-ahead* e seu custo computacional geralmente é maior. Portanto, a escolha entre regras *Look-ahead* e regras mais simples, como FAM e ECT, representa um compromisso entre a probabilidade de obter bons programas de produção e esforço computacional requerido. Nesta dissertação, dada a complexidade própria do problema estudado, não são utilizadas regras do tipo *Look-ahead*. No entanto, o desenvolvimento deste tipo de regras representa uma interessante direção de pesquisa futura.

Antes de descrever as regras de designação alguma notação adicional é requerida. O processador escolhido para o processamento da tarefa $j$ na estação $i$ é denotado por $T_{ij}$, ou de forma mais breve como $l$. A última tarefa que foi designada ao processador $l$ da estação $i$ é denotada como $L_{il}$. Adicionalmente, $i-1$ representa a última estação visitada pela tarefa $j$ e $i+1$ denota a seguinte estação a ser visitada pela tarefa $j$, respectivamente. Considere-se que inicialmente $L_{il} = C_{i0} = d_{i0} = 0$, $i \in M$ e

$l \in M_i$. Agora, o tempo de finalização da tarefa $j$ na estação $i$ pode ser calculado através das seguintes equações:

$$C_{ij} = \max\left\{rm_{il}, d_{iL_{il}} + S_{il,L_{il},j} \cdot A_{il,L_{il},j}\right\} + \left(1 - A_{il,L_{il},j}\right) \cdot S_{il,L_{il},j} + p_{ilj} \quad j \in N, i = FS \qquad (5.1)$$

$$C_{ij} = \max\left\{rm_{il}, d_{iL_{il}} + S_{il,L_{il},j} \cdot A_{il,L_{il},j}, C_{i-1,j} + t_{i-1,i}\right\} + \left(1 - A_{il,L_{il},j}\right) \cdot S_{il,L_{il},j} + p_{ilj} \quad j \in N, i > FS \qquad (5.2)$$

Estes cálculos devem ser feitos tarefa por tarefa para obter o tempo de finalização de cada tarefa em cada estação. Para cada tarefa $j$, o tempo de finalização na primeira estação ($i = FS$) é calculado através da equação (5.1), considerando tempos de liberação dos processadores ($rm_{il}$), tempo de saída da tarefa predecessora no processador designado ($d_{iL_{il}}$), tempos de preparação dependentes da sequência ($S_{il,L_{il},j}$) e tempos de processamento ($p_{ilj}$). Para as demais estações ($i > FS$), o tempo de finalização é calculado através da equação (5.2), na qual são consideradas informações acerca do tempo de finalização da tarefa $j$ na estação anterior ($C_{i-1,j}$) e o tempo de transporte requerido entre estações ($t_{i-1,i}$).

Note que uma vez conhecido o tempo de finalização da tarefa $j$ na estação $i$, $C_{ij}$, o tempo de saída da tarefa $j$ da estação $i-1$, $d_{i-1,j}$, pode ser facilmente calculado como:

$$d_{i-1,j} = C_{ij} - p_{ilj} - \left(1 - A_{il,L_{il},j}\right) \cdot S_{il,L_{il},j} - t_{i-1,i} \quad j \in N, i > FS \qquad (5.3)$$

Da mesma forma, o tempo de saída da tarefa $j$ da última estação do sistema, $i = LS$, é calculado através de:

$$d_{ij} = C_{ij} \quad j \in N, i = LS \qquad (5.4)$$

Finalmente, as regras de designação são definidas a seguir:

1. FAM: Esta regra designa a tarefa $j$ ao primeiro processador elegível que fique disponível em cada estação $i$. Este é o processador com menor tempo de saída de sua última tarefa programada, ou menor tempo de liberação se nenhuma tarefa tem sido programada ainda. Portanto:

$$T_{ij} = \arg\min_{l \in E_{ij}} \left\{\max\left(d_{iL_{il}}, rm_{il}\right)\right\} \quad j \in N, i \in M \qquad (5.5)$$

2. ECT: Para cada tarefa $j$ escolhe o processador elegível da estação $i$ que é capaz de completá-la no tempo mais cedo. Se a tarefa $j$ é designada ao

processador $l$ da estação $i$, o tempo no qual o processador $l$ finaliza a tarefa $j$ é denotado como $CT_{ilj}$. Assim:

$$T_{ij} = \arg\min_{l \in E_{ij}} \left\{ CT_{ilj} \right\} \quad j \in N, i \in M \tag{5.6}$$

Observe que dado um $l$ fixo, $CT_{ilj}$ pode ser calculado utilizando as equações (5.1)-(5.2).

As duas regras de designação utilizadas calculam um valor para cada processador elegível usando informação estática da instância (tempos de processamento, tempos de preparação, tempos de liberação, etc.) e informação dinâmica sobre o programa de produção parcial feito até o momento. O processador que minimiza tal valor é selecionado.

## 5.2. Heurística CDS

Esta heurística, proposta por Campbell, Dudek e Smith (1970), gera um conjunto de $m-1$ problemas artificiais de dois processadores em série, os quais são resolvidos utilizando o algoritmo de Johnson. A melhor sequência encontrada é a solução do problema original de $m$ processadores. Para aplicar esta heurística ao HFS uma adaptação simples foi considerada: O problema de $m$ estações com múltiplos processadores em paralelo é reduzido a um problema de $m$ estações com um único processador por estação, em que o tempo de processamento da tarefa $j$ no processador da estação $i$ é denotado como $APT_{ij} = \sum_{l \in E_{ij}} p_{ilj} \Big/ \left| E_{ij} \right|$, o qual representa o tempo de processamento médio (APT, *Average Processing Time*) da tarefa $j$ na estação $i$.

***Passo 1:*** Para cada tarefa $j$ calcular $APT_{ij}$.

***Passo 2:*** Faça $p = 1$.

***Passo 3:*** Calcular os tempos de processamento do $p^{th}$ problema artificial de dois processadores.

***Passo 3.1.:*** Para cada tarefa $j$ calcular $\theta_{j1}^{p} = \sum_{i=1}^{p} APT_{ij}$, o qual indica o tempo de processamento da tarefa $j$ no processador 1 para o $p^{th}$ problema artificial.

***Passo 3.2.:*** Para cada tarefa $j$ calcular $\theta_{j2}^{p} = \sum_{i=m+1-p}^{m} APT_{ij}$, o qual indica o tempo de processamento da tarefa $j$ no processador 2 para o $p^{th}$ problema artificial.

***Passo 4:*** Aplicar o algoritmo de Johnson ao $p^{th}$ problema artificial de dois processadores e salvar a sequência gerada, $\pi_p$.

***Passo 5:*** Se $p < m-1$, faça $p = p+1$ e vá ao ***Passo 3.*** Se $p = m-1$, vá ao ***Passo 6.***

***Passo 6:*** Calcular o *makespan* das $m-1$ sequências armazenadas, usando os dados originais do problema.

***Passo 7:*** Selecionar como melhor sequência, $\pi^{*}$, aquela com o menor *makespan* entre as $m-1$ sequências avaliadas.

O algoritmo de Johnson, utilizado para gerar a sequência de tarefas $\pi_p$, opera de forma simples: inicialmente, o conjunto de tarefas é particionado em dois subconjuntos. O primeiro subconjunto contém as tarefas cujo $\theta_{j1}^{p} < \theta_{j2}^{p}$. De forma similar, o segundo subconjunto contém as tarefas cujo $\theta_{j2}^{p} < \theta_{j1}^{p}$. Empates podem ser quebrados de forma arbitrária. As tarefas pertencentes ao primeiro subconjunto são sequenciadas segundo a regra do tempo de processamento mais curto (*SPT, Shortest Processing Time*), enquanto as tarefas pertencentes ao segundo subconjunto são sequenciadas segundo a regra do tempo de processamento mais longo (*LPT, Longest Processing Time*). A sequência final é obtida anexando a sequência *LPT* à sequencia *SPT*.

### 5.3. Heurística NEH

Esta heurística, proposta por Nawaz, Enscore e Ham (1983), é considerada largamente na literatura como a melhor heurística construtiva para resolver o *Flowshop* clássico e baseia-se na suposição que uma tarefa com maior tempo de processamento total deve ter maior prioridade que uma tarefa com menor tempo de processamento total. Inicialmente, as tarefas são organizadas em ordem decrescente do tempo de processamento total. Este ordenamento indica a sequência de inserção das tarefas na permutação a ser construída. Em seguida, as duas primeiras tarefas do ordenamento são selecionadas e sequenciadas nas duas formas possíveis, como se o problema tivesse somente duas tarefas a serem programadas. A melhor sequência obtida é utilizada como base para a inserção da terceira tarefa. Esta terceira tarefa é inserida nas três possíveis

posições da sequência que contém as duas primeiras tarefas e a melhor sequência, das três possíveis, é mantida para a inserção da quarta tarefa. Este processo é repetido até que todas as tarefas sejam sequenciadas.

Uma adaptação simples para HFS consiste em organizar as tarefas em ordem decrescente do tempo de processamento total médio (*TAPT*, *Total Average Processing Time*), denotado por $TAPT_j = \sum_{i \in M} APT_{ij}$ . Lembre-se que $APT_{ij}$ representa o tempo de processamento médio da tarefa $j$ na estação $i$ .

A seguir é apresentado o algoritmo passo a passo:

***Passo 1:*** Para cada tarefa $j$ calcular $TAPT_j = \sum_{i \in M} APT_{ij}$ .

***Passo 2:*** Organize as tarefas em uma lista em ordem decrescente de $TAPT_j$ .

***Passo 3:*** Selecione as primeiras duas tarefas da lista do ***Passo 2*** e determine a melhor sequência para estas, calculando o *makespan* das duas possíveis sequências. Faça $k = 3$.

***Passo 4:*** Selecione a tarefa na $k^{th}$ posição da lista gerada no ***Passo 2*** e encontre a melhor sequência, inserindo-a em todas as $k$ possíveis posições na sequência parcial obtida no passo anterior.

***Passo 5:*** Se $k = n$, PARE, caso contrário faça $k = k + 1$ e vá ao ***Passo 4.***

O número de sequências geradas pelo algoritmo é $\left(n(n+1)/2\right) - 1$, das quais $n$ são completas e as restantes são sequências parciais (NAWAZ; ENSCORE; HAM, 1983).

### 5.4. Heurística NEHKK1

A literatura indica que a força da heurística NEH reside principalmente na ordem de prioridade (Passo 2) de acordo com a qual as tarefas são selecionadas para serem programadas durante a fase de inserção (Passo 4). Por exemplo, Framinan, Leisten e Rajendran (2003) concluíram que a ordem de prioridade estabelecida pela heurística NEH é superior a 136 ordens diferentes examinadas em seu estudo. No entanto, Kalczynski e Kamburowski (2008) propuseram uma nova ordem de prioridade e um método simples para quebrar empates, os quais permitiram obter uma heurística capaz de superar à NEH no tradicional conjunto de problemas proposto por Taillard (1993). Note que a heurística NEH não especifica como quebrar empates no caso que duas sequências diferentes atinjam o mesmo *makespan*. Este estudo revelou que o rendimento da heurística NEH pode ser melhorado se a ordem de prioridade (Passo 1) é

baseada em tempos de Johnson em lugar dos tempos de processamento total. A seguir apresentasse, passo a passo, a heurística denominada NEHKK1:

***Passo 1:*** Para cada tarefa $j$ calcular $\hat{a}_j = \sum_{i \in M} \left[ (m-1)(m-2)/2 + m - i \right] \cdot APT_{ij}$ e $\hat{b}_j = \sum_{i \in M} \left[ (m-1)(m-2)/2 + i - 1 \right] \cdot APT_{ij}$

***Passo 2:*** Organize as tarefas em uma lista em ordem decrescente de $c_j$, tal que $c_j = \hat{a}_j$ se $\hat{a}_j \leq \hat{b}_j$ ou $c_j = \hat{b}_j$ se $\hat{a}_j > \hat{b}_j$.

***Passo 3:*** Selecione as primeiras duas tarefas da lista do ***Passo 2*** e determine a melhor sequência para estas, calculando o *makespan* das duas possíveis sequências. Faça $k = 3$.

***Passo 4:*** Seja $r$ a tarefa na $k^{th}$ posição da lista gerada no ***Passo 2.*** Encontre a melhor sequência, inserindo-a em todas as $k$ possíveis posições na sequência parcial obtida no passo anterior. No caso de empates, a melhor posição, $k^*$, é a primeira (última) posição na qual o empate é atingido se $\hat{a}_r \leq \hat{b}_r$ ($\hat{a}_r > \hat{b}_r$).

***Passo 5:*** Se $k = n$, PARE, caso contrário faça $k = k + 1$ e vá ao ***Passo 4.***

Note que a ordem de prioridade do Passo 2 resulta da aplicação do algoritmo de Johnson sobre os tempos $\hat{a}_j$ e $\hat{b}_j$ para cada tarefa $j$, considerando $\hat{a}_j$ e $\hat{b}_j$ como o tempo de processamento da tarefa $j$ no primeiro e segundo processador, respectivamente. As expressões para calcular $\hat{a}_j$ e $\hat{b}_j$ foram derivadas a partir de numerosas experiências de simulação que não foram reportadas por Kalczynski e Kamburowski (2008). Observe também que para utilizar a heurística NEHKK1 no HFS é necessário considerar o tempo de processamento médio de cada tarefa $j$ em cada estação $i$ para calcular os valores $\hat{a}_j$ e $\hat{b}_j$, tal como apresentado no Passo 1.

### 5.5. Heurística PFE

Ronconi (2004) testou várias heurísticas para resolver o *Flowshop* clássico com bloqueio, explorando características específicas do problema com o objetivo de gerar soluções de maior qualidade. A principal contribuição dessa pesquisa foi o desenvolvimento de uma nova heurística que resulta da combinação da heurística NEH com a heurística PF (MCCORMICK et al., 1989), a qual é capaz de obter melhores soluções que a heurística NEH.

A seguir mostra-se, passo a passo, a heurística PFE:

***Passo 1:*** Gere uma lista, ou sequência, de tarefas utilizando a heurística PF.

***Passo 2:*** Selecione as primeiras duas tarefas da lista do ***Passo 1*** e determine a melhor sequência para estas, calculando o *makespan* das duas possíveis sequências. Faça $k = 3$.

***Passo 3:*** Selecione a tarefa na $k^{th}$ posição da lista gerada no ***Passo 1*** e encontre a melhor sequência, inserindo-a em todas as $k$ possíveis posições na sequência parcial obtida no passo anterior.

***Passo 4:*** Se $k = n$, PARE, caso contrário faça $k = k + 1$ e vá ao ***Passo 3.***

Note que a heurística PFE simplesmente utiliza uma lista de prioridade diferente daquela utilizada pela NEH. Assim, a heurística PFE é diferente da heurística NEH somente no Passo 1, enquanto os demais passos são os mesmos.

Para aplicar a heurística PF ao HFS (Passo 1), como feito com a heurística CDS, o problema de $m$ estações com múltiplos processadores em paralelo é reduzido a um problema de $m$ estações com um único processador por estação, tal que o tempo de processamento da tarefa $j$ no processador da estação $i$ é denotado como $APT_{ij}$.

A heurística PF funciona da seguinte forma: a tarefa com o menor tempo de processamento total médio, $TAPT_j = \sum_{i \in M} APT_{ij}$, é selecionada como a primeira tarefa da sequência. Esta tarefa, denominada $j_1$, não encontra bloqueio e prossegue sem problemas de um processador para outro, gerando um "perfil". Este perfil é determinado pelo tempo de saída da tarefa $j_1$ do processador da estação $i$:

$$d_{ij_1} = \sum_{h=1}^{i} APT_{h,j_1}, \quad i \in M \tag{5.7}$$

Para determinar a segunda tarefa da sequência, cada tarefa ainda não programada é testada. Para cada tarefa candidata determina-se a quantidade de tempo que os processadores estão ociosos e a quantidade de tempo que a tarefa está bloqueada em um processador. O tempo de saída de uma tarefa candidata para a segunda posição da sequência, denominada $j_2$, pode ser computado recursivamente como:

$$d_{1,j_2} = \max\left(d_{1,j_1} + APT_{1,j_2}, d_{2,j_1}\right) \tag{5.8}$$

$$d_{i,j_2} = \max\left(d_{i-1,j_2} + APT_{i,j_2}, d_{i+1,j_1}\right), \quad i = 2,\ldots,m-1 \tag{5.9}$$

$$d_{m,j_2} = \max\left(d_{m-1,j_2}, d_{m,j_1}\right) + APT_{m,j_2} \tag{5.10}$$

O tempo improdutivo do processador $i$, isto é, o tempo que o processador esteve ocioso ou bloqueado, é calculado como $d_{i,j_2} - d_{i,j_1} - p_{i,j_2}$. Para cada tarefa candidata se calcula o tempo total que os processadores estiveram ociosos ou bloqueados, isto é, $\sum_{i \in M}\left(d_{i,j_2} - d_{i,j_1} - p_{i,j_2}\right)$. A tarefa candidata com a menor quantidade total de tempo improdutivo é selecionada como a segunda tarefa da sequência.

Depois de selecionar a tarefa que se "ajusta" melhor na segunda posição da sequência, o novo "perfil" (o tempo de saída desta tarefa de todos os processadores) é calculado e o procedimento é repetido. Das tarefas restantes, novamente aquela que se ajusta melhor é selecionada. Este processo continua até que todas as tarefas sejam programadas.

Note que nesta descrição da heurística PF, cada processador é considerado igualmente importante. No entanto, como é conhecido, o tempo improdutivo no processador gargalo é pior que o tempo improdutivo em qualquer outro processador, possivelmente menos ocupado. Assim, poderia ser apropriado multiplicar o tempo improdutivo de cada processador por um fator proporcional ao seu grau de ocupação. Quanto maior o grau de ocupação de um processador determinado, maior o peso do mesmo (PINEDO, 2008).

Por outro lado, Pan e Wang (2012) indicam que o tempo improdutivo dos processadores nas estações iniciais pode causar um maior atraso no começo do processamento das tarefas, quando comparado com o tempo improdutivo dos processadores nas últimas estações. Da mesma forma, o tempo improdutivo causado pelas primeiras tarefas da sequência poderia ter um maior efeito que o tempo improdutivo causado pelas ultima tarefas da sequência. Em consequência, os pesos deveriam ser maiores para os processadores nas estações iniciais e diminuir para os processadores nas últimas estações. Igualmente, os pesos deveriam ser maiores quando se está programando as primeiras posições da sequência e diminuir na medida em que se avança para as posições finais da sequência.

Liu e Reeves (2001) apresentam uma forma de calcular os pesos, satisfazendo os requerimentos anteriores. O peso do processador da estação $i$ é calculado como:

$$\omega_i = \frac{m}{\left(i + \frac{k(m-i)}{n-2}\right)}, \quad i \in M \tag{5.11}$$

Note que estes pesos diferenciam o efeito de processadores em diferentes estações, $i$, e tarefas em diferentes posições da sequência, $k$.

Os resultados apresentados por Ronconi (2004) incentivam a implementação da heurística PFE, pois espera-se que a exploração de características específicas do problema permita gerar soluções de melhor qualidade.

Finalmente, observe que a versão ponderada da heurística PFE obtém-se simplesmente utilizando a heurística PF ponderada no Passo 1. Isto facilita a total reutilização do código e, consequentemente ambas as versões serão testadas.

### 5.6. Heurística PF-NEH

Como mencionado anteriormente, Pan e Wang (2012) também destacam que a heurística PFE gera melhores resultados que a heurística NEH devido a que as características específicas do problema são exploradas. No entanto, na heurística PFE, a heurística PF é utilizada somente para gerar a lista de prioridade das tarefas (Passo 1) e, portanto as características específicas do problema exploradas pela heurística PF não são bem utilizadas no processo de inserção (Passo 3). Esta forma de combinar as heurísticas NEH e PF poderia decrescer a desempenho da heurística PFE. Para resolver este problema Pan e Wang (2012) propõem outro método para combinar as heurísticas NEH e PF:

Seja $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ a sequência gerada pela heurística PF. No método proposto, somente as últimas $\lambda$ tarefas da sequência $\beta$, isto é, as tarefas $\beta_{n-\lambda+1}, \beta_{n-\lambda+2}, \ldots, \beta_n$, passam através da fase de inserção da heurística NEH, enquanto a sequência parcial $\beta_1, \beta_2, \ldots, \beta_{n-\lambda}$ é utilizada como ponto inicial, como se estas tarefas já houvessem sido sequenciadas. Devido a que as posições relativas das tarefas na sequência parcial $\beta_1, \beta_2, \ldots, \beta_{n-\lambda}$ são geradas utilizando a heurística PF, este método de combinação das heurísticas NEH e PF explora de melhor forma as características do problema. A seguir mostra-se, passo a passo, a heurística PF-NEH:

***Passo 1:*** Gere a lista de tarefas $\beta = (\beta_1, \beta_2, \ldots, \beta_n)$ utilizando a heurística PF.

***Passo 2:*** Selecione as primeiras $n-\lambda$ tarefas da lista do ***Passo 1*** como ponto inicial, como se elas já houvessem sido sequenciadas. Faça $k = n-\lambda+1$.

***Passo 3:*** Selecione a tarefa na $k^{th}$ posição da lista gerada no ***Passo 1*** e encontre a melhor sequência, inserindo-a em todas as $k$ possíveis posições na sequência parcial obtida no passo anterior.

***Passo 4:*** Se $k = n$, PARE, caso contrário faça $k = k + 1$ e vá ao ***Passo 3.***

Os resultados mostrados por Pan e Wang (2012) indicam a superioridade da heurística PF-NEH em relação às heurísticas NEH e PFE, o qual motivou sua implementação nesta dissertação. Note que as heurísticas PFE e PF-NEH são similares e, portanto o código é totalmente reutilizável, o qual permite economizar tempo na implementação da heurística.

Observe-se que é necessário estabelecer o valor $\lambda$, o qual é um parâmetro requerido pela heurística. Com este objetivo, serão considerados três valores para o número de tarefas utilizadas como ponto inicial, $n - \lambda$, correspondente ao Passo 2 da heurística. Especificamente, $n - \lambda \in \{\lceil 0.25n \rceil, \lceil 0.50n \rceil, \lceil 0.75n \rceil\}$. Desta forma, $\lambda$ depende diretamente do número de tarefas a serem programadas e representa 75%, 50% e 25% das tarefas, respectivamente. A heurística PF-NEH é implementada para cada valor de $n - \lambda$ e, consequentemente é denominada PF-NEH$_{0.25}$, PF-NEH$_{0.50}$ e PF-NEH$_{0.75}$, respectivamente.

