EVERTON LUIZ DE MELO

**Meta-heurísticas *Iterated Local Search*, GRASP e *Artificial Bee Colony*
aplicadas ao *Job Shop* Flexível para minimização do atraso total**

São Paulo
2014

EVERTON LUIZ DE MELO

**Meta-heurísticas *Iterated Local Search*, GRASP e *Artificial Bee Colony*
aplicadas ao *Job Shop* Flexível para minimização do atraso total**

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção do
título de Doutor em Ciências.

São Paulo
2014

EVERTON LUIZ DE MELO

**Meta-heurísticas *Iterated Local Search*, GRASP e *Artificial Bee Colony*
aplicadas ao *Job Shop* Flexível para minimização do atraso total**

Tese apresentada à Escola Politécnica da
Universidade de São Paulo para obtenção do
título de Doutor em Ciências.

Área de Concentração:
Engenharia de Produção

Orientadora:
Prof^a. Livre-Docente Débora Pretti Ronconi

São Paulo
2014

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 20 de fevereiro de 2014.

Assinatura do autor _____

Assinatura do orientador _____

CATALOGAÇÃO-NA-PUBLICAÇÃO

Melo, Everton Luiz de

Meta-heurísticas *Iterated Local Search*, GRASP e *Artificial Bee Colony* aplicadas ao *Job Shop* Flexível para minimização do atraso total / E.L. de Melo. -- versão corr. -- São Paulo, 2014.
222 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Produção.

1.Otimização combinatória 2.Programação da produção
3.Modelos matemáticos 4.Heurística I.Universidade de São Paulo.
Escola Politécnica. Departamento de Engenharia de Produção II.t.

AGRADECIMENTOS

Os agradecimentos devidos são muitos, mas neste espaço são mencionadas as colaborações mais diretamente relacionadas à elaboração do presente trabalho.

À professora Débora Ronconi, pela orientação, dedicação e ensinamentos transmitidos no decorrer do doutorado.

Aos professores da Escola Politécnica e do Instituto de Matemática e Estatística com os quais cursei disciplinas: Débora Ronconi, Routh Terada, Marco Brinati, Afonso Fleury e Davi Nakano.

Aos colegas da pós-graduação pelo companheirismo: Celso Sakuraba, Irineu Brito, Guilherme Mainieri, Sergio Gomez, João Manguino e Ricardo Damm.

Aos professores que fizeram parte da banca do exame de qualificação, Cintia Scrich e José Arroyo, pelas contribuições, bem como aos professores que participaram da banca de pré-qualificação, Miguel Santoro e Hugo Yoshizaki.

Aos participantes do projeto GOLD: professora Cristina Fernandes, professor Paulo Feofiloff, professor Ernesto Birgin e Marcio Oshiro. Também à HP pelo apoio financeiro.

Aos colaboradores do Departamento de Engenharia de Produção, da Escola Politécnica e da Universidade de São Paulo.

À CAPES pela bolsa concedida, à USP pelo suporte, aos governos e às populações federal e estadual.

A meus pais, Eliza e Valdenir, e a Emília, pelo apoio nesse período.

A Deus por possibilitar que tudo isso acontecesse.

RESUMO

O ambiente de produção abordado neste trabalho é o *Job Shop* Flexível (JSF), uma generalização do *Job Shop* (JS). O problema de programação de tarefas, ou *jobs*, no ambiente JS é classificado por Garey; Johnson e Sethi (1976) como NP-Difícil e o JSF é, no mínimo, tão difícil quanto o JS. O JSF é composto por um conjunto de *jobs*, cada qual constituído por operações. Cada operação deve ser processada individualmente, sem interrupção, em uma única máquina de um subconjunto de máquinas habilitadas. O principal critério de desempenho considerado é a minimização dos atrasos dos *jobs*. São apresentados modelos de Programação Linear Inteira Mista (PLIM) para minimizar o atraso total e o instante de término da última operação, o *makespan*. São propostas novas regras de prioridade dos *jobs*, além de adaptações de regras da literatura. Tais regras são utilizadas por heurísticas construtivas e são aliadas a estratégias cujo objetivo é explorar características específicas do JSF. Visando aprimorar as soluções inicialmente obtidas, são propostas buscas locais e outros mecanismos de melhoria utilizados no desenvolvimento de três meta-heurísticas de diferentes categorias. Essas meta-heurísticas são: *Iterated Local Search* (ILS), classificada como meta-heurística de trajetória; *Greedy Randomized Adaptive Search* (GRASP), meta-heurística construtiva; e *Artificial Bee Colony* (ABC), meta-heurística populacional recentemente proposta. Esses métodos foram selecionados por alcançarem bons resultados para diversos problemas de otimização da literatura. São realizados experimentos computacionais com 600 instâncias do JSF, permitindo comparações entre os métodos de resolução. Os resultados mostram que explorar as características do problema permite que uma das regras de prioridade propostas supere a melhor regra da literatura em 81% das instâncias. As meta-heurísticas ILS, GRASP e ABC chegam a conseguir mais de 31% de melhoria sobre as soluções iniciais e a obter atrasos, em média, somente 2,24% superiores aos das soluções ótimas. Também são propostas modificações nas meta-heurísticas que permitem obter melhorias ainda mais expressivas sem aumento do tempo de execução. Adicionalmente é estudada uma versão do JSF com operações de Montagem e Desmontagem (JSFMD) e os experimentos realizados com um conjunto de 150 instâncias também indicam o bom desempenho dos métodos desenvolvidos.

PALAVRAS-CHAVE: Programação de tarefas. *Job shop* flexível. Atraso total. Programação Linear Inteira Mista. Heurística