Pan e Wang (2012) generalizaram esta heurística, denominando-a PF-NEH($x$). Nesta heurística várias sequências são geradas e a melhor delas é utilizada como solução final. Para gerar várias sequências, os autores simplesmente trocam a primeira tarefa da sequência obtida pela heurística PF (Passo 1), pelas tarefas nas posições $2, 3, \ldots, x$, respectivamente. Adicionalmente, os autores aperfeiçoam mais a heurística incluindo um passo de busca local que visa melhorar a sequência gerada pela PF-NEH. Esta nova heurística foi denominada PF-NEH$_{LS}$($x$). Os resultados indicaram que estas estratégias permitem gerar melhores soluções que a heurística PF-NEH original. No entanto, nesta dissertação não é utilizada esta ideia, pois o objetivo é fazer uma comparação mais justa com as demais heurísticas, as quais não utilizam estas estratégias para serem mais competitivas. Observe também que o uso destas estratégias resulta em um aumento do custo computacional.

## 5.7. Avaliação Computacional

Nesta seção são analisados os resultados da avaliação computacional das heurísticas através da técnica de análise de variância (ANOVA, *Analysis of Variance*). Este procedimento está projetado para construir um modelo estatístico que descreve o impacto de dois ou mais fatores sobre uma determinada variável dependente. De igual forma, o procedimento permite determinar se há diferenças significativas entre as

médias de diferentes níveis dos fatores, assim como estabelecer se existe interação entre os fatores.

Observe que a ANOVA não identifica quais médias são diferentes, mas somente indica que nem todas são iguais. Assim, para determinar quais médias são estatisticamente diferentes de outras é preciso utilizar métodos de comparações múltiplas. Especificamente, será utilizado o método da diferença mínima significativa de Fisher, denominado simplesmente como LSD (*Least Significant Difference*), o qual determina a diferença mínima entre duas médias que pode ser declarada como estatisticamente significativa. Montgomery (2005) indica este método é um dos mais eficacez e potentes para detectar diferenças reais entre médias, o qual justifica sua escolha.

Por outro lado, é preciso destacar que a ANOVA pressupõe que os resíduos do modelo estatístico seguem uma distribuição normal e independente com média zero e variância $\sigma^2$ constante, mas desconhecida. Assim, estes pressupostos devem ser verificados através das técnicas estatísticas adequadas. Informação mais detalhada acerca da ANOVA, métodos de comparações múltiplas e testes estatísticos para a verificação de pressupostos podem ser encontrados nos livros de Montgomery e Runger (2003) e Montgomery (2005).

De forma preliminar, alguns testes foram elaborados com o objetivo de estudar o desempenho das duas regras de designação descritas na seção 5.1. Nestes testes, as heurísticas CDS, NEH e NEHKK1 foram implementadas para cada regra de designação, obtendo-se seis heurísticas: $CDS_{FAM}$, $CDS_{ECT}$, $NEH_{FAM}$, $NEH_{ECT}$, $NEHKK1_{FAM}$ e $NEHKK1_{ECT}$. Os resultados deste teste, mostrados no Apêndice A, indicam que a regra de designação ECT é superior à regra FAM. A razão deste resultado é simples: a regra ECT considera mais informações que a regra FAM para tomar a decisão de designar tarefas a processadores. Evidentemente, ECT visa à minimização do tempo de finalização de cada tarefa $j$ em cada estação $i$, enquanto a regra FAM simplesmente visa minimizar o tempo ocioso dos processadores de cada estação.

Esta abordagem pode ser de grande utilidade em problemas com processadores idênticos e tempos de preparação não dependentes da sequência, porém em problemas mais complexos, com processadores não relacionados e tempos de preparação dependentes da sequência, o primeiro processador disponível não necessariamente é o processador que pode finalizar a tarefa no tempo mais cedo. Assim,

designar uma tarefa ao primeiro processador disponível, sem considerar informações relacionadas aos tempos de processamento e preparação entre as tarefas envolvidas, pode conduzir a tempos de finalização excessivamente longos.

Baseados nestes resultados, as demais heurísticas (PFE e PF-NEH) são implementadas somente com a regra de designação ECT. Em qualquer caso, os resultados atingidos pelas heurísticas $CDS_{FAM}$, $NEH_{FAM}$ e $NEHKK1_{FAM}$ serão apresentados com propósitos comparativos.

A seguir listam-se as heurísticas implementadas:

- Heurística CDS com regra de designação FAM: $CDS_{FAM}$
- Heurística CDS com regra de designação ECT: $CDS_{ECT}$
- Heurística NEH com regra de designação FAM: $NEH_{FAM}$
- Heurística NEH com regra de designação ECT: $NEH_{ECT}$
- Heurística NEHKK1 com regra de designação FAM: $NEHKK1_{FAM}$
- Heurística NEHKK1 com regra de designação ECT: $NEHKK1_{ECT}$
- Heurística PFE com regra de designação ECT: PFE
- Heurística PFE ponderada com regra de designação ECT: *w*PFE
- Heurísticas PF-NEH com regra de designação ECT: PF-$NEH_{0.25}$, PF-$NEH_{0.50}$ e PF-$NEH_{0.75}$.

Todas as heurísticas foram implementadas na linguagem C++ e executadas no mesmo *notebook* utilizado para a avaliação do modelo de programação inteira mista da seção 4.1.

A avaliação computacional das heurísticas foi dividida em duas partes, cada uma correspondente a um conjunto diferente de instâncias. A seção 5.7.1 apresenta a avaliação computacional utilizando o mesmo conjunto de instâncias utilizadas na avaliação do modelo de programação inteira mista (ver seção 4.2). Por outro lado, na seção 5.7.2 apresenta-se um novo conjunto de instâncias de maior tamanho e avalia-se o desempenho das diferentes heurísticas sobre este novo conjunto.

### 5.7.1. Avaliação Conjunto de Teste A

Nesta seção é apresentada a avaliação computacional das heurísticas, utilizando o mesmo conjunto de instâncias utilizadas na avaliação computacional do modelo de programação inteira mista, conformado por 1.728 problemas.

Para medir o desempenho das heurísticas, a percentagem de desvio relativo (RPD, *Relative Percentage Deviation*) sobre a melhor solução conhecida de cada

instância é calculada. Lembre-se que a melhor solução conhecida para cada instância é determinada através modelo de programação inteira mista da seção 4.1:

$$RPD = \left( \frac{Heu_{Sol} - MIP_{Best}}{MIP_{Best}} \right) \cdot 100$$

em que $Heu_{Sol}$ é a solução encontrada por uma determinada heurística e $MIP_{Best}$ é a melhor solução encontrada pelo modelo de programação inteira mista. Note que $MIP_{Best}$ não é conhecido para as 51 instâncias nas quais o modelo de programação inteira não foi capaz de encontrar uma solução factível. Neste caso, é evidente que as heurísticas são vantajosas, pois em curtos tempos computacionais podem gerar soluções factíveis, embora não seja possível estabelecer nada concreto acerca da qualidade destas soluções.

A Tabela 5.1 mostra o RPD médio das heurísticas, considerando somente aquelas instâncias para as quais uma solução ótima é conhecida. Note que os resultados são consolidados por cada combinação de $n$ e $m$. Assim, cada célula da Tabela 5.1 representa a média sobre todos os demais fatores.

Como esperado a heurística CDS apresenta o pior resultado com um RPD médio de 22,5 e 16,613, para as regras FAM e ECT. Em termos gerais, as heurísticas que utilizam a regra de designação FAM apresentam piores resultados que seus correspondentes pares com a regra ECT. Isto confirma os resultados do teste preliminar descrito no Apêndice A.

Observe que a heurística NEHKK1 é superior à heurística NEH quando ambas utilizam a regra ECT. No entanto, quando implementadas com a regra FAM as diferenças entre as heurísticas são insignificantes. Esta situação indica que as diferenças entre o desempenho das heurísticas NEH e NEHKK1 não são significativas, contradizendo os resultados de Kalczynski e Kamburowski (2008), os quais indicaram que NEHKK1 supera à NEH para qualquer tamanho de instância.

De igual forma, é possível observar que a heurística *w*PFE é superior à heurística PFE, embora a diferença seja desprezível. Isto que indica que os pesos utilizados na *w*PFE para ponderar os processadores tem algum efeito no resultado final, mas não o suficiente para gerar resultados significativamente melhores. Note também que ambas as heurísticas apresentam um desempenho inferior ao mostrado pelas heurísticas $NEH_{ECT}$ e $NEHKK1_{ECT.}$

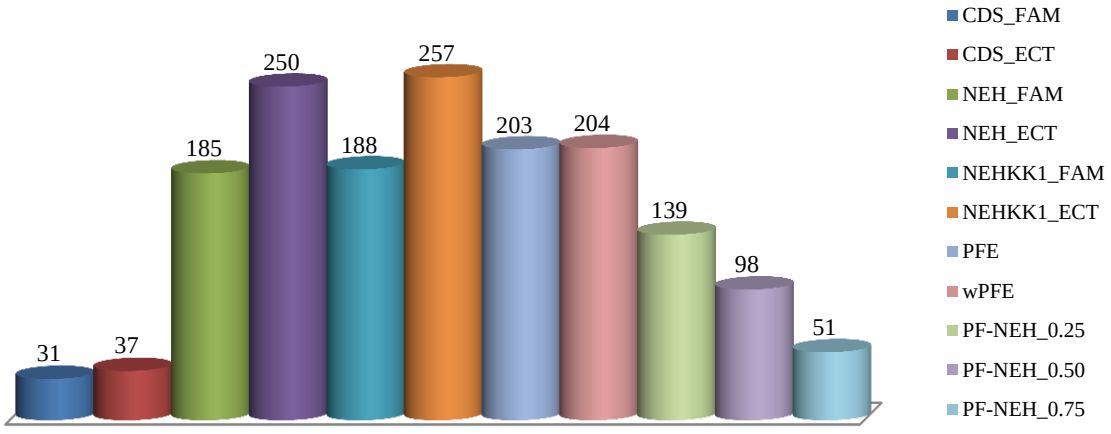
**Tabela 5.1 – RPD médio sobre o conjunto de teste A (Instâncias com solução ótima conhecida)**

| $n \times m$ | $CDS_{FAM}$ | $CDS_{ECT}$ | $NEH_{FAM}$ | $NEH_{ECT}$ | $NEHKK1_{FAM}$ | $NEHKK1_{ECT}$ | PFE | *w*PFE | $PF\text{-}NEH_{0.25}$ | $PF\text{-}NEH_{0.50}$ | $PF\text{-}NEH_{0.75}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5×3 | 21,100 | 15,227 | 9,779 | 3,443 | 9,643 | 3,240 | 3,464 | 3,464 | 5,626 | 6,463 | 8,791 |
| 5×5 | 18,490 | 12,804 | 9,348 | 3,862 | 8,963 | 3,865 | 4,727 | 4,727 | 6,620 | 7,947 | 11,296 |
| 5×7 | 14,313 | 10,430 | 8,508 | 4,185 | 7,907 | 4,278 | 5,229 | 5,229 | 6,757 | 8,704 | 11,104 |
| 7×3 | 25,186 | 18,105 | 10,573 | 4,607 | 11,224 | 4,425 | 5,720 | 5,622 | 6,809 | 7,822 | 13,775 |
| 7×5 | 21,906 | 16,063 | 10,998 | 6,047 | 10,923 | 5,553 | 6,630 | 6,564 | 6,898 | 9,250 | 16,187 |
| 7×7 | 19,930 | 14,627 | 10,539 | 6,200 | 10,933 | 6,572 | 7,477 | 7,426 | 8,659 | 10,841 | 17,518 |
| 9×3 | 32,567 | 23,104 | 15,784 | 6,660 | 15,826 | 6,027 | 8,649 | 8,827 | 9,192 | 10,926 | 15,287 |
| 9×5 | 27,162 | 19,734 | 12,823 | 7,280 | 13,131 | 7,020 | 8,784 | 8,603 | 9,414 | 11,802 | 16,474 |
| 9×7 | 18,225 | 16,795 | 8,211 | 6,903 | 8,888 | 7,191 | 8,188 | 8,288 | 9,616 | 12,317 | 18,417 |
| 11×3 | 29,110 | 25,162 | 9,422 | 6,411 | 9,473 | 5,639 | 8,784 | 8,726 | 8,723 | 12,647 | 18,366 |
| 11×5 | 23,406 | 22,877 | 4,882 | 5,158 | 6,234 | 5,284 | 10,272 | 10,377 | 9,578 | 10,806 | 18,207 |
| 11×7 | 18,219 | 18,219 | 4,946 | 4,946 | 4,112 | 5,760 | 7,660 | 7,660 | 4,356 | 3,738 | 14,099 |
| **Média** | **22,500** | **16,613** | **10,642** | **5,241** | **10,684** | **5,068** | **6,398** | **6,383** | **7,505** | **9,306** | **14,003** |

Analisando as heurísticas PF-NEH, observa-se que os melhores resultados são atingidos pela heurística PF-NEH$_{0.25}$, na qual o primeiro 25% das tarefas são utilizadas como ponto inicial e o restante 75% passam pela fase de inserção da heurística NEH. Note que, na medida em que diminui a percentagem de tarefas que passam pela fase de inserção, o desempenho da heurística também tende a diminuir. Isto indica que a potência da heurística PF-NEH encontra-se, principalmente, no uso da fase de inserção da heurística NEH e não no uso de heurística PF. Note, por exemplo, que na heurística PFE todas as tarefas devem passar pela fase de inserção e, consequentemente, os resultados obtidos são melhores.

A Figura 5.1 ilustra o número de instâncias que cada heurística resolveu até otimalidade. Esta figura permite confirmar todas as conclusões expostas anteriormente. Por exemplo, as heurísticas CDS$_{FAM}$ e CDS$_{ECT}$ apresentam o mais baixo desempenho, obtendo o menor número de soluções ótimas. As heurísticas NEH$_{ECT}$ e NEHKK1$_{ECT}$ encontram o maior número de soluções ótimas, 250 e 257, respectivamente. O desempenho das heurísticas PFE e *w*PFE não varia significativamente, obtendo 203 e 204 soluções ótimas, respectivamente. Finalmente, as heurísticas PF-NEH obtêm menor número de soluções ótimas que a heurística PFE e seu desempenho tende a diminuir quando o número de tarefas que passam pela fase de inserção diminui, caindo de 139 a 51 soluções ótimas, no caso das heurísticas PF-NEH$_{0.25}$ e PF-NEH$_{0.75}$, respectivamente.

**Figura 5.1 – Número de soluções ótimas para cada heurística**




Por outro lado, uma vantagem dos métodos heurísticos em relação a métodos exatos é o menor esforço computacional requerido para obter uma solução para o problema estudado. Neste caso, os tempos computacionais de todas as heurísticas foram da ordem de microssegundos e, portanto não são apresentados aqui.

A seguir analisam-se os resultados do subconjunto de instâncias para as quais somente uma solução factível é conhecida. A Tabela 5.2 mostra o RPD médio das heurísticas, considerando somente este subconjunto de instâncias. Os resultados são consolidados por cada combinação de $n$ e $m$, de tal forma que cada célula da Tabela 5.2 representa a média sobre todos os demais fatores.

Os resultados da Tabela 5.2 evidenciam que para vários tamanhos de instância as heurísticas são capazes de obter melhores soluções que aquelas encontradas pelo *solver*, tal como descrito pelos valores de RPD médio negativos. Em termos gerais, os resultados desta tabela confirmam as conclusões expostas anteriormente.

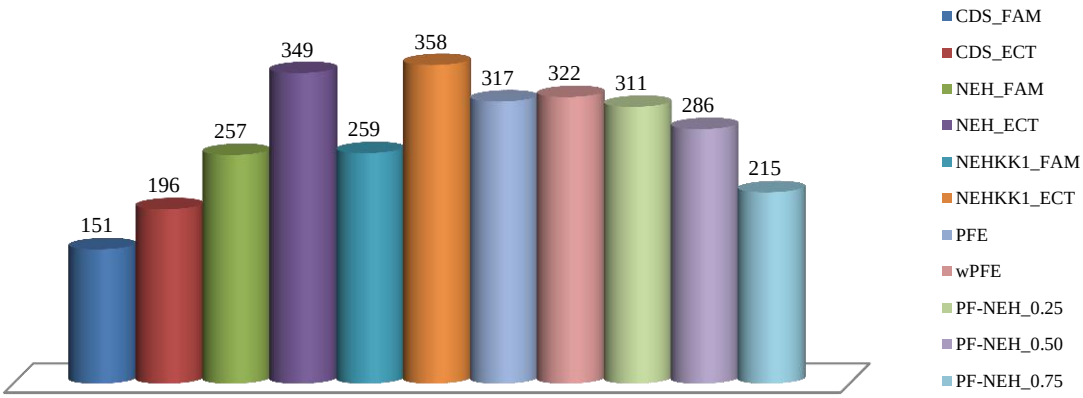
As heurísticas $CDS_{FAM}$ e $CDS_{ECT}$ mostram o desempenho mais baixo, com RPD médio de 12,409 e 4,363, respectivamente. Novamente, os melhores resultados são atingidos pelas heurísticas $NEH_{ECT}$ e $NEHKK1_{ECT}$, as quais obtêm melhores soluções para quase todos os tamanhos de instância. Quando implementadas com a regra FAM, os desempenhos de ambas as heurísticas é praticamente igual. As heurísticas PFE e *w*PFE também são capazes de encontrar melhores soluções para quase todos os tamanhos de instâncias, porém continuam sendo inferiores às heurísticas $NEH_{ECT}$ e $NEHKK1_{ECT}$. Quando comparadas entre elas, a heurística *w*PFE é continua sendo ligeiramente superior à heurística PFE. Finalmente, a heurística PF-$NEH_{0.25}$ apresenta os melhores resultados entre as heurísticas do tipo PF-NEH, enquanto PF-$NEH_{0.75}$ apresenta o desempenho mais baixo. Note que, novamente, todas as heurísticas do tipo PF-NEH são superadas pelas heurísticas PFE e *w*PFE.

A Figura 5.2 ilustra o número de instâncias para as quais as heurísticas geraram uma melhor solução que aquela encontrada através do modelo de programação matemática. Em conjunto, todas estas observações indicam que, mesmo para instâncias de entre 9 e 11 tarefas, as heurísticas podem fornecer soluções de boa qualidade em tempos computacionais mais curtos que o requerido pelo *solver*. Portanto, um compromisso entre a qualidade da solução desejada e o tempo computacional disponível deve ser atingido.

**Tabela 5.2 – RPD médio sobre o conjunto de teste A (Instâncias com solução factível conhecida)**

| $n \times m$ | $CDS_{FAM}$ | $CDS_{ECT}$ | $NEH_{FAM}$ | $NEH_{ECT}$ | $NEHKK1_{FAM}$ | $NEHKK1_{ECT}$ | PFE | *w*PFE | $PF\text{-}NEH_{0.25}$ | $PF\text{-}NEH_{0.50}$ | $PF\text{-}NEH_{0.75}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $7 \times 5$ | 13,286 | 10,962 | 2,533 | -2,674 | 4,072 | -2,510 | -0,536 | -0,536 | 1,149 | -0,211 | 7,186 |
| $7 \times 7$ | 4,875 | 3,843 | -2,095 | -2,881 | -2,320 | -3,599 | -3,920 | -4,409 | -3,042 | -1,076 | 2,599 |
| $9 \times 3$ | 27,674 | 13,819 | 15,225 | 1,867 | 13,840 | 4,303 | 4,713 | 4,713 | 5,961 | 2,817 | 9,570 |
| $9 \times 5$ | 14,331 | 8,404 | 1,901 | -2,868 | 2,477 | -3,423 | -3,024 | -3,027 | -2,530 | -1,122 | 2,892 |
| $9 \times 7$ | 8,623 | 2,235 | -0,221 | -6,430 | -0,125 | -6,414 | -5,311 | -5,200 | -4,688 | -2,806 | 1,253 |
| $11 \times 3$ | 31,780 | 16,766 | 12,813 | -0,605 | 13,305 | -1,100 | 1,279 | 1,031 | 2,151 | 3,600 | 8,683 |
| $11 \times 5$ | 8,437 | 0,850 | -2,776 | -9,613 | -3,427 | -10,144 | -8,827 | -8,937 | -8,900 | -6,603 | -1,655 |
| $11 \times 7$ | 4,167 | -2,495 | -6,607 | -12,400 | -6,597 | -12,596 | -11,455 | -11,329 | -10,632 | -8,457 | -3,052 |
| **Média** | **12,409** | **4,363** | **0,417** | **-6,812** | **0,427** | **-7,111** | **-5,874** | **-5,905** | **-5,320** | **-3,507** | **1,274** |

**Figura 5.2 – Número de melhores soluções factíveis para cada heurística**

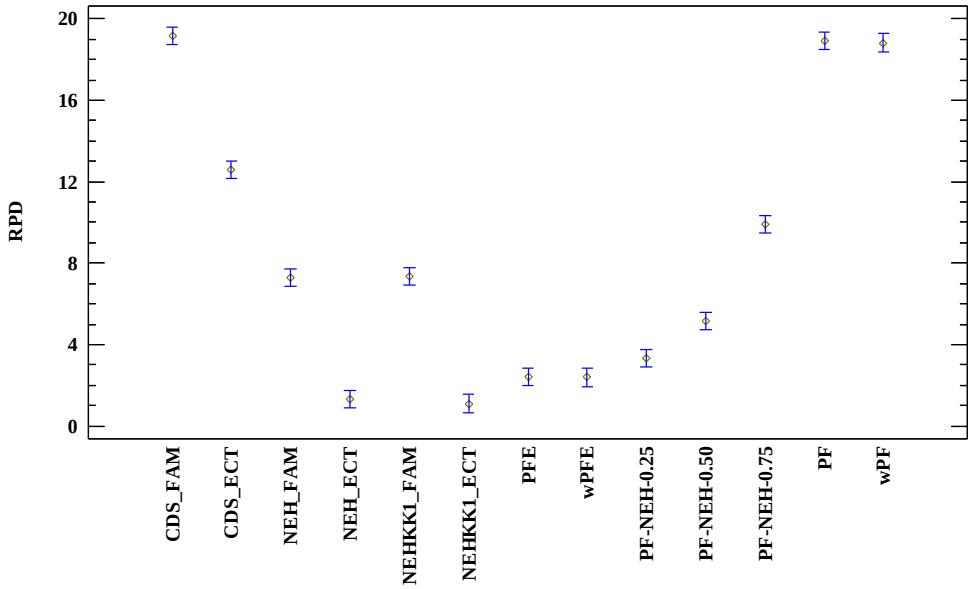



Para finalizar, uma ANOVA multifatorial é apresentada com o objetivo de identificar se as diferenças existentes entre o RPD médio das heurísticas são estatisticamente significativas. Neste teste, o número de tarefas ($n$), o número de estações ($m$), o número de processadores paralelos por estação ($m_i$) e o número de *buffers* paralelos por estação ($b_i$) e as heurísticas são considerados como fatores, enquanto o RPD representa a variável de resposta. Desta forma, a ANOVA considera o efeito causado pelo tamanho das instâncias. Os pressupostos da ANOVA (normalidade, igualdade de variância e independência de resíduos) foram validados através das técnicas adequadas. A tabela ANOVA é mostrada no Apêndice B.

Os resultados da ANOVA são apresentados graficamente através da Figura 5.3, a qual mostra as médias e intervalos de confiança de 95% de Fisher LSD para cada heurística. Estes intervalos de confiança servem para determinar se dois valores são estatisticamente diferentes. Intervalos que não se sobrepõem verticalmente indicam que há diferenças estatisticamente significativas entre algumas heurísticas. Note que o teste incluiu as heurísticas PF e *w*PF com propósitos comparativos.

Todos os resultados apresentados até o momento são claramente resumidos pela Figura 5.3. A heurística $CDS_{FAM}$, junto com as heurísticas PF e *w*PF, obtêm os piores resultados. Note que entre estas três heurísticas não há diferenças estatisticamente significativas.

**Figura 5.3 – Gráfico de médias e intervalos de confiança de 95% de Fisher para as heurísticas - Conjunto de teste A**




Os melhores resultados são obtidos pelas heurísticas $NEHKK1_{ECT}$ e $NEH_{ECT}$, respectivamente. Note que $NEHKK1_{ECT}$ obtém resultados levemente melhores que a heurística $NEH_{ECT}$, porém o gráfico de médias permite estabelecer que tal diferença não é estatisticamente significativa. Igualmente, observa-se que PFE e *w*PFE apresentam um desempenho menor que $NEHKK1_{ECT}$ e $NEH_{ECT}$, mas são estatisticamente iguais entre si. Por fim, entre as heurísticas do tipo PF-NEH, a heurística $PF\text{-}NEH_{0.25}$ é claramente superior às demais.

A partir de todos os resultados apresentados ao longo desta seção, conclui-se que as heurísticas $NEHKK1_{ECT}$ e $NEH_{ECT}$ foram as melhores. Porém, quando comparadas entre elas é possível estabelecer que não existe diferenças estatisticamente significativas. Portanto, ambas as heurísticas podem potencialmente serem utilizadas sem esperar diferencias significantes entre os resultados atingidos.

### 5.7.2. Avaliação Conjunto de Teste B

Nesta seção é apresentada a avaliação computacional das heurísticas, utilizando um novo conjunto de instâncias de maior tamanho. Para maior clareza, este novo conjunto de instâncias será denominado conjunto de teste B. Os fatores considerados para a geração deste conjunto são apresentados na Tabela 5.3. Note que este conjunto de instâncias é diferente do conjunto de teste A somente pelos valores dos fatores $n$, $m$, $m_i$ e $b_i$.

**Tabela 5.3 – Fatores do conjunto de teste B**

| Fatores | Símbolo | Valores |
|---|---|---|
| Número de tarefas | $n$ | 50,100 |
| Número de estações | $m$ | 7, 9, 11 |
| Número de processadores paralelos em estações de processamento | $m_i$ | 3, 4, $U[1,4]$ |
| Número de processadores paralelos em estações *buffer* | $b_i$ | 3, 4, $U[1,4]$ |
| Probabilidade de um processador ser elegível | $PE_{ij}$ | 0.50, 1 |
| Distribuição dos tempos de preparação | $DS_{iljk}$ | $U[25,74], U[75,125]$ |
| Probabilidade do tempo de preparação ser antecipatório | $PA_{iljk}$ | 0, 0.50, 1 |

Da mesma forma que o conjunto de teste A, os valores dos parâmetros mostrados na Tabela 5.3 foram escolhidos com o objetivo de gerar instâncias que representem diferentes cenários para o problema estudado.

O número de combinações destes fatores é $2 \cdot 3 \cdot 3 \cdot 3 \cdot 2 \cdot 2 \cdot 3$ ou $2^3 \cdot 3^4 = 648$. Para cada combinação foram feitas três réplicas, pelo qual se tem 1.944 instâncias. Como o modelo de programação inteira mista é computacionalmente inviável para resolver este conjunto de instâncias, as heurísticas serão comparadas entre si. Deste modo, para este conjunto de teste, o RPD calcula-se como:

$$RPD = \left( \frac{Heu_{Sol} - Min_{Sol}}{Min_{Sol}} \right) \cdot 100$$

em que $Min_{Sol}$ é a melhor solução encontrada por qualquer uma das heurísticas testadas para uma instância dada.

A Tabela 5.4 mostra o RPD médio das heurísticas para o conjunto de teste B. Os resultados são consolidados para cada combinação de $n$ e $m$, de tal forma que cada célula da Tabela 5.4 representa a média sobre todos os demais fatores.

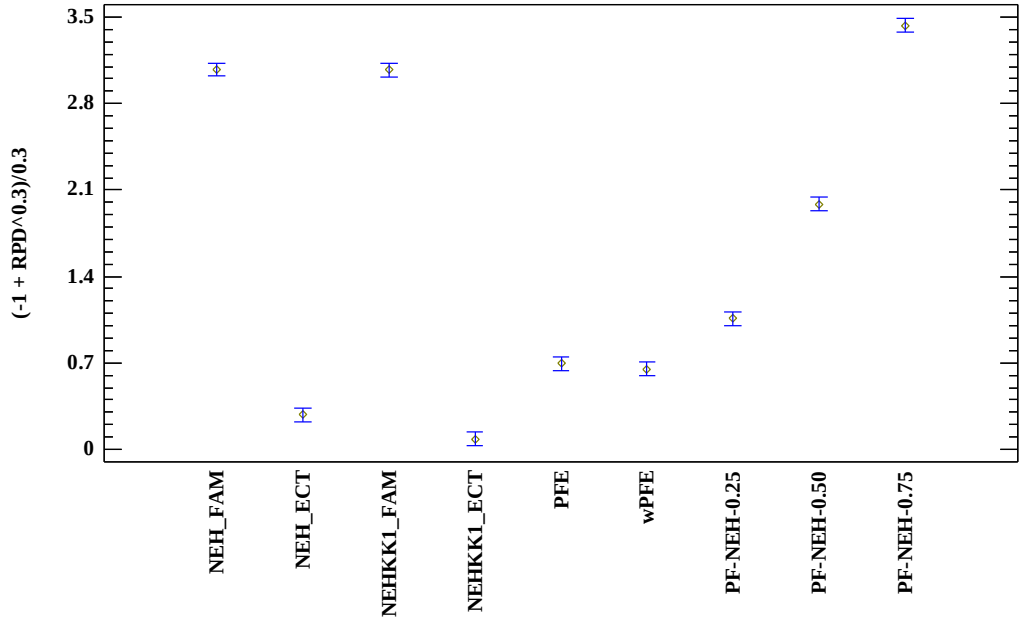
Como esperado, os resultados para o conjunto de teste B tem o mesmo comportamento observado no conjunto de teste A. As heurísticas $CDS_{FAM}$ e $CDS_{ECT}$ obtêm os piores resultados, enquanto as heurísticas $NEHKK1_{ECT}$ e $NEH_{ECT}$ são as melhores. Note que esta vez a diferença entre $NEHKK1_{ECT}$ e $NEH_{ECT}$ é significativa, tal como confirmado posteriormente através da ANOVA. De igual forma, o desempenho das heurísticas PFE e *w*PFE não apresenta diferenças significativas, enquanto a heurística PF-$NEH_{0.25}$ atinge os melhores resultados entre as heurísticas do tipo PF-NEH.

**Tabela 5.4 – RPD médio sobre o conjunto de teste B**

| $n \times m$ | $CDS_{FAM}$ | $CDS_{ECT}$ | $NEH_{FAM}$ | $NEH_{ECT}$ | $NEHKK1_{FAM}$ | $NEHKK1_{ECT}$ | PFE | *w*PFE | $PF\text{-}NEH_{0.25}$ | $PF\text{-}NEH_{0.50}$ | $PF\text{-}NEH_{0.75}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 50×7 | 38,729 | 22,142 | 14,264 | 1,826 | 14,197 | 1,261 | 2,332 | 2,470 | 3,028 | 5,150 | 11,464 |
| 50×9 | 34,101 | 18,698 | 13,046 | 1,700 | 13,111 | 1,490 | 2,488 | 2,222 | 2,820 | 4,982 | 10,402 |
| 50×11 | 33,693 | 20,874 | 10,806 | 1,559 | 10,736 | 1,363 | 2,122 | 2,261 | 2,928 | 5,382 | 11,613 |
| 100×7 | 44,438 | 26,318 | 14,038 | 1,261 | 13,994 | 0,967 | 2,108 | 1,970 | 2,479 | 4,655 | 10,444 |
| 100×9 | 41,279 | 24,195 | 13,949 | 1,199 | 13,782 | 1,002 | 1,957 | 1,772 | 2,468 | 4,371 | 10,223 |
| 100×11 | 39,717 | 24,699 | 12,769 | 1,255 | 12,579 | 0,975 | 2,013 | 1,966 | 2,525 | 4,547 | 10,469 |
| **Média** | **38,660** | **22,821** | **13,145** | **1,467** | **13,067** | **1,176** | **2,170** | **2,110** | **2,708** | **4,848** | **10,769** |

Para verificar se as diferenças entre as heurísticas são significativas, uma nova ANOVA é realizada. Lembre-se que para aplicar este teste os pressupostos de normalidade, igualdade de variância e independência de resíduos devem ser validados através das técnicas adequadas. A tabela completa da ANOVA é apresentada no Apêndice B. Os resultados da ANOVA são ilustrados através do gráfico de médias e intervalos de confiança de 95% de Fisher LSD, apresentados na Figura 5.4.

**Figura 5.4 – Gráfico de médias e intervalos de confiança de 95% de Fisher para as heurísticas - Conjunto de teste B**




Observe que na Figura 5.4 a variável de resposta é definida como $\left(-1+RPD^{0.3}\right)/0.3$. Este tipo de transformação da variável de resposta é frequentemente utilizado para estabilizar a variância e melhorar o ajuste dos resíduos à distribuição normal (BOX; COX, 1964; MONTGOMERY, 2005). Note também que, devido ao baixo desempenho mostrado, as heurísticas $CDS_{FAM}$ e $CDS_{ECT}$ não são mais consideradas e, portanto são excluídas da ANOVA. Da mesma forma, as heurísticas PF e *w*PF não são incluídas para mais análises, dado o baixo desempenho mostrado na seção 5.7.1.

Como esperado, a ANOVA confirma que a diferença entre $NEHKK1_{ECT}$ e $NEH_{ECT}$ é significativa. Portanto, para o conjunto de teste B, a heurística $NEHKK1_{ECT}$ obtém os melhores resultados, seguido da heurística $NEH_{ECT}$. Por outro lado, as diferenças entre as heurísticas PF e *w*PFE não são estatisticamente significativas, embora a média da heurística *w*PFE é levemente melhor. Por fim, os resultados das

heurísticas tipo PF-NEH não variam: PF-$NEH_{0.25}$ e PF-$NEH_{0.75}$ atingem os melhores e piores resultados entre este tipo de heurística, respectivamente.

É preciso destacar que estes resultados somente indicam que a heurística $NEHKK1_{ECT}$ está atingindo as melhores soluções para as instâncias do conjunto de teste B, pois não temos indicativo algum de quão longe estas soluções estão do ótimo real.

Para finalizar, a Tabela 5.5 apresenta o tempo computacional médio requerido por cada heurística para resolver o conjunto de teste B. Os resultados são consolidados para cada combinação de $n$ e $m$.

**Tabela 5.5 – Tempo computacional médio das heurísticas (segundos) - Conjunto de teste B**

| $n \times m$ | $NEH_{FAM}$ | $NEH_{ECT}$ | $NEHKK1_{FAM}$ | $NEHKK1_{ECT}$ | PFE | *w*PFE | PF-$NEH_{0,25}$ | PF-$NEH_{0,50}$ | PF-$NEH_{0,75}$ |
|---|---|---|---|---|---|---|---|---|---|
| $50 \times 7$ | 0,047 | 0,066 | 0,042 | 0,062 | 0,062 | 0,063 | 0,062 | 0,054 | 0,037 |
| $50 \times 9$ | 0,058 | 0,082 | 0,054 | 0,080 | 0,080 | 0,080 | 0,078 | 0,069 | 0,048 |
| $50 \times 11$ | 0,070 | 0,099 | 0,068 | 0,098 | 0,097 | 0,098 | 0,096 | 0,085 | 0,058 |
| $100 \times 7$ | 0,328 | 0,481 | 0,329 | 0,541 | 0,480 | 0,487 | 0,470 | 0,419 | 0,292 |
| $100 \times 9$ | 0,446 | 0,686 | 0,449 | 0,801 | 0,680 | 0,683 | 0,663 | 0,594 | 0,411 |
| $100 \times 11$ | 0,555 | 0,850 | 0,560 | 0,999 | 0,841 | 0,854 | 0,820 | 0,770 | 0,518 |
| **Média** | 0,251 | 0,377 | 0,251 | 0,430 | 0,373 | 0,378 | 0,365 | 0,332 | 0,227 |

Como observado, o tempo computacional não é relevante, pois todas as heurísticas requerem pouco esforço computacional. A heurística $NEHKK1_{ECT}$ tem o maior tempo computacional médio, mas este tempo é compensado pela qualidade das soluções geradas. Note que o tempo computacional da heurística PF-NEH tende a diminuir na medida em que o número de tarefas que passa pela fase de inserção diminui. Este comportamento é esperado, pois o número de sequências parciais que são geradas e avaliadas na fase de inserção diminui, o que implica uma redução do esforço computacional requerido.

Considerando juntamente os resultados dos dois conjuntos de testes analisados neste capítulo, é possível concluir que a heurística NEHKK1 é superior as demais heurísticas, obtendo soluções de maior qualidade com um custo computacional levemente maior. Também é possível concluir que a eficácia das heurísticas que combinam PF e NEH deve-se, principalmente, à fase de inserção da NEH e não ao uso da heurística PF. Em qualquer caso, é justo destacar que o HFS é um sistema de produção mais complexo que o *Flowshop* clássico e, portanto, é normal que o desempenho de heurísticas, inicialmente projetadas para *Flowshop*, diminua quando

utilizadas em ambientes de produção mais complexos, os quais envolvem um maior número de variáveis e decisões a serem tomadas.

# 6. Meta-heurística: Busca Local Iterada (ILS)

O termo meta-heurística, introduzido no artigo seminal de Glover (1986) acerca de Busca Tabu, deriva da composição de duas palavras. A palavra "heurística" tem sua origem na palavra grega "*heuriskein*", a qual significa a arte de descobrir novas estratégias para resolver problemas. O sufixo "*meta*", também de origem grego, significa metodologia de nível superior (BLUM; ROLI, 2003; TALBI, 2009). Desta forma, uma meta-heurística é definida como um processo mestre de alto nível que orienta e modifica outras heurísticas com o objetivo de explorar soluções além da otimalidade local.

Formalmente, Glover e Kochenberger (2003) indicam que meta-heurísticas são métodos de solução que orquestram uma interação entre procedimentos de melhoria local e estratégias de nível superior para criar um processo capaz de escapar de mínimos locais e realizar uma busca robusta do espaço de solução.

Numerosas ferramentas e mecanismos que surgiram a partir da criação de meta-heurísticas provaram ser extraordinariamente eficazes, tornando estes métodos como os preferidos para tentar resolver vários tipos de problemas complexos, particularmente aqueles de natureza combinatória. Diferentemente dos métodos exatos, as meta-heurísticas não são capazes de garantir a otimalidade das soluções que encontram. Porém, em problemas do mundo real, os quais se caracterizam por terem altos níveis de complexidade, os métodos exatos frequentemente se mostram incapazes de encontrar soluções de igual, ou melhor, qualidade às obtidas por meta-heurísticas.

Na prática, as meta-heurísticas tem gerado um grande interesse em diversas áreas de tecnologia, indústria e serviço, devido a que provaram serem algoritmos eficientes para resolver uma ampla gama de complexos problemas de otimização pertencentes a diferentes domínios, como: logística, bioinformática e biologia computacional, desenho de engenharia, criação de redes, meio ambiente, transporte, mineração de dados, finanças, negócios, entre outros (TALBI, 2009).

No âmbito da programação da produção, a recente revisão de literatura do HFS de Ruiz e Vázquez-Rodríguez (2010) mostrou que métodos meta-heurísticos são requeridos para resolver problemas com considerações mais realistas. Segundo os autores, modelos de programação matemática e algoritmos *B&B* conformam 25% dos artigos revisados, enquanto regras de liberação e métodos heurísticos representam outro 50% dos artigos analisados. Isto implica que os 25% restantes abordam o problema através de métodos meta-heurísticos, dentre os quais Algoritmos Genéticos (GA,

*Genetic Algorithm*), Recozimento Simulado (SA, *Simulated Annealing*) e Busca Tabu (TS, *Tabu Search*) são os mais utilizados.

Este resultado apresenta uma clara oportunidade de pesquisa e motiva a implementação de métodos meta-heurísticos para resolver o problema estudado. Consequentemente, nesta seção apresenta-se um algoritmo de Busca Local Iterada (ILS, *Iterated Local Search*) com o objetivo de encontrar soluções de maior qualidade que aquelas encontradas pelos métodos heurísticos introduzidos no Capítulo 5.

A meta-heurística ILS tem sido escolhida, principalmente, por sua simplicidade, fácil codificação e os poucos operadores e parâmetros que devem ser ajustados, o qual resulta em um menor tempo de implementação e parametrização. Por outro lado, somente Naderi, Ruiz e Zandieh (2010) e Urlings, Ruiz e Stützle (2010) utilizam ILS para resolver problemas de programação da produção em sistema HFS, o que significa que o uso do ILS neste tipo de sistemas de produção é ainda pouco explorado.

A estrutura deste capítulo é a seguinte: a seção 6.1 descreve o funcionamento da meta-heurística ILS, detalhando cada um de seus componentes. A seção 6.2 apresenta a parametrização do ILS através da abordagem estatística de delineamento experimental. Finalmente, a seção 6.3 apresenta os resultados da avaliação computacional.

## 6.1. Descrição do ILS

A ideia subjacente do ILS é a de construir uma caminhada aleatória no espaço $S^*$, o qual representa o espaço de ótimos locais e é definido pela saída de um algoritmo de busca local determinado. Para desenvolver um algoritmo ILS são necessários quatro componentes básicos: um procedimento para gerar uma solução inicial, um procedimento de busca local, um esquema para perturbar uma solução e, finalmente, um critério de aceitação para determinar a partir de qual solução a busca deveria continuar (DEN BESTEN; STÜTZLE; DORIGO, 2001).

Em termos gerais, o ILS começa a partir de uma solução comumente obtida heuristicamente. Em seguida, uma busca local é aplicada sobre esta solução inicial, obtendo-se um ótimo local. Com o objetivo de escapar deste ótimo local, uma perturbação na solução é realizada e um novo ótimo local é encontrado depois de aplicar a busca local novamente. Finalmente, o critério de aceitação é usado com o objetivo de

decidir se o novo ótimo local deveria substituir o anterior. Este procedimento é repetido até que um determinado critério de terminação seja satisfeito.

Note que o ILS não foca a busca no espaço completo de soluções, mas sim no espaço formado pelas soluções que são retornadas pelo algoritmo de busca local utilizado, isto é, no espaço de ótimos locais. Adicionalmente, o ILS é diferente de outras meta-heurísticas, como SA e TS, pelo fato que ele não segue uma trajetória no espaço de busca, mas modificações, as quais correspondem a saltos no espaço de busca, são aplicadas para permitir escapar de ótimos locais. A Figura 6.1 ilustra a estrutura de um algoritmo ILS.

**Figura 6.1 – Pseudocódigo do algoritmo Busca Local Iterada (ILS)**

**procedure** ILS
  $\pi_0 \leftarrow GerarSoluçãoInicial$
  $\pi \leftarrow BuscaLocal(\pi_0)$
  **repeat**
    $\pi' \leftarrow Perturbação(\pi)$
    $\pi'' \leftarrow BuscaLocal(\pi')$
    $\pi \leftarrow Criterio\_de\_Aceitação(\pi, \pi'')$
  **until** criterio de terminação satisfeito
**end**

**Fonte: (PAN; RUIZ, 2012).**

Apesar da sua simplicidade, o ILS tem apresentado resultados do estado da arte quando aplicado ao problema do *Flowshop* clássico (RUIZ; MAROTO, 2005). Igualmente, mostra-se muito competitivo em outros problemas como o particionamento de grafos, a programação da produção em *Job shop* e o problema de minimização do atraso total ponderado em uma única máquina (STÜTZLE, 1998).

Informação detalhada do ILS pode ser encontrada na recente revisão apresentada por Lourenço, Martin e Stützle (2010). A seguir apresenta-se a descrição de cada componente do ILS.

### 6.1.1. Solução Inicial

A busca local aplicada à solução inicial $\pi_0$ gera o primeiro ótimo local $\pi$, a partir do qual começa a busca no espaço de ótimos locais. Esta solução $\pi_0$ pode ser uma solução aleatória inicial ou uma solução obtida através de uma heurística construtiva. Geralmente, começar a partir de uma solução obtida heuristicamente produz

uma solução $\pi$ de melhor qualidade em um menor tempo computacional, pois a busca local precisa menos iterações para atingir o ótimo local.

Dado os resultados obtidos no Capítulo 5, a heurística $NEHKK1_{ECT}$ será utilizada para gerar a solução inicial. Note que problemas de programação da produção requerem soluções de alta qualidade em curtos tempos computacionais, portanto o ILS deveria começar com uma solução da melhor qualidade possível.

Observe-se que esta escolha de solução inicial limita o espaço de busca ao espaço de permutações das tarefas. Porém, como destacado anteriormente, uma permutação não é suficiente para estabelecer um programa de produção completo para HFS. Assim, baseados nos resultados obtidos até o momento, a regra de designação ECT será utilizada para designar tarefas aos diferentes processadores.

Em qualquer caso, os resultados de Urlings, Ruiz e Serifoğlu (2010) indicaram que esquemas mais sofisticados para representar soluções são promissórios somente quando empregados em problemas pequenos, pois quando o tamanho do problema aumenta tem-se sérios problemas de eficiência devido ao aumento do espaço de busca e o maior tempo requerido para explorá-lo. Igualmente, Sacchi (1997) testou vários algoritmos genéticos com diversos esquemas de representação para resolver um HFS. Os resultados encontrados indicaram que os algoritmos com esquema de representação tipo permutação foram mais bem sucedidos que aqueles que utilizaram esquemas mais sofisticados e complexos.

Note que utilizar permutações ainda é intratável computacionalmente, porém é fácil de conceituar, implementar e, quando uma adequada regra de designação é utilizada, é possível encontrar soluções ótimas ou quase ótimas (RUIZ; VÁZQUEZ-RODRÍGUEZ, 2010; SANTOS; HUNSUCKER; DEAL, 1995).

### 6.1.2. Perturbação

A principal desvantagem de métodos de melhoria iterativa é que ficam presos em ótimos locais que são significativamente piores que o ótimo global. Tal como algoritmos do tipo SA, o ILS escapa de ótimos locais aplicando perturbações ao ótimo local atual. Em geral, a busca local não deveria ser capaz de desfazer a perturbação, pois assim o algoritmo voltaria para o mesmo ótimo local recentemente visitado. Notavelmente, movimentos aleatórios em uma estrutura de vizinhança maior que aquela utilizada pelo algoritmo de busca local são suficientes para atingir este objetivo. Porém,

ainda melhores resultados podem ser atingidos se a perturbação leva em conta propriedades do problema (LOURENÇO; MARTIN; STÜTZLE, 2010).

Um conceito de vital importância para o desempenho do ILS é a força da perturbação, a qual é definida como o número de componentes da solução que são modificados. Por exemplo, no problema do caixeiro viajante seria o número de arcos que são modificados no *tour*, enquanto no problema do *Flowshop* clássico seria o número de tarefas que são movidas da sua posição. A questão é determinar quanto deveria mudar a solução atual: se a perturbação é muito forte, o ILS se comportará como um algoritmo de busca aleatório e terá pouca probabilidade de encontrar melhores soluções. Por outro lado, se a perturbação é demasiado pequena, a busca local voltará ao ótimo local recentemente visitado e a diversificação da busca será limitada.

Outro aspecto importante é a natureza da perturbação. Para alguns problemas, perturbações de tamanho fixo (independente do tamanho do problema) podem levar a obter resultados satisfatórios. Para outros problemas, fazer perturbações de tamanho fixo pode levar a pobres resultados.

Portanto, no ILS não é suficiente estabelecer ou determinar um mecanismo para realizar uma perturbação, mas também é necessário definir o tamanho e natureza da mesma. Ou seja, é preciso definir quantos elementos de uma solução devem ser modificados, assim como estabelecer se este número depende, ou não, do tamanho do problema.

A seguir apresentam-se os operadores de perturbação considerados para o desenvolvimento do ILS:

- *GL* (*Giant Leap*): este operador, o qual tenta atingir um equilíbrio entre intensificação e diversificação, gera um número *NumPert* de soluções modificadas a partir da solução incumbente, $\pi$, e seleciona a melhor delas como solução perturbada. Para modificar a solução incumbente, *GL* seleciona e realoca aleatoriamente um número *d* de tarefas na permutação. Note que esta coleção de movimentos aleatórios altera a solução atual de forma imprevisível e, portanto o resultado será, provavelmente, uma solução muito pior. Portanto, com o objetivo de levar a busca a regiões promissórias do espaço, *GL* não gera uma única solução perturbada, mas gera várias soluções modificadas e escolhe a melhor delas como solução perturbada. (NADERI; RUIZ; ZANDIEH, 2010; NADERI; ZANDIEH; ROSHANAEI,

2008). Observe que este operador precisa a definição dos parâmetros *NumPert* e *d*.

- *GeigerPert*: este operador escolhe aleatoriamente um subconjunto de quatro tarefas consecutivas de $\pi$. Especificamente, este subconjunto é formado pelas tarefas nas posições $j, j+1, j+2$ e $j+3$, respectivamente. A solução perturbada $\pi'$ é gerada movendo a tarefa na posição $j$ para $j+3$, a tarefa na posição $j+1$ para $j+2$, a tarefa na posição $j+2$ para $j$ e, finalmente, a tarefa na posição $j+3$ para $j+1$. Note que este operador modifica significativamente a solução incumbente $\pi$, mas deixa intactas as tarefas nas posições $k < j$ e $k > j+3$. Desta forma *GeigerPert* visa levar a busca até regiões ainda não exploradas, enquanto mantém a maioria das características da atual solução incumbente (GEIGER, 2011). Observe que o tamanho da perturbação é fixo, pois sempre são modificados quatro elementos da solução incumbente.
- *TypeNEH*: este operador seleciona aleatoriamente um número *d* de tarefas de $\pi$. Estas *d* tarefas são removidas de $\pi$ na ordem na qual foram selecionadas. Como resultado, duas subsequências são obtidas. A subsequência $\pi_D$, formada por $n-d$ tarefas, é a sequência resultante de remover as *d* tarefas de $\pi$, enquanto a sequência $\pi_R$ contém as tarefas que foram removidas de $\pi$. Logo, este operador reinsere cada tarefa de $\pi_R$ na melhor posição possível de $\pi_D$, tal como feito pela heurística NEH. A ordem na qual a reinserção das tarefas é feita é exatamente a mesma na qual elas foram removidas de $\pi$ (RUIZ; STÜTZLE, 2007). Note que desta forma é possível obter uma solução perturbada suficientemente diferente de $\pi$ e de boa qualidade, pois cada elemento removido é reinserido na melhor posição possível.

É preciso mencionar que o último operador de perturbação, *TypeNEH*, está diretamente relacionado com o algoritmo guloso iterado (IG, *Iterated Greedy*), o qual gera uma sequência de soluções através de iterações de uma heurística construtiva gulosa. Este algoritmo e o ILS apresentam algumas diferenças: em lugar de iterar sobre uma busca local, como feito pelo ILS, o IG itera sobre uma heurística construtiva (RUIZ; STÜTZLE, 2007). Adicionalmente, no IG o passo da busca local é aplicado de formal opcional (PAN; RUIZ, 2012; RUIZ; STÜTZLE, 2008).

Para maior clareza, o pseudocódigo do IG é apresentado na Figura 6.2. Note que o passo de busca local é aplicado opcionalmente e que o operador de perturbação é substituído por um operador de destruição e construção da solução incumbente.

**Figura 6.2 – Pseudocódigo do algoritmo Guloso Iterado (IG)**

**procedure** IG
  $\pi_0 \leftarrow GerarSoluçãoInicial$
  $\pi \leftarrow BuscaLocal(\pi_0)$ %Opcional
  **repeat**
    $\pi' \leftarrow Destruição\_Construção(\pi)$
    $\pi'' \leftarrow BuscaLocal(\pi')$ %Opcional
    $\pi \leftarrow Criterio\_de\_Aceitação(\pi, \pi'')$
  **until** criterio de terminação satisfeito
**end**

**Fonte: (RUIZ; STÜTZLE, 2008)**

### 6.1.3. Busca Local

Algoritmos de busca local começam desde alguma solução dada e tentam encontrar uma melhor solução na vizinhança da solução atual. Em caso que uma melhor solução seja encontrada, esta substitui a solução atual e a busca local continua a partir dela. A vizinhança de uma solução é definida, em termos simples, como o conjunto de soluções que podem ser geradas a partir dela. Note que é preciso estabelecer claramente a estrutura da vizinhança, isto é, a forma na qual a vizinhança é definida. Esta escolha determina largamente se a busca encontrará soluções de alta qualidade ou se, pelo contrário, encontrará ótimos locais pobres (AHUJA et al., 2002).

Como regra geral, quanto maior a vizinhança, melhor é qualidade das soluções ótimas locais encontradas. Ao mesmo tempo, quanto maior a vizinhança, maior o tempo requerido para explorá-la em cada iteração. Devido a que varias execuções da busca são feitas, a partir de diferentes pontos iniciais, tempos de execução mais longos por cada iteração resultam em menos rodadas por unidade de tempo. Por esta razão, uma vizinhança maior não necessariamente produz um método mais efetivo a menos que seja possível explorá-la de forma eficiente.

A seguir se definem as estruturas de vizinhança comumente utilizadas em problemas de programação da produção:

- Intercâmbio adjacente (AI, *Adjacent Interchange*): esta estrutura troca pares de tarefas adjacentes na permutação. O tamanho da vizinhança é $n-1$.

- *Swap*: esta estrutura, às vezes denominada *Exchange*, troca as tarefas nas posições $j$ e $k$, $j \neq k$. Note que esta estrutura não se limita a trocas entre tarefas adjacentes. O tamanho da vizinhança é $\frac{1}{2}n(n-1)$.
- Inserção: esta estrutura remove a tarefa na posição $j$ e inseri-la na posição $k$. O tamanho da vizinhança é $(n-1)^2$.

A estrutura de vizinhança Inserção é considerada superior às estruturas AI e *swap* para o problema de programação da produção em *Flowshop* clássico, no qual as soluções são representadas através de permutações de tarefas (OSMAN; POTTS, 1989; RUIZ; MAROTO; ALCARAZ, 2005; STÜTZLE, 1998; TAILLARD, 1990). Consequentemente, os operadores de busca local que são apresentados posteriormente baseiam-se totalmente em movimentos de inserção.

Seja $v=(j,k)$ um par de posições, tal que $j,k \in \{1,2,\ldots,n\} \wedge j \neq k$. A nova sequência $\pi'$ é obtida a partir de $\pi$, removendo a tarefa na posição $j$ e inserindo-a na posição $k$. Desta forma, $\pi' = (\pi_1,\ldots,\pi_{j-1},\pi_{j+1},\ldots,\pi_k,\pi_j,\pi_{k+1},\ldots,\pi_n)$ se $j<k$ e $\pi' = (\pi_1,\ldots,\pi_{k-1},\pi_j,\pi_k,\ldots,\pi_{j-1},\pi_{j+1},\ldots,\pi_n)$ se $j>k$.

A vizinhança inteira de $\pi$ pode ser gerada a partir do conjunto de movimentos $V = \{(j,k): j,k \in \{1,2,\ldots,n\} \wedge k \notin \{j-1, j\}\}$. Note que para qualquer $j$ e $k$ tal que $|j-k|=1$ os movimentos $v=(j,k)$ e $v'=(k,j)$ resultam na mesma permutação. Portanto, considerar $k \notin \{j-1, j\}$ ajuda a evitar permutações redundantes.

Stützle (1998) indica que o comportamento e desempenho do ILS são sensíveis à escolha do algoritmo de busca local e, portanto esta escolha deveria ser otimizada quando possível. Segundo Lourenço, Martin e Stützle (2010) é lógico pensar que quanto melhor o algoritmo de busca local, melhor o correspondente ILS. No entanto, se o tempo computacional é fixado, poderia ser melhor aplicar mais frequentemente um algoritmo de busca local mais rápido, porém menos eficaz que um algoritmo mais lento e mais poderoso. Certamente, a melhor escolha depende de quanto tempo mais é requerido para executar o melhor algoritmo de busca local. Se a diferença de velocidade não é grande, então vale a pena utilizar o melhor algoritmo.

Para O ILS proposto, três operadores de busca local baseados na estrutura de vizinhança tipo inserção são definidos a seguir:

O operador *LSNaderi*, apresentado na Figura 6.3, é um simples algoritmo de busca local que é interrompido assim que a primeira melhoria é encontrada. A busca começa com a tarefa na posição $j=1$ da solução incumbente $\pi$, referida como $\pi_1$, a qual é selecionada e reinserida em uma nova posição selecionada de forma aleatória. Se a nova sequência, $\pi'$, resulta em um melhor *makespan*, então a solução incumbente é substituída por $\pi'$ e a busca local é finalizada. Caso contrário, o procedimento é repetido com a tarefa na seguinte posição da sequência, $j+1$. Note que a busca itera, no máximo, para as $n$ posições da sequência (NADERI; RUIZ; ZANDIEH, 2010).

**Figura 6.3 – Pseudocódigo do algoritmo de busca local LSNaderi**

**procedure** LSNaderi$(\pi)$
$j=1$
  **While** $j \leq n$ **do**
    $\pi' \leftarrow$ sequência produzida reinserindo $\pi_j$ em uma posição aleatoria diferente
    **if** $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
      $\pi = \pi'$
      **break**
    **endif**
    $j = j+1$
  **endwhile**

**Fonte: (NADERI; RUIZ; ZANDIEH, 2010)**

Rajendran e Ziegler (1997) propõem um algoritmo de busca baseado em inserção, denominado *RZ*. Uma descrição deste procedimento de busca local é apresentada na Figura 6.4, em que $\pi^s = (\pi_1^s, \pi_2^s, \ldots, \pi_n^s)$ representa uma sequência semente e $\pi'$ é uma sequência obtida através da inserção das tarefas de $\pi^s$ em diferentes posições. Para cada posição $j$ de $\pi^s$, a tarefa $\pi_j^s$ é removida de $\pi'$ e reinserida na melhor posição possível, isto é, aquela que resulta no menor *makespan*. Finalmente, $\pi'$ substitui a solução incumbente, $\pi$, se seu *makespan* for menor.

**Figura 6.4 – Pseudocódigo do algoritmo de busca local RZ**

**procedure** $\mathrm{RZ}(\pi)$

$\pi^s = \pi$

**For** $j = 1$ **to** $n$ **do**

$\pi' \leftarrow \pi$

Remover a tarefa $\pi_j^s$ de $\pi'$

Testar $\pi_j^s$ em todas as possíveis posições de $\pi'$, exceto na posição original

Inserir $\pi_j^s$ em $\pi'$ na posição que resulta no menor *makespan*

**if** $C_{\max}(\pi') < C_{\max}(\pi)$ **then**

$\pi = \pi'$

**endif**

**endfor**

O último operador de busca local, *LSUrlings*, é similar ao procedimento *RZ*. A tarefa na posição $j$ da solução incumbente $\pi$, $\pi_j$, é testada em cada possível posição da sequência, exceto na posição original, e inserida naquela que resulta no menor *makespan*. Em seguida, caso haja uma melhoria no *makespan*, a solução incumbente é atualizada. Diferentemente do operador *RZ*, o qual finaliza após percorrer as $n$ posições da sequência, este procedimento repete-se até que nenhuma melhoria no valor do *makepan* seja encontrada. Adicionalmente, o procedimento começa a partir da tarefa na última posição da sequência, pois maior informação acerca do programa de produção pode ser reutilizada nos cálculos do algoritmo, o que resulta em um menor tempo computacional. Informação acerca deste algoritmo de busca, assim como detalhes de implementação podem ser encontrados no trabalho de Urlings, Ruiz e Stützle (2010).

### 6.1.4. Critério de Aceitação

O ILS faz um percorrido aleatório no espaço dos ótimos locais. O mecanismo de perturbação junto com a busca local definem as possíveis transições entre a solução atual $\pi^*$ e a solução vizinha $\pi^{*}{}'$, as quais pertencem ao conjunto de ótimos locais. O critério de aceitação determina se $\pi^{*}{}'$ é aceita ou não como nova solução atual e também tem forte influencia na efetividade do percorrido no espaço dos ótimos locais.

Em termos simples, o critério de aceitação controla o equilíbrio entre intensificação e diversificação da busca. A intensificação está relacionada à exploração minuciosa das porções do espaço de busca que parecem promissórias, com o objetivo de garantir que as melhores soluções destas áreas sejam encontradas. Por outro lado, a

diversificação está relacionada com a exploração de novas áreas do espaço de busca e seu objetivo é levar a busca até áreas ainda não visitadas.

Uma intensificação forte é atingida através do critério de aceitação *Better*, no qual somente melhores soluções são aceitas. Assim, entre os dois ótimos locais $\pi^*$ e $\pi^{*}{}'$ escolhe-se sempre aquele de melhor qualidade, isto é, o ótimo local com melhor valor de função objetivo.

No outro extremo, o critério *RW* (*Random Walk*) aceita como nova solução o ótimo local visitado mais recentemente, isto é, entre $\pi^*$ e $\pi^{*}{}'$ sempre escolhe $\pi^{*}{}'$. Note que este critério favorece diversificação sobre intensificação.

Várias opções intermediárias entre estes dois casos extremos também são possíveis. Por exemplo, o critério *LSMC* sempre aceita $\pi^{*}{}'$ se é melhor que $\pi^*$. Caso contrário, se $\pi^{*}{}'$ é pior que $\pi^*$, $\pi^{*}{}'$ é aceito com uma probabilidade $\exp\left\{ f\left(\pi^*\right) - f\left(\pi^{*}{}'\right)/T \right\}$, em que $T$ é um parâmetro denominado temperatura, o qual usualmente é reduzido ao longo da execução do algoritmo, tal como em algoritmos do tipo SA. Note que *LSMC* é similar ao critério *RW* se $T$ é muito alto, enquanto é similar ao critério *Better* se $T$ é muito baixo. Desta forma, o critério *LSMC* tende a enfocar na diversificação nas primeiras iterações do ILS e depois, na medida em que $T$ é reduzido, somente melhores soluções são aceitas, enfocando na intensificação.

Como observado um aspecto importante deste critério é a necessidade de definir previamente a forma como o parâmetro $T$ será reduzido ao longo do algoritmo. Alternativamente, uma variação deste critério simplesmente mantém constante o valor de $T$ ao longo do algoritmo, de tal forma que a probabilidade de aceitar ou não a solução $\pi^{*}{}'$ depende diretamente da diferença de qualidade entre $\pi^{*}{}'$ e $\pi^*$ (RUIZ; STÜTZLE, 2007, 2008; URLINGS; RUIZ; STÜTZLE, 2010).

Estes três critérios de aceitação serão utilizados na parametrização do ILS.

## 6.2. Parametrização do ILS

Pesquisadores de praticamente todas as áreas do conhecimento realizam experimentos, geralmente para descobrir algo acerca de um processo ou sistema particular. Formalmente, um experimento pode definir-se como um teste ou conjunto de testes nos quais se fazem mudanças deliberadas nas variáveis de entrada de um processo ou sistema para observar e identificar as razões de possíveis mudanças nas variáveis de saída. De igual forma, a experimentação tem um papel crucial no desenvolvimento de

processos robustos, isto é, um processo que seja afetado minimamente por fontes de variabilidade externa (MONTGOMERY, 2005).

Este conceito é totalmente aplicável ao desenvolvimento de algoritmos. Em particular, o ILS está composto por quatro componentes ou operadores principais, encargados de gerar a solução inicial, executar a busca loca, perturbar a solução incumbente e determinar com qual solução deveria continuar a busca, respectivamente. Claramente diversas alternativas estão disponíveis para cada operador individual e cada uma pode levar a diferentes desempenhos em termos de qualidade e eficiência computacional.

O objetivo desta seção é determinar, através de um extenso delineamento experimental, quais são os operadores de maior influência ou impacto no desempenho do ILS e identificar a melhor escolha para cada um deles, de forma que o ILS desenvolvido possa encontrar soluções com a maior qualidade possível em curtos tempos computacionais.

den Besten, Stützle e Dorigo (2001) propuseram uma forma heurística para configurar o ILS: a cada passo somente um operador é considerado, mantendo fixos os demais em algumas escolhas "razoáveis". Neste ponto, as diversas alternativas do operador estudado são testadas e a melhor delas é selecionada. Uma vez feita uma escolha para o operador analisado, o processo é repetido com o seguinte operador do algoritmo.

A desvantagem desta estratégia é que não considera as interações entre os operadores, isto é, não identifica se o desempenho de um determinado operador do algoritmo é afetado significativamente pela escolha feita para outros operadores.

A abordagem adequada para analisar vários operadores é um experimento fatorial, no qual os vários fatores variam conjuntamente, em lugar de um por vez. Este tipo de experimento considera todas as possíveis combinações dos fatores, permitindo detectar as interações entre eles (MONTGOMERY, 2005).

A Tabela 6.1 mostra os diferentes fatores e níveis considerados no delineamento experimental.

**Tabela 6.1 – Fatores e níveis considerados no delineamento experimental**

| Fatores | Níveis |
|---|---|
| Operador de Busca Local | *LSNaderi*, *RZ*, *LSUrlings* |
| Operador de Perturbação | *GeigerPert*, *GL*(2), *GL*(3), *GL*(4), *TypeNEH*(2), *TypeNEH*(3), *TypeNEH*(4) |
| Critério de Aceitação | *Better*, *RW*, *LSMC* |

Como observado, para a busca local são considerados os três operadores de busca introduzidos na seção 6.1.3. Para a perturbação são considerados os três operadores definidos na seção 6.1.2. Especificamente, *GL*(2), *GL*(3) e *GL*(4) denotam o operador *GL* com tamanhos de perturbação de 2, 3 e 4, respectivamente. De forma similar, *TypeNEH*(2), *TypeNEH*(3) e *TypeNEH*(4) representam o operador *TypeNEH* com tamanhos de perturbação de 2, 3 e 4, respectivamente. Por definição, o operador *GeigerPert* tem um tamanho de perturbação de 4. Estes tamanhos de perturbação foram definidos com base nos resultados de testes preliminares, os quais mostraram que perturbações de tamanho fixo atingiam melhores resultados que tamanhos de perturbação dependentes do tamanho do problema.

Note que o operador *GL* precisa, adicionalmente, da definição do parâmetro *NumPert*, o qual determina o número de soluções perturbadas que devem ser geradas. Seguindo os resultados dos testes preliminares, este valor é fixado em 20. Desta forma, o operador *GL* deve gerar vinte soluções a partir da solução incumbente e escolher a melhor delas como solução perturbada final.

Finalmente, como critérios de aceitação são considerados os operadores *Better*, *RW* e *LSMC*. Para o operador *LSMC* considera-se o parâmetro *T* como uma constante, a qual depende diretamente do problema a ser resolvido. Desta forma não há necessidade de estabelecer um esquema para a diminuição da temperatura ao longo da execução do algoritmo. Para cada problema, *T* é calculado como segue:

$$T = \lambda \cdot \left( \frac{\sum_{i \in M} \sum_{j \in N} APT_{ij}}{10 \cdot n \cdot m} \right)$$

em que $\lambda$ é um parâmetro que deve ser ajustado (RUIZ; STÜTZLE, 2007, 2008). Resultados dos testes preliminares para o parâmetro $\lambda$ mostraram que o ILS é robusto em relação a este parâmetro, isto é, os resultados não mudam significativamente quando os valores de $\lambda$ são alterados. Em qualquer caso, $\lambda = 7$ mostrou resultados levemente melhores e, portanto é selecionado. Os resultados dos testes preliminares são apresentados detalhadamente no Apêndice C.

Todos os fatores mencionados resultam em um total de $3 \cdot 7 \cdot 3 = 63$ combinações e algoritmos diferentes. Para cada uma destas combinações foi resolvido um conjunto de 80 instâncias (ver Apêndice C). Devido à natureza estocástica do ILS, cinco réplicas para cada combinação e instâncias foram feitas. Desta forma o delineamento experimental tem $63 \cdot 80 \cdot 5 = 25200$ observações.

Note que é necessário fazer todos os testes da parametrização do ILS utilizando um conjunto de instâncias diferente do conjunto de teste final com o objetivo de evitar viés nos resultados finais.

Como critério de terminação dos algoritmos é configurado um tempo máximo de execução, definido como $t \cdot n \cdot \sum_{i \in M} m_i$, em que $t = 25$ milissegundos. Desta forma todos os algoritmos tem o mesmo tempo computacional disponível para resolver uma determinada instância. Adicionalmente, configurar o tempo limite de execução desta forma permite uma melhor análise estatística, pois se um tempo de execução é fixado para todas as instâncias, o efeito do tamanho da mesma não pode ser adequadamente estudado. Isto é, se o mesmo tempo (ou número de iterações) é dado para todas as instâncias, sem importar o tamanho da mesma, piores resultados para instâncias de maior tamanho não podem ser atribuídos unicamente ao tamanho da instância, pois eles poderiam ser resultado de tempo computacional insuficiente (NADERI; RUIZ; ZANDIEH, 2010; PAN; RUIZ, 2012).

Para comparar o desempenho dos diferentes operadores do ILS, novamente a percentagem de desvio relativo (RPD) é utilizada. Todos os operadores foram implementados na linguagem C++ e executados no mesmo *notebook* utilizado para a avaliação do modelo de programação inteira mista e das heurísticas, apresentados nos capítulos anteriores.

Os resultados da parametrização são analisados através da ANOVA e sumarizados através de gráficos de médias e intervalos de confiança de 95% de Fisher LSD. Para uma melhor compreensão da metodologia utilizada para a parametrização do ILS, as diferentes tabelas ANOVA requeridas serão apresentadas ao longo da apresentação dos resultados.

A Tabela 6.2 mostra a ANOVA dos diferentes operadores do ILS com um nível de confiança de 95%. Valores P menores a 0,05 indicam que o respectivo fator ou interação é estatisticamente significativo. No entanto, os valores P não permitem identificar qual fator é mais importante. Com este objetivo, utiliza-se a coluna da

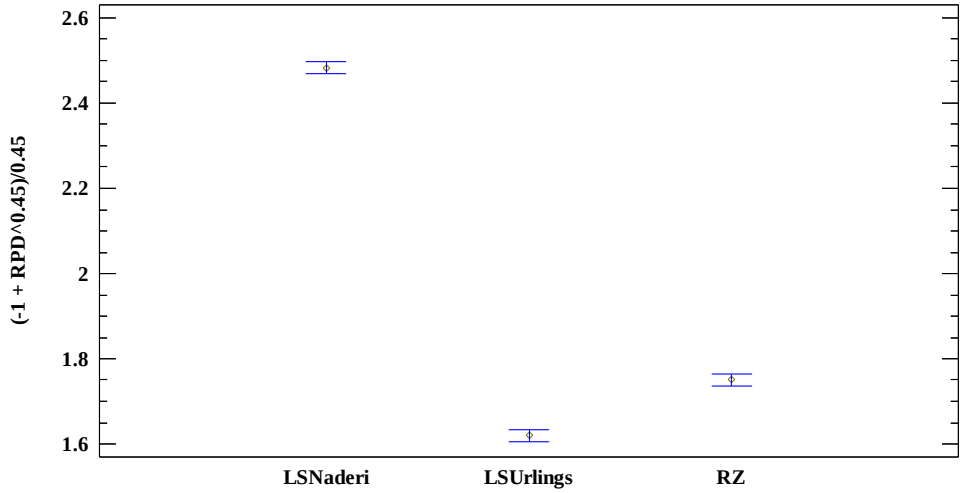
Razão-F: quanto maior a Razão-F, maior a influencia do respectivo fator ou interação sobre a variável de resposta. Note que a ANOVA inclui o efeito do tamanho da instância.

**Tabela 6.2 – Análise de Variância da parametrização do ILS**

| Fonte | Soma de Quadrados | Graus de liberdade | Quadrado Médio | Razão-F | Valor-P |
|---|---|---|---|---|---|
| **Efeitos Principais** | | | | | |
| A: Perturbação | 253,243 | 6 | 42,207 | 239,800 | 0,000 |
| B: Busca Local | 726,948 | 2 | 363,474 | 2065,060 | 0,000 |
| C: Critério | 450,891 | 2 | 225,446 | 1280,860 | 0,000 |
| D: Tarefas | 271,343 | 1 | 271,343 | 1541,620 | 0,000 |
| E: Estações | 3,292 | 1 | 3,292 | 18,700 | 0,000 |
| F: Processadores | 1,179 | 1 | 1,179 | 6,700 | 0,010 |
| G: Buffers | 21,014 | 1 | 21,014 | 119,390 | 0,000 |
| **Interações** | | | | | |
| AB | 79,494 | 12 | 6,625 | 37,640 | 0,000 |
| AC | 15,057 | 12 | 1,255 | 7,130 | 0,000 |
| AD | 12,916 | 6 | 2,153 | 12,230 | 0,000 |
| AE | 0,383 | 6 | 0,064 | 0,360 | 0,903 |
| AF | 3,241 | 6 | 0,540 | 3,070 | 0,005 |
| AG | 1,605 | 6 | 0,268 | 1,520 | 0,167 |
| BC | 111,197 | 4 | 27,799 | 157,940 | 0,000 |
| BD | 23,959 | 2 | 11,979 | 68,060 | 0,000 |
| BE | 0,357 | 2 | 0,178 | 1,010 | 0,363 |
| BF | 5,142 | 2 | 2,571 | 14,610 | 0,000 |
| BG | 0,727 | 2 | 0,364 | 2,070 | 0,127 |
| CD | 45,351 | 2 | 22,675 | 128,830 | 0,000 |
| CE | 0,810 | 2 | 0,405 | 2,300 | 0,100 |
| CF | 0,389 | 2 | 0,194 | 1,100 | 0,332 |
| CG | 0,304 | 2 | 0,152 | 0,860 | 0,422 |
| DE | 8,364 | 1 | 8,364 | 47,520 | 0,000 |
| DF | 8,879 | 1 | 8,879 | 50,440 | 0,000 |
| DG | 3,251 | 1 | 3,251 | 18,470 | 0,000 |
| EF | 5,378 | 1 | 5,378 | 30,550 | 0,000 |
| EG | 0,966 | 1 | 0,966 | 5,490 | 0,019 |
| FG | 9,389 | 1 | 9,389 | 53,340 | 0,000 |
| **Resíduos** | 871,433 | 4951 | 0,176 | | |
| Total (Corregido) | 2936,500 | 5039 | | | |

O operador do ILS com a maior Razão-F é a busca local, com valor de 2065,060. Observe que a interações da busca local com a perturbação e o critério de aceitação são muito menores, com valores de 37,640 e 157,940, respectivamente. Desta forma, a busca local é o operador do ILS que maior influencia tem sobre a variável de resposta. Este resultado era esperado, pois da eficácia deste operador depende a qualidade das soluções encontradas pelo ILS. Para determinar qual dos operadores de busca local é a melhor alternativa, utiliza-se o gráfico de médias e intervalos de confiança de 95% de Fisher LSD, apresentado pela Figura 6.5.

**Figura 6.5 – Gráfico de média e intervalos de confiança de 95% de Fisher do operador de busca local**




A Figura 6.5 permite identificar claramente que o operador de busca local *LSUrlings* atinge os melhores resultados. Com base neste resultado, o operador de busca local é fixado como *LSUrlings*.

A seguir, uma nova ANOVA é realizada com os fatores restantes. Os resultados são apresentados na Tabela 6.3.

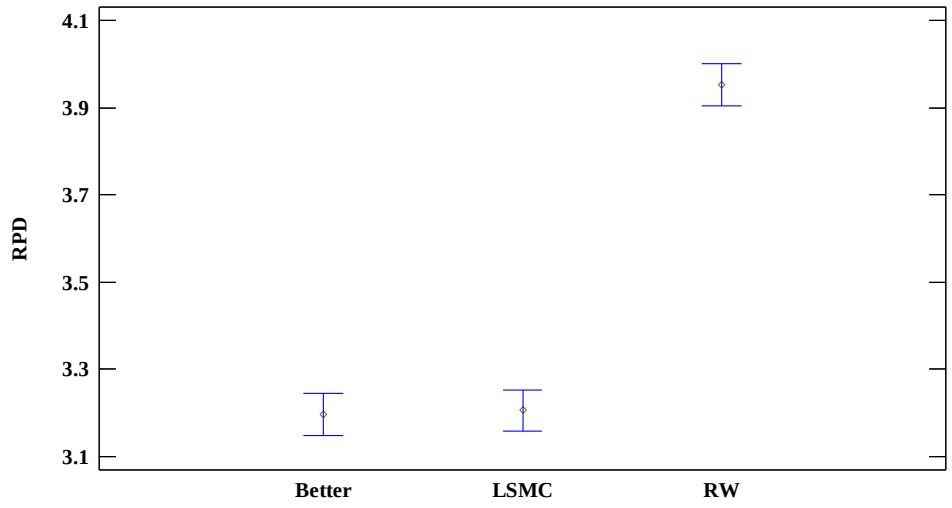
**Tabela 6.3 – Análise de variância da parametrização do ILS. Busca local fixada em *LSUrlings*.**

| Fonte | Soma de Quadrados | Graus de liberdade | Quadrado Médio | Razão-F | Valor-P |
|---|---|---|---|---|---|
| Efeitos Principais | | | | | |
| A: Tarefas | 134,907 | 1 | 134,907 | 205,670 | 0,000 |
| B: Estações | 0,650 | 1 | 0,650 | 0,990 | 0,319 |
| C: Processadores | 24,773 | 1 | 24,773 | 37,770 | 0,000 |
| D: Buffers | 16,331 | 1 | 16,331 | 24,900 | 0,000 |
| E: Perturbação | 51,648 | 6 | 8,608 | 13,120 | 0,000 |
| F: Critério | 210,654 | 2 | 105,327 | 160,570 | 0,000 |
| Interações | | | | | |
| AB | 7,155 | 1 | 7,155 | 10,910 | 0,001 |
| AC | 15,593 | 1 | 15,593 | 23,770 | 0,000 |
| AD | 2,691 | 1 | 2,691 | 4,100 | 0,043 |
| AE | 5,037 | 6 | 0,839 | 1,280 | 0,263 |
| AF | 58,434 | 2 | 29,217 | 44,540 | 0,000 |
| BC | 15,884 | 1 | 15,884 | 24,220 | 0,000 |
| BD | 2,113 | 1 | 2,113 | 3,220 | 0,073 |
| BE | 1,211 | 6 | 0,202 | 0,310 | 0,933 |
| BF | 1,324 | 2 | 0,662 | 1,010 | 0,365 |
| CD | 16,411 | 1 | 16,411 | 25,020 | 0,000 |
| CE | 0,740 | 6 | 0,123 | 0,190 | 0,980 |
| CF | 2,328 | 2 | 1,164 | 1,770 | 0,170 |
| DE | 1,351 | 6 | 0,225 | 0,340 | 0,914 |
| DF | 0,369 | 2 | 0,185 | 0,280 | 0,755 |
| EF | 3,133 | 12 | 0,261 | 0,400 | 0,965 |
| Resíduos | 1060,670 | 1617 | 0,656 | | |
| Total (Corregido) | 1633,410 | 1679 | | | |

Desta vez o fator com maior influencia sobre a variável de resposta é o número de tarefas, com Razão-F de 205,670. No entanto, este é fator não é controlável, pois em situações práticas não é possível determinar a priori quantas tarefas deverão ser programadas. Dentro dos fatores controláveis, o critério de aceitação é o mais influente, com Razão-F de 160,570. Observe que a interação do critério de aceitação com a perturbação não é significativa, pois apresenta um valor P de 0,965.

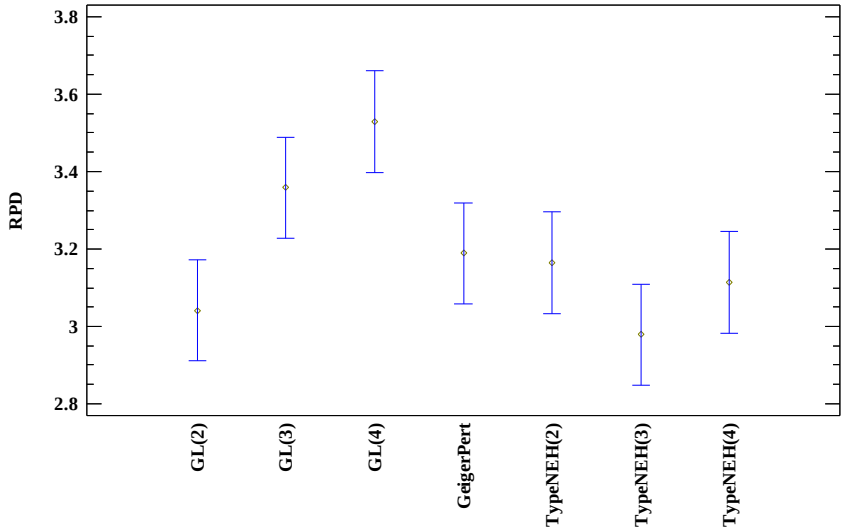
A Figura 6.6, através do gráfico de médias e intervalos de confiança de 95% de Fisher LSD, permite determinar que não existe diferença estatisticamente significativa entre *Better* e *LSMC*. De fato, observa-se que a diferenças entre as médias é quase imperceptível, com *Better* sendo levemente melhor que *LSMC*. Desta forma, *Better* é selecionado como critério de aceitação. Note que escolher *LSMC* teria sido igualmente válido, pois estatisticamente não existe diferença entre ambos os critérios.

**Figura 6.6 – Gráfico de médias e intervalo de confiança de 95% de Fisher de critério de aceitação**




Finalmente, uma última ANOVA é realizada, desta vez fixando os parâmetros de busca local e critério de aceitação. Evidentemente, somente falta determinar a melhor escolha do parâmetro de perturbação e, portanto a tabela ANOVA não é apresentada. O gráfico de médias é intervalos de confiança de 95% de Fisher é ilustrado pela Figura 6.7. Esta figura permite estabelecer que não existe diferenças estatisticamente significativas entre os operadores *GL*(2), *GeigerPert*, *TypeNEH*(2), *TypeNEH*(3) e *TypeNEH*(4), respectivamente.

**Figura 6.7 – Gráfico de média e intervalos de confiança de 95% de Fisher do operador de perturbação**




Este resultado se deve principalmente à redução do tamanho de amostra utilizada na ANOVA. Note que, na medida em que os demais fatores são fixados, o número de dados do experimento é reduzido significativamente, o que implica que a ANOVA tem menos dados disponíveis e, portanto perde potência. Consequentemente a probabilidade de cometer erro estatístico tipo II aumenta, isto é, aumenta a probabilidade de concluir que não existe diferença estatisticamente significativa quando em realidade esta existe.

Embora estes resultados indicam que não há diferenças estatisticamente significativas entre vários operadores, intuitivamente é possível observar que o operador *TypeNEH*(3) atinge resultados relativamente melhores.

Em resumo, a configuração final do ILS é a seguinte: a solução inicial é gerada através da heurística $NEHKK1_{ECT}$, a busca local é realiza pelo operador *LSUrlings*, a perturbação é feita pelo operador *TypeNEH*(3) e, finalmente, o critério de aceitação é *Better*. A avaliação computacional deste ILS será apresentada nas seguintes seções.

## 6.3. Avaliação Computacional

Nesta seção apresentam-se os resultados da avaliação computacional do ILS sobre os conjuntos de teste A e B. O ILS é comparado com as heurísticas propostas no capítulo 5 e os resultados analisados através de análises de variância.

Todas as tabelas ANOVA deste capítulo podem ser consultadas no Apêndice B. Os resultados da avaliação computacional sobre o conjunto de teste A são

apresentados na seção 6.3.1, enquanto a seção 6.3.2 apresenta os resultados da avaliação computacional sobre o conjunto de teste B.

### 6.3.1. Avaliação Computacional do Conjunto de Teste A

Nesta seção apresenta-se o resultado da avaliação computacional do ILS sobre o conjunto de teste A. Inicialmente é analisado o subconjunto de instâncias para as quais uma solução ótima é conhecida. Posteriormente, considera-se o subconjunto de instâncias para as quais somente uma solução factível é conhecida. Finalmente, uma ANOVA é utilizada para estabelecer se as diferenças observadas entre o ILS e as heurísticas são significativas.

Devido à natureza estocástica do ILS, cada instância é resolvida cinco vezes e o valor do *makespan* resulta de tomar a média destas cinco réplicas. Para medir o desempenho do ILS, o RPD sobre a melhor solução conhecida de cada instância é calculado:

$$RPD = \left( \frac{Heu_{Sol} - MIP_{Best}}{MIP_{Best}} \right) \cdot 100$$

em que $Heu_{Sol}$ denota a solução encontrada pelo ILS e $MIP_{Best}$ é a melhor solução encontrada pelo modelo de programação inteira mista para uma instância dada.

A Tabela 6.4 mostra o RPD médio do ILS, considerando somente as instâncias para as quais uma solução ótima é conhecida. Os resultados são consolidados por cada combinação de $n$ e $m$. Portanto, cada célula da Tabela 6.4 representa a média sobre todos os demais fatores.

Como observado, o ILS melhora significativamente os resultados atingidos pelas heurísticas $NEHKK1_{ECT}$ e $NEH_{ECT}$, as quais foram as duas melhores heurísticas segundo os resultados do capítulo 5.

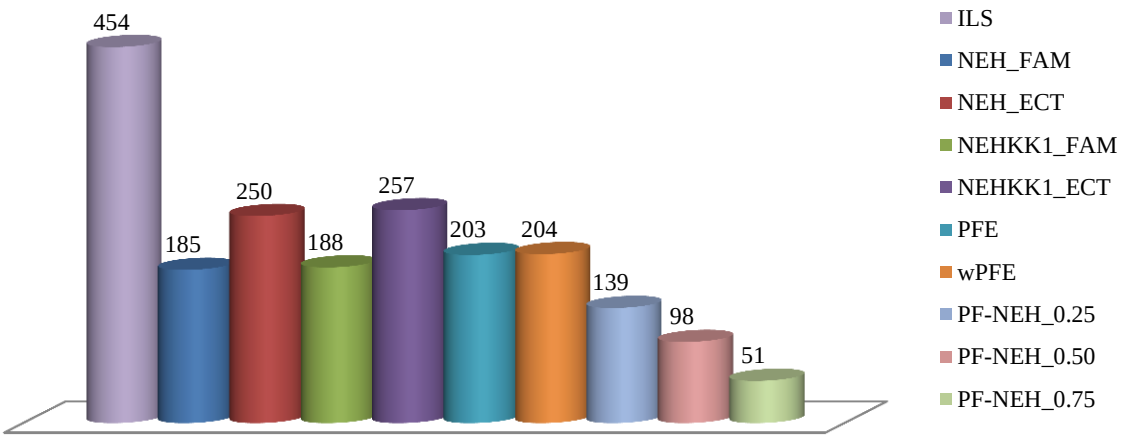
**Tabela 6.4 – RPD médio do ILS sobre o conjunto de teste A (Instâncias com solução ótima conhecida)**

| $n \times m$ | ILS | $NEH_{FAM}$ | $NEH_{ECT}$ | $NEHKK1_{FAM}$ | $NEHKK1_{ECT}$ | PFE | *w*PFE | $PF\text{-}NEH_{0.25}$ | $PF\text{-}NEH_{0.50}$ | $PF\text{-}NEH_{0.75}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 5×3 | 1,047 | 9,779 | 3,443 | 9,643 | 3,240 | 3,464 | 3,464 | 5,626 | 6,463 | 8,791 |
| 5×5 | 1,598 | 9,348 | 3,862 | 8,963 | 3,865 | 4,727 | 4,727 | 6,620 | 7,947 | 11,296 |
| 5×7 | 2,373 | 8,508 | 4,185 | 7,907 | 4,278 | 5,229 | 5,229 | 6,757 | 8,704 | 11,104 |
| 7×3 | 1,674 | 10,573 | 4,607 | 11,224 | 4,425 | 5,720 | 5,622 | 6,809 | 7,822 | 13,775 |
| 7×5 | 2,305 | 10,998 | 6,047 | 10,923 | 5,553 | 6,630 | 6,564 | 6,898 | 9,250 | 16,187 |
| 7×7 | 3,041 | 10,539 | 6,200 | 10,933 | 6,572 | 7,477 | 7,426 | 8,659 | 10,841 | 17,518 |
| 9×3 | 2,439 | 15,784 | 6,660 | 15,826 | 6,027 | 8,649 | 8,827 | 9,192 | 10,926 | 15,287 |
| 9×5 | 2,757 | 12,823 | 7,280 | 13,131 | 7,020 | 8,784 | 8,603 | 9,414 | 11,802 | 16,474 |
| 9×7 | 3,514 | 8,211 | 6,903 | 8,888 | 7,191 | 8,188 | 8,288 | 9,616 | 12,317 | 18,417 |
| 11×3 | 2,654 | 9,422 | 6,411 | 9,473 | 5,639 | 8,784 | 8,726 | 8,723 | 12,647 | 18,366 |
| 11×5 | 1,459 | 4,882 | 5,158 | 6,234 | 5,284 | 10,272 | 10,377 | 9,578 | 10,806 | 18,207 |
| 11×7 | 2,546 | 4,946 | 4,946 | 4,112 | 5,760 | 7,660 | 7,660 | 4,356 | 3,738 | 14,099 |
| **Média** | **2,155** | **10,642** | **5,241** | **10,684** | **5,068** | **6,398** | **6,383** | **7,505** | **9,306** | **14,003** |

Lembre-se que o critério de terminação do ILS é o tempo computacional máximo permitido, o qual depende diretamente da instância a ser resolvida. Desta forma, o tempo computacional máximo permitido para o ILS é de 0,625 e 5,775 segundos para as instâncias de maior e menor tamanho, respectivamente. Isto indica que em curtos tempos computacionais o ILS é capaz de encontrar soluções com um desvio médio de 2,155% da solução ótima.

Outro aspecto importante na análise do ILS é o número de soluções ótimas que é capaz de encontrar. Este resultado é ilustrado na Figura 6.8, na qual mostram-se o número de instâncias resolvidas até otimalidade para cada método de solução.

**Figura 6.8 – Número de soluções ótimas encontradas pelo ILS - Conjunto de teste A**




Como observado, o ILS encontra 76,65% e 81,60% mais soluções ótimas que as heurísticas $NEHKK1_{ECT}$ e $NEH_{ECT}$, respectivamente. Este resultado indica que além de encontrar soluções de melhor qualidade, o ILS também é capaz de encontrar um maior número de soluções ótimas através da busca no espaço dos ótimos locais.

Por outro lado, quando comparado com as 1.166 instâncias resolvidas até otimalidade pelo modelo de programação inteira mista, o ILS somente foi capaz de resolver o 38,94% das instâncias até otimalidade. Porém, este aspecto é compensado pelo fato que, em menores tempos computacionais, o ILS atinge soluções com um *gap* médio ao redor de 2%.

Como esperado, os resultados não variam significativamente quando analisamos o subconjunto de instâncias para as quais somente uma solução factível é conhecida. O ILS se consolida como o melhor método de solução, encontrando soluções que são, em média, 10,63% melhores que aquelas encontradas pelo modelo de programação inteira mista.

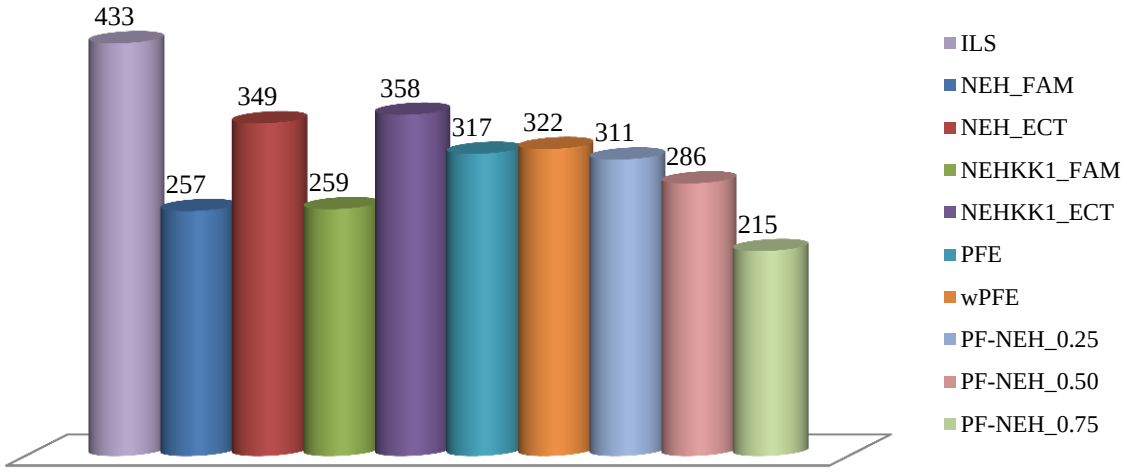
**Tabela 6.5 – RPD do ILS sobre o conjunto de teste A (Instâncias com solução factível conhecida)**

| $n \times m$ | ILS | $NEH_{ECT}$ | $NEH_{FAM}$ | $NEHKK1_{ECT}$ | $NEHKK1_{FAM}$ | PFE | $PF\text{-}NEH_{0.25}$ | $PF\text{-}NEH_{0.50}$ | $PF\text{-}NEH_{0.75}$ | *w*PFE |
|---|---|---|---|---|---|---|---|---|---|---|
| $7 \times 5$ | -2,510 | -2,674 | 2,533 | -2,510 | 4,072 | -0,536 | 1,149 | -0,211 | 7,186 | -0,536 |
| $7 \times 7$ | -6,447 | -2,881 | -2,095 | -3,599 | -2,320 | -3,920 | -3,042 | -1,076 | 2,599 | -4,409 |
| $9 \times 3$ | 1,051 | 1,867 | 15,225 | 4,303 | 13,840 | 4,713 | 5,961 | 2,817 | 9,570 | 4,713 |
| $9 \times 5$ | -7,632 | -2,868 | 1,901 | -3,423 | 2,477 | -3,024 | -2,530 | -1,122 | 2,892 | -3,027 |
| $9 \times 7$ | -9,799 | -6,430 | -0,221 | -6,414 | -0,125 | -5,311 | -4,688 | -2,806 | 1,253 | -5,200 |
| $11 \times 3$ | -4,841 | -0,605 | 12,813 | -1,100 | 13,305 | 1,279 | 2,151 | 3,600 | 8,683 | 1,031 |
| $11 \times 5$ | -13,719 | -9,613 | -2,776 | -10,144 | -3,427 | -8,827 | -8,900 | -6,603 | -1,655 | -8,937 |
| $11 \times 7$ | -15,805 | -12,400 | -6,607 | -12,596 | -6,597 | -11,455 | -10,632 | -8,457 | -3,052 | -11,329 |
| **Média** | -10,631 | -6,812 | 0,417 | -7,111 | 0,427 | -5,874 | -5,320 | -3,507 | 1,274 | -5,905 |

Os resultados para este subconjunto de instâncias são sumarizados pela Tabela 6.5. Os dados são consolidados para cada combinação de $n$ e $m$, de tal forma que cada célula da Tabela 6.5 representa a média sobre todos os demais fatores.

Por outro lado, a Figura 6.9 apresenta o número de instâncias nas quais cada método de solução foi capaz de encontrar uma melhor solução que o modelo de programação inteira mista.

**Figura 6.9 – Número de melhores soluções factíveis encontradas pelo ILS – Conjunto de teste A**

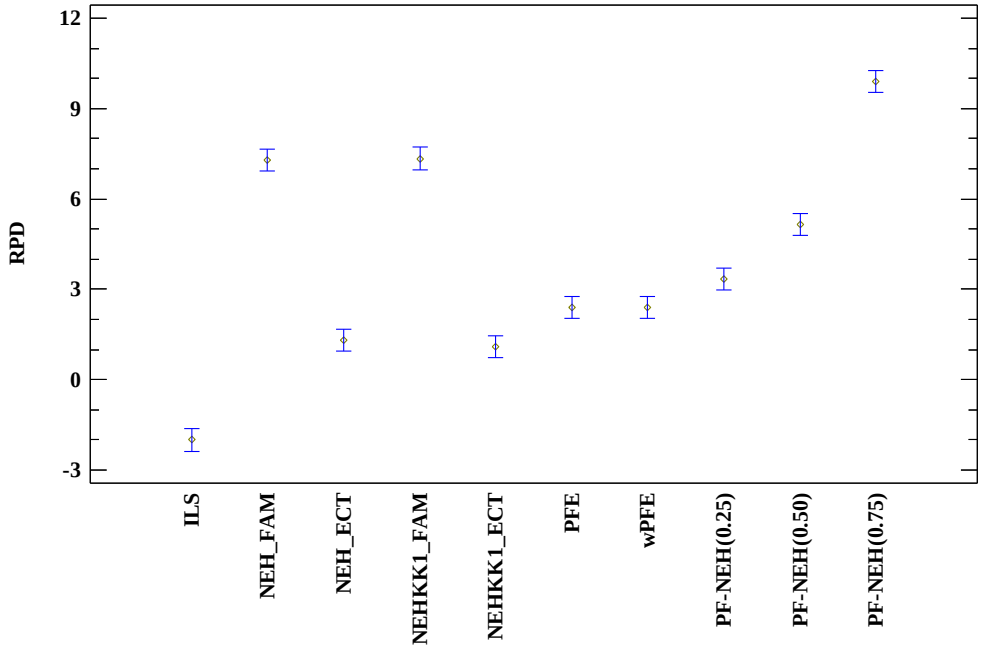



Observe que o ILS consegue melhorar 433 das 511 instâncias para as quais somente uma solução factível é conhecida. Isto implica que ainda há 42 instâncias para as quais o ILS não foi capaz de encontrar uma melhor solução que o modelo de programação inteira mista. Uma provável explicação desta situação é que para estas 42 instâncias o *solver* encontrou uma solução ótima, mas não foi capaz de provar a otimalidade das mesmas. Em qualquer caso, não há evidências objetivas que sustentem esta hipótese.

Para finalizar esta seção, é realizada uma ANOVA com o objetivo de identificar a existência de diferenças estatisticamente significativas entre o ILS e as demais heurísticas. Os resultados são ilustrados através do gráfico de médias e intervalos de confiança de 95% de Fisher LSD na Figura 6.10.

Como esperado, o ILS é significativamente melhor que as demais heurísticas. Este resultado indica que, mesmo para instâncias de entre 5 e 11 tarefas, o ILS é uma alternativa viável para encontrar soluções de alta qualidade em curtos tempos computacionais.

**Figura 6.10 – Gráfico de médias e intervalos de confiança de 95% de Fisher do ILS**




### 6.3.2. Avaliação Computacional do Conjunto de Teste B

Nesta seção são mostrados os resultados computacionais do ILS sobre o conjunto de teste B. Lembre-se que para este conjunto de instâncias o modelo de programação inteira mista mostra-se inviável computacionalmente, devido a requerimentos de memória e tempo computacional. Por esta razão o ILS simplesmente é comparado com o desempenho das heurísticas.

Tal como feito na seção 5.7.1, cada instância é resolvida cinco vezes e o valor do *makespan* do ILS resulta de tomar a média destas cinco réplicas. Para este conjunto de instâncias o RPD calcula-se como:

$$RPD = \left( \frac{Heu_{Sol} - Min_{Sol}}{Min_{Sol}} \right) \cdot 100$$

em que $Heu_{Sol}$ representa o *makespan* do ILS e $Min_{Sol}$ representa a melhor solução conhecida para uma instância dada.

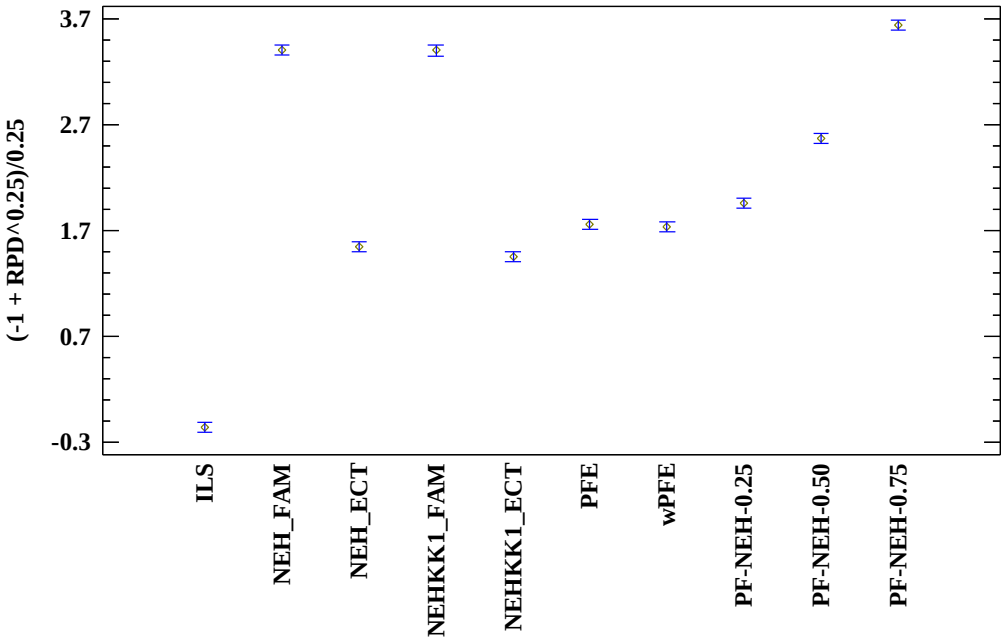
A Tabela 6.6 mostra o RPD médio para cada método de solução, consolidando os resultados para cada combinação de $n$ e $m$. Como observado, o ILS atinge resultados significativamente melhores que os resultados atingidos pelas heurísticas. Isto implica que o ILS, através da aplicação iterada de operadores de perturbação e busca local, é capaz de encontrar soluções significativamente melhores que aquelas geradas através de simples heurísticas construtivas.

**Tabela 6.6 – RPD do ILS sobre o conjunto de teste B**

| $n \times m$ | ILS | $NEH_{FAM}$ | $NEH_{ECT}$ | $NEHKK1_{FAM}$ | $NEHKK1_{ECT}$ | PFE | *w*PFE | $PF\text{-}NEH_{0.25}$ | $PF\text{-}NEH_{0.50}$ | $PF\text{-}NEH_{0.75}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $50 \times 7$ | 1,025 | 17,881 | 4,962 | 17,813 | 4,372 | 5,494 | 5,636 | 6,206 | 8,390 | 14,893 |
| $50 \times 9$ | 1,065 | 16,574 | 4,881 | 16,642 | 4,657 | 5,699 | 5,427 | 6,042 | 8,273 | 13,873 |
| $50 \times 11$ | 1,038 | 13,880 | 4,373 | 13,805 | 4,167 | 4,957 | 5,103 | 5,786 | 8,315 | 14,735 |
| $100 \times 7$ | 0,845 | 16,460 | 3,441 | 16,413 | 3,136 | 4,315 | 4,174 | 4,693 | 6,917 | 12,857 |
| $100 \times 9$ | 0,823 | 16,279 | 3,302 | 16,107 | 3,098 | 4,084 | 3,893 | 4,604 | 6,547 | 12,546 |
| $100 \times 11$ | 0,839 | 15,123 | 3,420 | 14,930 | 3,131 | 4,197 | 4,150 | 4,720 | 6,789 | 12,852 |
| **Média** | **0,939** | **16,033** | **4,063** | **15,952** | **3,760** | **4,791** | **4,730** | **5,342** | **7,538** | **13,626** |

Por fim, uma ANOVA é realizada para confirmar os resultados apresentados na Tabela 6.6. Os resultados do teste são ilustrados através da gráfica de médias e intervalos de confiança de 95% de Fisher LSD na Figura 6.11.

**Figura 6.11 – Gráfico de médias e intervalos de confiança de 95% de Fisher do ILS - Conjunto de teste B**




A partir da Figura 6.11 observa-se claramente que a diferença entre o ILS e os demais métodos é significativa. Também, é possível observar que a diferença entre o ILS e as heurísticas neste conjunto de instâncias é consideravelmente maior que a diferença observada no conjunto de teste A. Deste modo, é possível concluir que quando as instâncias a serem resolvidas são relativamente pequenas, tanto as heurísticas quanto o ILS são capazes de encontrar soluções de alta qualidade. Isto não implica que as diferenças observadas não são significativas, mas simplesmente que as heurísticas também são uma alternativa a levar em conta quando a instância a ser resolvida é pequena.

Por outro lado, quando o tamanho da instância aumenta o desempenho das heurísticas diminui significativamente e, consequentemente, a diferença de qualidade entre as heurísticas e o ILS é mais notória.

Em relação ao tempo computacional, é evidente que o ILS precisa de um maior esforço computacional que as heurísticas, mas este é compensado pela qualidade das soluções encontradas. Especificamente, o tempo computacional do ILS oscilou entre

22,50 e 120,21 segundos, o qual é ainda um tempo computacional aceitável para problemas de programação da produção.

Nesta seção foi apresentado e avaliado computacionalmente um algoritmo do tipo ILS. A calibração do ILS foi feita através de técnicas estatísticas, como delineamento experimental e ANOVA, que permitiram identificar e configurar os parâmetros de maior influencia no desempenho do ILS.

O ILS proposto foi avaliado em dois conjuntos de testes diferentes. Em ambos os conjuntos o ILS mostrou os melhores resultados, mas evidenciou que seu desempenho é consideravelmente melhor em instâncias de entre 50 e 100 tarefas, nas quais heurísticas e modelos de programação matemática não apresentam bons resultados. Em instâncias de entre 5 e 11 tarefas as diferenças são significativas, porém menos notórias devido a que o modelo de programação inteira e as heurísticas apresentaram bons resultados.

O tempo computacional do ILS foi configurado com um tempo máximo de execução que depende do problema a ser resolvido. Este critério de terminação é cada vez mais comum na literatura, pois garante que instâncias de maior tamanho tenham mais tempo para serem resolvidas. Por outro lado, este critério de terminação está em conformidade aos requerimentos práticos da programação da produção, na qual programas de produção de alta qualidade devem ser gerados em curtos tempos computacionais.

# 7. Conclusões e Perspectivas Futuras

Nesta dissertação estudou-se o problema de programação da produção em sistemas *Flowshop* Híbrido com máquinas paralelas não relacionadas, tempos de preparação dependentes da sequência e da máquina (tanto antecipatórios quanto não antecipatórios), capacidade de armazenagem intermediária limitada, tempos de transporte entre estações, tempos de liberação e elegibilidade de máquinas. Como critério de otimização considerou-se o *makespan*, cuja minimização está diretamente relacionada com a utilização eficiente dos recursos de produção.

O estudo deste problema visou tratar diretamente um dos mais recentes requerimentos da literatura da programação da produção: estudar problemas mais realistas e complexos que considerem múltiplas restrições de forma conjunta com o objetivo de diminuir o *gap* entre literatura e prática. Desta forma, a principal contribuição desta dissertação foi o estudo de um problema com considerações realistas e de ampla relevância em diversos setores industriais, começando com sua modelação como um problema de programação inteira mista e, em seguida, propondo métodos de solução capazes de obter soluções de boa qualidade em curtos tempos computacionais.

No Capítulo 4, um modelo de programação inteira mista foi proposto e avaliado sobre um extenso conjunto de instâncias de entre 9 e 11 tarefas, as quais representam as diferentes características do problema estudado. O modelo foi implementado na linguagem de programação matemática GAMS e resolvido através do *solver* comercial CPLEX com um tempo máximo de execução de 3.600 segundos. Os resultados da avaliação computacional indicaram que o modelo de programação inteira mista é viável somente para a resolução de instâncias de até nove tarefas e cinco estações, pois para instâncias além deste tamanho, o tempo computacional requerido torna-se proibitivo.

Estes resultados motivaram a adaptação e implementação de métodos heurísticos capazes de gerar soluções de boa qualidade em tempos computacionais curtos. O Capítulo 5 apresentou as diversas heurísticas que foram adaptadas, assim como os resultados da avaliação computacional sobre dois conjuntos de instâncias diferentes.

A avaliação sobre o primeiro conjunto de instâncias permitiu a comparação direta entre as heurísticas e modelo de programação inteira mista. Os resultados obtidos permitiram identificar que as heurísticas $NEHKK1_{ECT}$ e $NEH_{ECT}$ tiveram um bom desempenho, encontrando soluções com um desvio médio de 5% em relação à solução

ótima. Por outro lado, quando somente uma solução factível é conhecida, as heurísticas $NEHKK1_{ECT}$ e $NEH_{ECT}$ foram capazes de gerar soluções de maior qualidade, superando as soluções encontradas pelo *solver*. Os resultados do segundo conjunto de instâncias, conformado por problemas de entre 50 e 100 tarefas, indicaram que a heurística $NEHKK1_{ECT}$ foi capaz de gerar soluções significativamente melhores que as soluções geradas pelas demais heurísticas.

Finalmente, com o objetivo de encontrar soluções ainda melhores, o Capítulo 6 apresentou um algoritmo de busca local iterada (ILS). Esta é uma meta-heurística simples e eficiente, composta por quatro elementos básicos: uma solução inicial, um operador de busca local, um operador de perturbação e um critério de aceitação. Devido aos poucos parâmetros requeridos, o tempo de implementação e configuração desta meta-heurística é notavelmente reduzido, o qual motivou sua utilização.

Baseados em resultados da literatura, uma solução factível representa-se através de uma permutação de tarefas, enquanto o programa de produção completo é gerado através da regra de designação ECT, a qual designa uma determinada tarefa ao processador que é capaz de finalizá-la no tempo mais cedo. Para a parametrização dos demais operadores, diversos experimentos cuidadosamente planejados foram executados. Os resultados foram analisados através da técnica estatística ANOVA, a qual permite identificar o efeito gerado por cada elemento do algoritmo sobre uma determinada variável de resposta. Como resultado destes experimentos, o melhor operador para cada componente do ILS foi identificado e utilizado na configuração final do algoritmo.

O ILS proposto foi avaliado em dois conjuntos de testes diferentes. Em ambos os conjuntos o ILS mostrou os melhores resultados, mas evidenciou que seu desempenho é consideravelmente melhor em instâncias de entre 50 e 100 tarefas, nas quais heurísticas e modelos de programação matemática não apresentam bons resultados. Em instâncias de até 11 tarefas as diferenças são significativas, porém menos notórias, pois o modelo de programação inteira e as heurísticas também apresentaram bons desempenhos.

Em relação ao tempo computacional, o ILS apresentou maior esforço que as heurísticas, mas este foi compensado pela qualidade das soluções atingidas. Por outro lado, o tempo computacional do ILS é ainda aceitável, ficando em torno de 120 segundos para instância de maior tamanho, o que viabiliza seu uso em ambientes de

produção reais. Em qualquer caso, o esforço computacional do ILS é flexível e facilmente adaptável. Assim, dependendo da situação, o tempo computacional pode ser aumentado ou diminuído através de modificações no critério de finalização do algoritmo, o qual estabelece o tempo máximo de execução permitido.

Como pesquisa futura, várias ideias interessantes podem ser exploradas em profundidade. Inicialmente, o problema pode ser estendido para considerar múltiplos objetivos, tornando-o ainda mais realista. Note, por exemplo, que a minimização do *makespan* visa à utilização eficiente dos recursos de produção, porém poderia causar atraso no atendimento das datas de entregas prometidas aos clientes. Dado que em ambientes de produção reais ambos os objetivos são importantes, é necessário resolver o problema com múltiplos objetivos, gerando soluções que representem um compromisso entre um e outro objetivo.

Seguindo com o mesmo problema, o desempenho do ILS poderia ser aprimorado através do uso de algoritmos de busca mais sofisticados, como busca tabu ou recozimento simulado. Estes métodos provavelmente apresentariam um maior custo computacional, mas a qualidade da solução encontrada também deveria ser melhor. Por outro lado, experimentação minuciosa do critério de aceitação *LSMC* pode ser feita com o objetivo de estabelecer um esquema adequado para diminuição da temperatura. Uma adequada parametrização deste operador deveria permitir encontrar melhores soluções que aquelas encontradas com o critério *Better*, o qual tende a ficar preso em ótimos locais fortes.

O desenvolvimento de métodos de solução alternativos também é uma boa direção de pesquisa futura. Desta forma, meta-heurísticas mais clássicas, como algoritmos genéticos, busca tabu, recozimento simulado, entre outras, podem ser desenvolvidas e comparadas com o ILS proposto. Em geral estas meta-heurísticas possuem mais parâmetros a serem configurados e, portanto um maior esforço de codificação e parametrização deverá ser feito.

## Referências


AGNETIS, A. et al. Scheduling of flexible flow lines in an automobile assembly plant. **European Journal of Operational Research**, Amsterdam, v. 97, n. 2, p. 348-362, mar. 1997.

AHUJA, R. K. et al. A survey of very large-scale neighborhood search techniques. **Discrete Applied Mathematics**, Amsterdam, v. 123, n. 1, p. 75-102, 2002.

ALLAHVERDI, A. et al. A survey of scheduling problems with setup times or costs. **European Journal of Operational Research**, Amsterdam, v. 187, n. 3, p. 985-1032, 2008.

ALLAHVERDI, A.; GUPTA, J. N.; ALDOWAISAN, T. A review of scheduling research involving setup considerations. **International Journal of Management Science**, Oxford, v. 27, n. 2, p. 219-239, abr. 1999.

ALLAHVERDI, A.; SOROUSH, H. M. The significance of reducing setup times/setup costs. **European Journal of Operational Research**, Amsterdam, v. 187, n. 3, p. 978-984, 16 jun. 2008.

ATTAR, S. F.; MOHAMMADI, M.; TAVAKKOLI-MOGHADDAM, R. Hybrid flexible flowshop scheduling problem with unrelated parallel machines and limited waiting times. **The International Journal of Advanced Manufacturing Technology**, London, abr. 2013. Disponível em: <http://link.springer.com/content/pdf/10.1007%2Fs00170-013-4956-3.pdf>. Acesso em: 30 maio 2013.

BARR, R. S. et al. Designing and reporting on computational experiments with heuristic methods. **Journal of Heuristics**, Boston, v. 1, n. 1, p. 9-32, set. 1995.

BEHNAMIAN, J.; FATEMI GHOMI, S. M. T. Hybrid flowshop scheduling with machine and resource-dependent processing times. **Applied Mathematical Modelling**, Guildford, v. 35, n. 3, p. 1107-1123, mar. 2011.

BEHNAMIAN, J.; FATEMI GHOMI, S. M. T.; ZANDIEH, M. Hybrid flowshop scheduling with sequence-dependent setup times by hybridizing max-min ant system, simulated annealing and variable neighbourhood search. **Expert Systems**, [S.l], v. 29, n. 2, p. 156-169, dez. 2012.

BERTEL, S.; BILLAUT, J. C. A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. **European Journal of Operational Research**, Amsterdam, v. 159, n. 3, p. 651-662, dez. 2004.

BERTRAND, J. W. M.; FRANSOO, J. C. Operations management research methodologies using quantitative modeling. **International Journal of Operations & Production Management**, Bradford, v. 22, n. 2, p. 241-264, 2002.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. **ACM Computing Survey**, New York, v. 35, n. 3, p. 268-308, 2003.

BOX, G. E. P.; COX, D. R. An analysis of transformations. **Journal of the Royal Statistical Society**, London, v. 26, n. 2, p. 211-252, 1964.

CAMPBELL, H. G.; DUDEK, R. A.; SMITH, M. L. A heuristic algorithm for the n job, m machine sequencing problem. **Management Science**, Providence, v. 16, n. 10, p. B-630-B-637, 1 jun. 1970.

CHEN, C.-L.; CHEN, C.-L. A bottleneck-based heuristic for minimizing makespan in a flexible flow line with unrelated parallel machines. **Computers & Operations Research**, Oxford, v. 36, n. 11, p. 3073-3081, nov. 2009.

CONWAY, R. W.; MAXWELL, W. L.; MILLER, L. W. **Theory of scheduling**. [S.l.]: Addison-Wesley, 1967. p. 294

DEAL, D. E.; YANG, T.; HALLQUIST, S. Job scheduling in petrochemical production: two-stage processing with finite intermediate storage. **Computers and Chemical Engineering**, New York, v. 18, n. 4, p. 333-344, 1994.

DEN BESTEN, M.; STÜTZLE, T.; DORIGO, M. Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In: BOERS, E. J. W. et al. (Ed.). **Proceedings of evo workshops, LNCS**. Berlin: Springer, 2001. p. 441-451.

FRAMINAN, J. M.; LEISTEN, R.; RAJENDRAN, C. Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. **International Journal of Production Research**, London, v. 41, n. 1, p. 121-148, jan. 2003.

GEIGER, M. J. Decision support for multi-objective flow shop scheduling by the pareto iterated local search methodology. **Computers & Industrial Engineering**, New York, v. 61, n. 3, p. 805-812, out. 2011.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers & Operations Research**, Oxford, v. 13, n. 5, p. 533-549, 1986.

GLOVER, F.; KOCHENBERGER, G. A. **Handbook of metaheuristics (international series in operations research & management science)**. Berlin: Springer, 2003. p. 556

GÓMEZ-GASQUET, P.; ANDRÉS, C.; LARIO, F. C. An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan. **Expert Systems with Applications**, New York, v. 39, n. 9, p. 8095-8107, jul. 2012.

GRABOWSKI, J.; PEMPERA, J. Sequencing of jobs in some production system. **European Journal of Operational Research**, Amsterdam, v. 125, n. 3, p. 535-550, set. 2000.

GRAHAM, R. et al. Optimization and approximation in deterministic sequencing and scheduling: a survey. **Annals of Discrete Mathematics**, Amsterdam, v. 5, p. 287-326, 1979.

HAKIMZADEH ABYANEH, S.; ZANDIEH, M. Bi-objective hybrid flow shop scheduling with sequence-dependent setup times and limited buffers. **The International Journal of Advanced Manufacturing Technology**, London, v. 58, n. 1-4, p. 309-325, set. 2012.

JIN, Z. H. et al. Scheduling hybrid flowshops in printed circuit board assembly lines. **Production and Operations Management**, Baltimore, v. 11, p. 216-230, 2002.

JUNGWATTANAKIT, J. et al. Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. **The International Journal of Advanced Manufacturing Technology**, London, v. 37, n. 3-4, p. 354-370, mar. 2008.

JUNGWATTANAKIT, J. et al. A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. **Computers & Operations Research**, Oxford, v. 36, n. 2, p. 358-378, fev. 2009.

KALCZYNSKI, P. J.; KAMBUROWSKI, J. An improved neh heuristic to minimize makespan in permutation flow shops. **Computers & Operations Research**, Oxford, v. 35, n. 9, p. 3001-3008, set. 2008.

KARACAPILIDIS, N. I.; PAPPIS, C. P. Production planning and control in textile industry: a case study. **Computers in Industry**, Amsterdam, v. 30, n. 2, p. 127-144, set. 1996.

KIS, T.; PESCH, E. A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. **European Journal of Operational Research**, Amsterdam, v. 164, n. 3, p. 592-608, ago. 2005.

KOCHHAR, S.; MORRIS, R. J. T. Heuristic methods for flexible flow line scheduling. **Journal of Manufacturing Systems**, Dearborn, v. 6, n. 4, p. 299-314, 1987.

KOCHHAR, S.; MORRIS, R. J. T.; WONG, W. S. The local search approach to flexible flow line scheduling. **Engineering Costs and Production Economics**, Amsterdam, v. 14, n. 1, p. 25-37, maio. 1988.

KURZ, M. E.; ASKIN, R. G. Comparing scheduling rules for flexible flow lines. **International Journal of Production Economics**, Amsterdam, v. 85, n. 3, p. 371-388, set. 2003.

KURZ, M. E.; ASKIN, R. G. Scheduling flexible flow lines with sequence-dependent setup times. **European Journal of Operational Research**, Amsterdam, v. 159, n. 1, p. 66-82, nov. 2004.

LEUNG, J. Y.-T. (Ed.). **Handbook of scheduling algorithms, models, and performance analysis**. [S.l.]: CRC Press, 2004. p. 1120

LIN, H.-T.; LIAO, C.-J. A case study in a two-stage hybrid flow shop with setup time and dedicated machines. **International Journal of Production Economics**, Amsterdam, v. 86, n. 2, p. 133-143, nov. 2003.

LIU, C.-Y.; CHANG, S.-C. Scheduling flexible flow shops with sequence-dependent setup effects. **IEEE Transactions on Robotics and Automation**, New York, v. 16, n. 4, p. 408-419, 2000.

LIU, J.; REEVES, C. R. Constructive and composite heuristic solutions to the P//∑Ci scheduling problem. **European Journal of Operational Research**, Amsterdam, v. 132, n. 2, p. 439-452, jul. 2001.

LIU, Y.; KARIMI, I. A. Scheduling multistage batch plants with parallel units and no interstage storage. **Computers & Chemical Engineering**, New York, v. 32, n. 4-5, p. 671-693, abr. 2008.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. In: GENDREAU, M.; POTVIN, J.-Y. (Ed.). **Handbook of Metaheuristics**. Boston, MA: Springer US, 2010, v. 146, p. 363-397

LOW, C. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. **Computers & Operations Research**, Oxford, v. 32, n. 8, p. 2013-2025, ago. 2005.

LOW, C.; HSU, C.-J.; SU, C.-T. A two-stage hybrid flowshop scheduling problem with a function constraint and unrelated alternative machines. **Computers & Operations Research**, Oxford, v. 35, n. 3, p. 845-853, mar. 2008.

MCCORMICK, S. T. et al. Sequencing in an assembly line with blocking to minimize cycle time. **Operations Research**, Baltimore, v. 37, n. 6, p. 925-935, 1989.

MONTGOMERY, D. C. **Diseño y análisis de experimentos**. 2. ed. México, DF: Limusa Wiley, 2005. p. 686

MONTGOMERY, D. C.; RUNGER, G. C. **Applied statistics and probability for engineers**. 3. ed. New York: John Wiley & Sons, Inc., 2003. p. 720

NADERI, B. et al. An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. **Expert Systems with Applications**, New York, v. 36, n. 6, p. 9625-9633, ago. 2009.

NADERI, B.; RUIZ, R.; ZANDIEH, M. Algorithms for a realistic variant of flowshop scheduling. **Computers & Operations Research**, Oxford, v. 37, p. 236-246, 2010.

NADERI, B.; ZANDIEH, M.; ROSHANAEI, V. Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. **The International Journal of Advanced Manufacturing Technology**, London,v. 41, n. 11-12, p. 1186-1198, 16 jul. 2008.

NADERI, B.; ZANDIEH, M.; SHIRAZI, M. A. H. A. Modeling and scheduling a case of flexible flowshops: total weighted tardiness minimization. **Computers & Industrial Engineering**, New York, v. 57, n. 4, p. 1258-1267, nov. 2009.

NAWAZ, M.; ENSCORE, E.; HAM, I. A heuristic algorithm for the m-machine , n-job flow-shop sequencing problem. **International Journal of Management Science**, Oxford, v. 11, n. 1, p. 91-95, 1983.

NICHOLSON, T. A. J. **Optimization in industry**: optimization techniques. London: Longman Press, 1971.

OĞUZ, C.; LIN, B. M. T.; EDWIN CHENG, T. C. Two-stage flowshop scheduling with a common second-stage machine. **Computers & Operations Research**, Oxford, v. 24, n. 12, p. 1169-1174, dez. 1997.

OSMAN, I.; POTTS, C. Simulated annealing for permutation flow-shop scheduling. **Omega**, Elmsford, v. 17, n. 6, p. 551-557, jan. 1989.

PAN, Q.-K.; RUIZ, R. Local search methods for the flowshop scheduling problem with flowtime minimization. **European Journal of Operational Research**, Amsterdam, v. 222, n. 1, p. 31-43, out. 2012.

PAN, Q.-K.; WANG, L. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. **Omega**, Elmsford, v. 40, n. 2, p. 218-229, abr. 2012.

PINEDO, M. L. **Planning and scheduling in manufacturing and services**. New York: Springer, 2005. p. 506

PINEDO, M. L. **Scheduling:** theory, algorithms, and systems. 3. ed. New York: Springer, 2008. p. 671

QUADT, D.; KUHN, H. A taxonomy of flexible flow line scheduling procedures. **European Journal of Operational Research**, Amsterdam, v. 178, n. 3, p. 686-698, maio 2007.

RAJENDRAN, C.; ZIEGLER, H. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. **European Journal of Operational Research**, Amsterdam, v. 103, n. 1, p. 129-138, 1997.

RASHIDI, E.; JAHANDAR, M.; ZANDIEH, M. An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. **The International Journal of Advanced Manufacturing Technology**, London, v. 49, n. 9-12, p. 1129-1139, fev. 2010.

RIBAS, I.; LEISTEN, R.; FRAMIÑAN, J. M. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. **Computers & Operations Research**, Oxford, v. 37, n. 8, p. 1439-1454, ago. 2010.

RONCONI, D. P. A note on constructive heuristics for the flowshop problem with blocking. **International Journal of Production Economics**, Amsterdam, v. 87, n. 1, p. 39-48, jan. 2004.

RUIZ, R.; MAROTO, C. A comprehensive review and evaluation of permutation flowshop heuristics. **European Journal of Operational Research**, Amsterdam, v. 165, n. 2, p. 479-494, set. 2005.

RUIZ, R.; MAROTO, C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. **European Journal of Operational Research**, Amsterdam, v. 169, n. 3, p. 781-800, 16 mar. 2006.

RUIZ, R.; MAROTO, C.; ALCARAZ, J. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. **European Journal of Operational Research**, Amsterdam, v. 165, n. 1, p. 34-54, 16 ago. 2005.

RUIZ, R.; ŞERIFOĞLU, F.; URLINGS, T. Modeling realistic hybrid flexible flowshop scheduling problems. **Computers & Operations Research**, Oxford, v. 35, n. 4, p. 1151-1175, abr. 2008.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European Journal of Operational Research**, Amsterdam, v. 177, n. 3, p. 2033-2049, mar. 2007.

RUIZ, R.; STÜTZLE, T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. **European Journal of Operational Research**, Amsterdam, v. 187, n. 3, p. 1143-1159, jun. 2008.

RUIZ, R.; VÁZQUEZ-RODRÍGUEZ, J. A. The hybrid flow shop scheduling problem. **European Journal of Operational Research**, Amsterdam, v. 205, n. 1, p. 1-18, ago. 2010.

SACCHI, L. H. **Algoritmos genéticos para minimização de makespan em um flow shop flexível**. 1997. 127 f. Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 1997.

SANTOS, D. L.; HUNSUCKER, J. L.; DEAL, D. E. FLOWMULT: Permutation sequences for flow shops with multiple processors. **Journal of Information and Optimization Sciences**, [S.l], v. 16, n. 2, p. 351-366, 1995.

SAWIK, T. Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. **Mathematical and Computer Modelling**, Oxford, v. 31, n. 13, p. 39-52, jun. 2000.

SAWIK, T. An exact approach for batch scheduling in flexible flow lines with limited intermediate buffers. **Mathematical and Computer Modelling**, Oxford, v. 36, n. 4-5, p. 461-471, set. 2002.

STÜTZLE, T. Applying iterated local search to the permutation flow shop problem. **Technical Report**, Darmstadt, 1998. Disponível em: < http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.4045&rep=rep1&type=pdf >. Acesso em: 14 jul. 2013.

T´KINDT, V.; BILLAUT, J.-C. **Multicriteria scheduling:** theory, models and algorithms. 2. ed. Berlin: Springer, 2006. p. 376

TAILLARD, E. Some efficient heuristic methods for the flow shop sequencing problem. **European Journal of Operational Research**, Amsterdam, v. 47, n. Nov. 1988, p. 65-74, 1990.

TAILLARD, E. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, Amsterdam, v. 64, p. 278-285, 1993.

TALBI, E.-G. **Metaheuristics:** from design to implementation. New Jersey: John Wiley & Sons, Inc., 2009. p. 500

TANG, L.; XUAN, H. Lagrangian relaxation algorithms for real-time hybrid flowshop scheduling with finite intermediate buffers. **Journal of the Operational Research Society**, Oxford, v. 57, n. 3, p. 316-324, 3 ago. 2005.

TAVAKKOLI-MOGHADDAM, R.; SAFAEI, N.; SASSANI, F. A memetic algorithm for the flexible flow line scheduling problem with processor blocking. **Computers & Operations Research**, Oxford, v. 36, n. 2, p. 402-414, fev. 2009.

THORNTON, H. W.; HUNSUCKER, J. L. A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage. **European Journal of Operational Research**, Amsterdam, v. 152, n. 1, p. 96-114, jan. 2004.

TRAN, T. H.; NG, K. M. A water-flow algorithm for flexible flow shop scheduling with intermediate buffers. **Journal of Scheduling**, New York, v. 14, n. 5, p. 483-500, 11 nov. 2011.

TSUBONE, H. et al. A production scheduling system for a hybrid flow shop—a case study. **Omega**, Elmsford, v. 21, n. 2, p. 205-214, mar. 1993.

URLINGS, T. **Heuristics and metaheuristics for heavily constrained hybrid flowshop problems**. 2010. 175 f. Tese (Doutorado em Estatística e Otimização) - Universidad Politénica de Valencia, Valencia, 2010.

URLINGS, T.; RUIZ, R.; ŞERIFOĞLU, F. S. Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems. **International Journal of Metaheuristics**, Geneva, v. 1, n. 1, p. 30-54, 2010.

URLINGS, T.; RUIZ, R.; STÜTZLE, T. Shifting representation search for hybrid flexible flowline problems. **European Journal of Operational Research**, Amsterdam, v. 207, p. 1086-1095, 2010.

VIGNIER, A.; BILLAUT, J.; PROUST, C. Hybrid flowshop scheduling problems: state of the art. **Rairo-Recherche Operationnelle=Operations Research**, Paris, v. 33, n. 2, p. 117-83, 1999.

WANG, H. Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions. **Expert Systems**, [S.l] ,v. 22, n. 2, p. 78-86, 2005.

WANG, X.; TANG, L. A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers. **Computers & Operations Research**, Oxford, v. 36, n. 3, p. 907-918, mar. 2009.

WARDONO, B.; FATHI, Y. A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. **European Journal of Operational Research**, Amsterdam, v. 155, n. 2, p. 380-401, jun. 2004.

YAURIMA, V.; BURTSEVA, L.; TCHERNYKH, A. Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. **Computers & Industrial Engineering**, New York, v. 56, n. 4, p. 1452-1463, maio. 2009.

ZANDIEH, M.; FATEMI GHOMI, S. M. T.; MOATTAR HUSSEINI, S. M. An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. **Applied Mathematics and Computation**, New York, v. 180, n. 1, p. 111-127, 1 set. 2006.

ZANDIEH, M.; MOZAFFARI, E.; GHOLAMI, M. A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems. **Journal of Intelligent Manufacturing**, New York, v. 21, n. 6, p. 731-743, 3 mar. 2009.

# A. Testes preliminares das regras de designação

O objetivo destes testes é analisar o desempenho das regras de designação FAM e ECT, descritas na seção 5.1. Com este objetivo, se implementaram as heurísticas CDS, NEH e NEHKK1 para cada regra de designação. As heurísticas implementadas foram são denominadas $CDS_{FAM}$, $CDS_{ECT}$, $NEH_{FAM}$, $NEH_{ECT}$, $NEHKK1_{FAM}$ e $NEHKK1_{ECT}$, tal que os subscritos FAM e ECT denotam a regra de designação com a qual a heurística foi implementada.

Todas as heurísticas foram implementadas na linguagem C++ e executadas em um *notebook* com processador Intel Core i5 2.67 GHz com 4 Gigabytes de memória *RAM*. Devido ao baixo custo computacional destas heurísticas, as instâncias do conjunto definido na seção 4.2 são utilizadas neste teste computacional. Para maior clareza, a Tabela A.1 apresenta novamente a descrição de tal conjunto de instâncias, o qual contém 1.728 problemas.

Para medir o desempenho das heurísticas, a percentagem de desvio relativo (RPD) sobre a melhor solução conhecida de cada instância é calculada.

$$RPD = \left( \frac{Heu_{Sol} - MIP_{Best}}{MIP_{Best}} \right) \cdot 100$$

em que $Heu_{Sol}$ é a solução encontrada por uma determinada heurística e $MIP_{Best}$ é a melhor solução encontrada pelo modelo de programação inteira mista da seção 4.1. Note que $MIP_{Best}$ não é conhecido para as 51 instâncias nas quais o modelo de programação inteira não foi capaz de encontrar uma solução factível. Portanto, estas 51 instâncias foram excluídas do teste.

**Tabela A.1 – Fatores utilizados na geração de instâncias para o modelo de programação inteira mista**

| Fatores | Símbolo | Valores |
|---|---|---|
| Número de tarefas | $n$ | 5, 7, 9, 11 |
| Número de estações | $m$ | 3, 5, 7 |
| Número de processadores paralelos em estações de processamento | $m_i$ | 2, 3 |
| Número de processadores paralelos em estações *buffer* | $b_i$ | $m_i/2, m_i$ |
| Probabilidade de um processador ser elegível | $PE_{ij}$ | 0.50, 1 |
| Distribuição dos tempos de preparação | $DS_{iljk}$ | $U[25,74], U[75,125]$ |
| Probabilidade do tempo de preparação ser antecipatório | $PA_{iljk}$ | 0, 0.50, 1 |

Os resultados do teste preliminar sobre este conjunto de instâncias são resumidos pela Tabela A.2.

**Tabela A.2 – Resultados dos testes preliminares das regras de designação FAM e ECT**

| $n \times m$ | $CDS_{ECT}$ | $CDS_{FAM}$ | $NEH_{ECT}$ | $NEH_{FAM}$ | $NEHKK1_{ECT}$ | $NEHKK1_{FAM}$ |
|---|---|---|---|---|---|---|
| 5×3 | 15,227 | 21,100 | 3,443 | 9,779 | 3,240 | 9,643 |
| 5×5 | 12,804 | 18,490 | 3,862 | 9,348 | 3,865 | 8,963 |
| 5×7 | 10,430 | 14,313 | 4,185 | 8,508 | 4,278 | 7,907 |
| 7×3 | 18,105 | 25,186 | 4,607 | 10,573 | 4,425 | 11,224 |
| 7×5 | 15,957 | 21,727 | 5,865 | 10,822 | 5,385 | 10,780 |
| 7×7 | 13,541 | 18,413 | 5,285 | 9,267 | 5,548 | 9,598 |
| 9×3 | 22,589 | 32,295 | 6,394 | 15,753 | 5,932 | 15,715 |
| 9×5 | 14,389 | 21,108 | 2,492 | 7,669 | 2,093 | 8,104 |
| 9×7 | 5,794 | 10,970 | -3,171 | 1,840 | -3,089 | 2,079 |
| 11×3 | 20,206 | 30,686 | 2,270 | 11,424 | 1,661 | 11,735 |
| 11×5 | 3,905 | 10,513 | -7,564 | -1,714 | -8,005 | -2,087 |
| 11×7 | -2,138 | 4,410 | -12,100 | -6,408 | -12,279 | -6,412 |
| **Média** | **12,567** | **19,101** | **1,297** | **7,238** | **1,088** | **7,271** |

Note que todas as heurísticas implementadas com a regra ECT tiveram melhor desempenho que seus respectivos pares com a regra FAM. Em cada caso, a regra ECT permitiu encontrar soluções de melhor qualidade que a regra FAM. Os valores negativos indicam que a heurística foi capaz de gerar uma melhor solução que aquela encontrada pelo modelo de programação inteira mista. É possível perceber que tais valores aparecem para as instâncias de maior tamanho, nas quais o *solver* teve maior dificuldade para encontrar soluções ótimas e o *gap* foi consideravelmente alto.

Como esperado, a heurística CDS mostrou o desempenho mais baixo. Porém o objetivo deste teste não é avaliar o desempenho das heurísticas, mas o desempenho das regras de designação e, portanto, nenhuma conclusão em relação ao desempenho das heurísticas é feita. Em qualquer caso, este teste já permite visualizar que as heurísticas NEH e NEHKK1 são superiores à heurística CDS. Adicionalmente, a heurística NEHKK1 é ligeiramente superior à heurística NEH quando ambas as heurísticas utilizam a regra ECT. No entanto, quando a regra FAM é utilizada, as diferencias entre as heurísticas não são significantes. Esta situação indica que ganhos no desempenho da heurística NEHKK1 podem ser obtidos através da regra ECT.

De acordo com estes resultados, a regra ECT é mais adequada que a regra FAM. Portanto, as demais heurísticas descritas no capítulo 5 serão implementadas somente com a regra de designação ECT.

# B. Tabelas ANOVA

Neste apêndice as tabelas da ANOVA são dadas. As tabelas estão organizadas pela ordem na qual aparecem ao longo do texto.

**Tabela B.1 – Análise de Variância das Heurísticas - Conjunto de teste A**

| Fonte | Soma de Quadrados | Graus de liberdade | Quadrado Médio | Razão-F | Valor-P |
|---|---|---|---|---|---|
| **Efeitos Principais** | | | | | |
| A: Tarefas | 8124,710 | 3 | 2708,240 | 575,170 | 0,000 |
| B: Estações | 5915,460 | 2 | 2957,730 | 628,160 | 0,000 |
| C: Processadores | 2864,440 | 1 | 2864,440 | 608,350 | 0,000 |
| D: Buffers | 437,734 | 1 | 437,734 | 92,970 | 0,000 |
| E: Heurísticas | 27314,800 | 12 | 2276,230 | 483,420 | 0,000 |
| **Interações** | | | | | |
| AB | 6414,610 | 6 | 1069,100 | 227,050 | 0,000 |
| AC | 1324,780 | 3 | 441,593 | 93,790 | 0,000 |
| AD | 965,689 | 3 | 321,896 | 68,360 | 0,000 |
| AE | 616,053 | 36 | 17,113 | 3,630 | 0,000 |
| BC | 237,454 | 2 | 118,727 | 25,220 | 0,000 |
| BD | 499,401 | 2 | 249,701 | 53,030 | 0,000 |
| BE | 905,809 | 24 | 37,742 | 8,020 | 0,000 |
| CD | 11,239 | 1 | 11,239 | 2,390 | 0,123 |
| CE | 1021,410 | 12 | 85,117 | 18,080 | 0,000 |
| DE | 188,290 | 12 | 15,691 | 3,330 | 0,000 |
| **Resíduos** | 2368,410 | 503 | 4,709 | | |
| Total (Corregido) | 59210,300 | 623 | | | |

Os valores P provam a significância estatística de cada fator. Dado que 14 valores P são menores que 0,05 ($\alpha = 5\%$), estes fatores tem um efeito estatisticamente significativo sobre a variável de resposta, RPD, com um nível de confiança de 95%.

**Tabela B.2 – Análise de Variância das Heurísticas - Conjunto de teste B**

| Fonte | Soma de Quadrados | Graus de liberdade | Quadrado Médio | Razão-F | Valor-P |
|---|---|---|---|---|---|
| **Efeitos Principais** | | | | | |
| A: Tarefas | 4,028 | 1 | 4,028 | 46,720 | 0,000 |
| B: Estações | 0,183 | 2 | 0,091 | 1,060 | 0,348 |
| C: Processadores | 414,672 | 2 | 207,336 | 2404,580 | 0,000 |
| D: Buffers | 1,047 | 2 | 0,523 | 6,070 | 0,003 |
| E: Heurística | 749,649 | 8 | 93,706 | 1086,760 | 0,000 |
| **Interações** | | | | | |
| AB | 0,355 | 2 | 0,177 | 2,060 | 0,129 |
| AC | 0,127 | 2 | 0,064 | 0,740 | 0,479 |
| AD | 0,757 | 2 | 0,379 | 4,390 | 0,013 |
| AE | 2,017 | 8 | 0,252 | 2,920 | 0,004 |
| BC | 4,467 | 4 | 1,117 | 12,950 | 0,000 |
| BD | 0,660 | 4 | 0,165 | 1,910 | 0,107 |
| BE | 0,820 | 16 | 0,051 | 0,590 | 0,889 |
| CD | 0,063 | 4 | 0,016 | 0,180 | 0,947 |
| CE | 277,655 | 16 | 17,353 | 201,260 | 0,000 |
| DE | 3,331 | 16 | 0,208 | 2,410 | 0,002 |
| **Resíduos** | 34,145 | 396 | 0,086 | | |
| Total (Corregido) | 1493,980 | 485 | | | |

Dado que 9 valores P são menores que 0,05 ($\alpha = 5\%$), estes fatores tem um efeito estatisticamente significativo sobre a variável de resposta, $\left(-1+RPD^{0.3}\right)/0.3$, com um nível de confiança de 95%.

Note que uma transformação da variável de resposta foi requerida para validar os pressupostos de normalidade, igualdade de variância e independência de resíduos.

**Tabela B.3 – Análise de Variância do ILS - Conjunto de teste A**

| Fonte | Soma de Quadrados | Graus de liberdade | Quadrado Médio | Razão-F | Valor-P |
|---|---|---|---|---|---|
| **Efeitos Principais** | | | | | |
| A: Tarefas | 7667,570 | 3 | 2555,860 | 765,390 | 0,000 |
| B: Estações | 2993,680 | 2 | 1496,840 | 448,250 | 0,000 |
| C: Processadores | 2125,550 | 1 | 2125,550 | 636,530 | 0,000 |
| D: Buffers | 581,691 | 1 | 581,691 | 174,200 | 0,000 |
| E: Heurísticas | 5519,570 | 9 | 613,285 | 183,660 | 0,000 |
| **Interações** | | | | | |
| AB | 4122,710 | 6 | 687,118 | 205,770 | 0,000 |
| AC | 930,599 | 3 | 310,200 | 92,890 | 0,000 |
| AD | 744,948 | 3 | 248,316 | 74,360 | 0,000 |
| AE | 180,102 | 27 | 6,670 | 2,000 | 0,003 |
| BC | 161,098 | 2 | 80,549 | 24,120 | 0,000 |
| BD | 193,321 | 2 | 96,660 | 28,950 | 0,000 |
| BE | 137,149 | 18 | 7,619 | 2,280 | 0,002 |
| CD | 26,039 | 1 | 26,039 | 7,800 | 0,006 |
| CE | 743,093 | 9 | 82,566 | 24,730 | 0,000 |
| DE | 45,313 | 9 | 5,035 | 1,510 | 0,143 |
| **Resíduos** | 1278,950 | 383 | 3,339 | | |
| Total (Corregido) | 27451,400 | 479 | | | |

Dado que 14 valores P são menores que 0,05 ($\alpha = 5\%$), estes fatores tem um efeito estatisticamente significativo sobre a variável de resposta, RPD, com um nível de confiança de 95%.

**Tabela B.4 – Análise de Variância do ILS - Conjunto de teste B**

| Fonte | Soma de Quadrados | Graus de liberdade | Quadrado Médio | Razão-F | Valor-P |
|---|---|---|---|---|---|
| **Efeitos Principais** | | | | | |
| A: Tarefas | 7,641 | 1 | 7,641 | 117,150 | 0,000 |
| B: Estações | 0,030 | 2 | 0,015 | 0,230 | 0,794 |
| C: Processadores | 374,502 | 2 | 187,251 | 2870,900 | 0,000 |
| D: Buffers | 0,023 | 2 | 0,011 | 0,170 | 0,840 |
| E: Heurísticas | 651,677 | 9 | 72,409 | 1110,160 | 0,000 |
| **Interações** | | | | | |
| AB | 1,179 | 2 | 0,589 | 9,040 | 0,000 |
| AC | 6,607 | 2 | 3,303 | 50,650 | 0,000 |
| AD | 0,611 | 2 | 0,306 | 4,680 | 0,010 |
| AE | 2,436 | 9 | 0,271 | 4,150 | 0,000 |
| BC | 5,375 | 4 | 1,344 | 20,600 | 0,000 |
| BD | 0,529 | 4 | 0,132 | 2,030 | 0,089 |
| BE | 0,456 | 18 | 0,025 | 0,390 | 0,990 |
| CD | 0,824 | 4 | 0,206 | 3,160 | 0,014 |
| CE | 149,582 | 18 | 8,310 | 127,410 | 0,000 |
| DE | 2,380 | 18 | 0,132 | 2,030 | 0,008 |
| **Resíduos** | 28,829 | 442 | 0,065 | | |
| Total (Corregido) | 1232,680 | 539 | | | |

Dado que 11 valores P são menores que 0,05 ($\alpha = 5\%$), estes fatores tem um efeito estatisticamente significativo sobre a variável de resposta, $\left(-1+RPD^{0.25}\right)/0.25$, com um nível de confiança de 95%.

## C. Testes preliminares do ILS

Testes preliminares do ILS são importantes já que permitem analisar o desempenho de certos operadores, assim como observar e estabelecer possíveis valores para parâmetros que são requeridos por alguns operadores do ILS. Através destes testes preliminares procura-se:

- Estabelecer o tipo de perturbação (Tamanho fixo vs. Tamanho dependente do problema).
- Determinar o valor do parâmetro *NumPert*, requerido pelo operador de perturbação *GL*.
- Determinar o valor de $\lambda$, requerido para o cálculo do parâmetro $T$ do critério de aceitação *LSMC*.

Nos testes preliminares, assim como na parametrização do ILS, é importante utilizar um conjunto de instâncias diferente às instâncias empregadas na avaliação final com o objetivo de evitar viés nos resultados. A Tabela C.1 apresenta os fatores considerados na geração das instâncias utilizadas tanto nos testes preliminares quanto na parametrização do ILS.

**Tabela C.1 – Fatores considerados na geração das instâncias de parametrização do ILS**

| Fatores | Símbolo | Valores |
|---|---|---|
| Número de tarefas | $n$ | 50, 100 |
| Número de estações | $m$ | 7, 9 |
| Número de processadores paralelos em estações de processamento | $m_i$ | 3, 4 |
| Número de processadores paralelos em estações *buffer* | $b_i$ | 3, 4 |
| Probabilidade de um processador ser elegível | $PE_{ij}$ | 0.5 |
| Distribuição dos tempos de preparação | $DS_{iljk}$ | $U[25,75]$ |
| Probabilidade do tempo de preparação ser antecipatório | $PA_{iljk}$ | 0.5 |

Como feito anteriormente, os tempos de processamento são fixados em $U[1,99]$, os tempos de liberação dos processadores é fixado em $U[1,200]$e os tempos de transporte entre estações é fixado em $U[1,10]$. De igual forma, a probabilidade de um processador ser elegível, a distribuição dos tempos de preparação e a probabilidade da preparação ser antecipatória são fixadas em 0.5, $U[25,75]$ e 0.5, respectivamente.

O número total de combinações dos fatores é $2 \cdot 2 \cdot 2 \cdot 2 = 2^4 = 16$. Para cada combinação cinco instâncias diferentes foram geradas. Desta forma, há um total de 80 instâncias.

Como medida de qualidade é utilizada a percentagem de melhoria obtida pelo ILS sobre a solução inicial gerada pela heurística $NEHKK1_{ECT}$:

$$Improvement = \left( \frac{C_{\max}(\pi_0) - C_{\max}(\pi)}{C_{\max}(\pi_0)} \right) \cdot 100$$

em que $C_{\max}(\pi_0)$ denota o *makespan* da solução inicial e $C_{\max}(\pi)$ denota o *makespan* da solução final obtida pelo ILS. Obviamente, maiores valores de *Improvement* são preferíveis.

Note que *Improvement* e RPD são medidas de qualidade complementares, pois atingir a maior percentagem de melhoria em relação à solução inicial é equivalente a atingir a menor percentagem de desvio relativo da melhor solução conhecida.

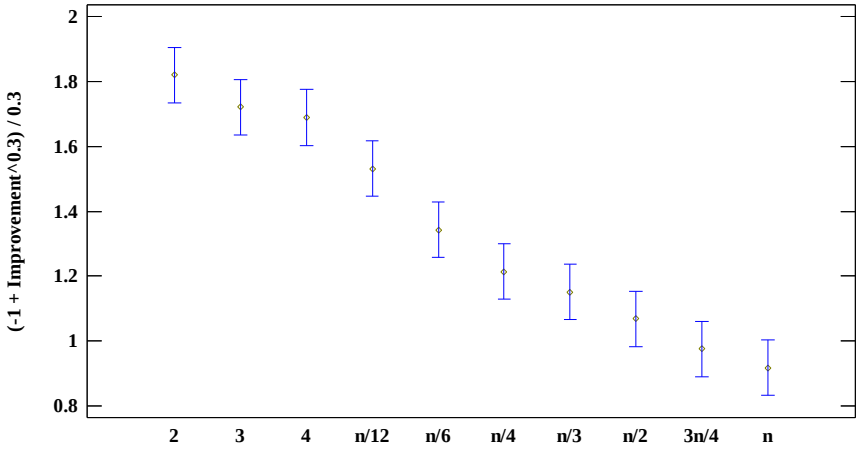
A primeira parte dos testes preliminares determina o melhor tipo de perturbação. A solução inicial do ILS é gerada pela heurística $NEHKK1_{ECT}$, a busca local é feita pelo operador *LSUrlings* e o critério de aceitação é *Better*. Como operador de perturbação utiliza-se um operador genérico, o qual seleciona e realoca *d* tarefas selecionadas aleatoriamente. Cada instância foi resolvida cinco vezes para cada valor de *d* e os resultados, para cada tamanho de instância, são mostrados na Tabela C.2. Observe que o tamanho da perturbação pode ser fixo ou dependente do número de tarefas a serem programadas.

Os resultados indicam que o desempenho do ILS diminui significativamente na medida em que o tamanho da perturbação aumenta. Adicionalmente, perturbações de tamanho fixo apresentam os melhores resultados. Isto indica que pequenas modificações da solução incumbente são suficientes para sair do ótimo local atual sem deteriorar significativamente a qualidade da solução. Pelo contrário, quando o tamanho da perturbação aumenta, a qualidade da solução é deteriorada consideravelmente e é provável encontrar um novo ótimo local de pior qualidade.

**Tabela C.2 – Resultados dos testes preliminares da perturbação**

| $n \times m$ | **2** | **3** | **4** | $n/12$ | $n/6$ | $n/4$ | $n/3$ | $n/2$ | $3n/4$ | $n$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $50 \times 7$ | 6,699 | 6,463 | 5,959 | 5,922 | 5,178 | 4,802 | 4,675 | 4,035 | 3,878 | 3,656 |
| $50 \times 9$ | 6,036 | 5,698 | 5,689 | 5,348 | 4,720 | 4,468 | 3,867 | 3,924 | 3,587 | 3,486 |
| $100 \times 7$ | 3,063 | 2,728 | 2,746 | 2,199 | 1,996 | 1,767 | 1,791 | 1,754 | 1,560 | 1,555 |
| $100 \times 9$ | 2,848 | 2,658 | 2,692 | 2,312 | 2,044 | 1,804 | 1,803 | 1,788 | 1,704 | 1,600 |
| **Média** | **4,662** | **4,387** | **4,272** | **3,945** | **3,484** | **3,210** | **3,034** | **2,875** | **2,682** | **2,574** |

**Figura C.1 – Gráfico de médias e intervalos de confiança de 95% de Fisher - Teste preliminar da perturbação**




Os resultados da ANOVA são resumidos através do gráfico de médias e intervalos de confiança de 95% de Fisher LSD, mostrados pela Figura C.1. Este gráfico estabelece que não existem diferenças significativas para valores de *d* entre 2 e 4. Desta forma, conclui-se que o tamanho da perturbação deve ser fixo (independente do tamanho do problema) e que $d \in \{2,3,4\}$. A seleção final do tamanho da perturbação é feita através da parametrização do ILS na seção 6.2.

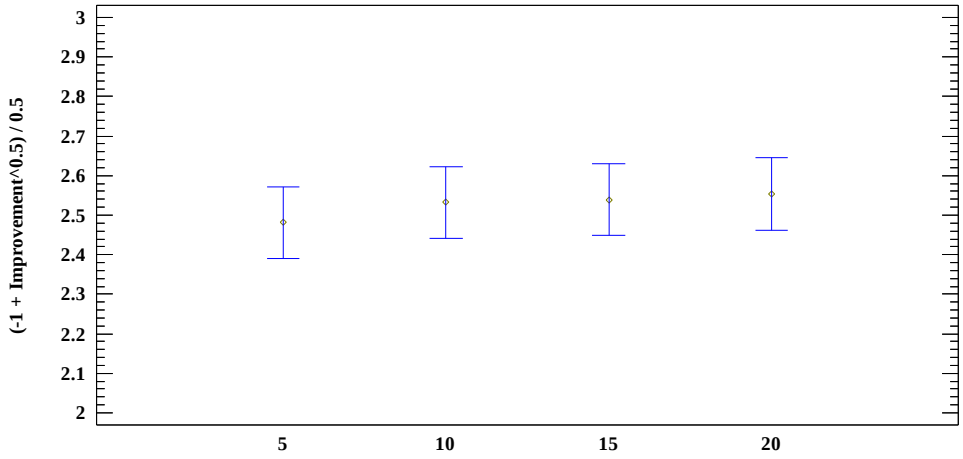
A segunda parte dos testes preliminares visa estabelecer o valor do parâmetro *NumPert*, requerido pelo operador *GL*. O ILS deste teste é igual ao ILS utilizado no teste anterior, exceto que a perturbação é realizada pelo operador *GL*. Seguindo os resultados anteriores, o tamanho da perturbação é fixado em 2. Note que *d* também poderia ter sido fixado como 3 ou 4, dado que não houve diferenças estatisticamente significativas entre eles. No entanto, quando $d = 2$ os resultados foram levemente melhores.

Cada instância foi resolvida cinco vezes para cada valor de *NumPert* e os resultados, para cada tamanho de instância, são mostrados na Tabela C.3. Como observado, as diferenças entre os diferentes valores de *NumPert* são mínimas, o que indica que os desempenho do ILS não melhora de forma significativa através da geração de mais soluções perturbadas.

**Tabela C.3 – Resultados dos testes preliminares do parâmetro *NumPert***

| $n \times m$ | 5 | 10 | 15 | 20 |
|---|---|---|---|---|
| $50 \times 7$ | 7,485 | 7,641 | 7,660 | 7,703 |
| $50 \times 9$ | 6,919 | 7,043 | 7,052 | 7,142 |
| $100 \times 7$ | 3,331 | 3,464 | 3,616 | 3,517 |
| $100 \times 9$ | 3,427 | 3,475 | 3,354 | 3,435 |
| **Média** | **5,290** | **5,406** | **5,420** | **5,449** |

**Figura C.2 – Gráfico de médias e intervalos de 95% de Fisher do parâmetro *NumPert***




A Figura C.2 resume os resultados de uma ANOVA realizada para o parâmetro *NumPert*. Como observado, todos os intervalos de confiança se sobrepõem, indicando que não existem diferenças significativas para os diversos valores do parâmetro *NumPert*. Dado este resultado, a escolha de qualquer valor é indiferente e optou-se por $NumPert = 20$.

Para finalizar apresentam-se os resultados dos testes preliminares do parâmetro $\lambda$, qual permite calcular o valor da temperatura, *T*, no critério de aceitação *LSMC*. Foram testados valores de $\lambda$ desde 0.1 até 1, com incrementos de 0,1. Os resultados são apresentados na Tabela C.4.

Novamente é possível observar que as diferenças entre os diversos valores testados não são significativas. Isto é melhor confirmado através de uma ANOVA, cujos resultados são mostrados pela Figura C.3. Deste modo, é possível concluir que o ILS é robusto em relação ao parâmetro $\lambda$ (RUIZ; STÜTZLE, 2007, 2008).

Em resumo, qualquer escolha do parâmetro $\lambda$ é indiferente devido a que a variação dos resultados atingidos será insignificante. Levando em conta o maior valor da média, escolheu-se um valor para $\lambda$ de 0,7.

**Tabela C.4 – Resultados dos testes preliminares do parâmetro lambda**

| $n \times m$ | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | 0,7 | 0,8 | 0,9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| $50 \times 7$ | 7,211 | 7,531 | 7,463 | 7,676 | 7,557 | 7,336 | 7,532 | 7,419 | 7,562 | 7,443 |
| $50 \times 9$ | 6,959 | 6,969 | 6,783 | 6,858 | 6,923 | 6,909 | 7,092 | 6,934 | 6,997 | 6,851 |
| $100 \times 7$ | 3,492 | 3,368 | 3,491 | 3,407 | 3,576 | 3,442 | 3,656 | 3,360 | 3,464 | 3,360 |
| $100 \times 9$ | 3,463 | 3,472 | 3,308 | 3,467 | 3,301 | 3,413 | 3,596 | 3,331 | 3,324 | 3,357 |
| **Média** | **5,282** | **5,335** | **5,261** | **5,352** | **5,339** | **5,275** | **5,469** | **5,261** | **5,337** | **5,253** |

**Figura C.3 – Gráfico de médias e intervalos de confiança de 95% de Fisher do parâmetro lambda**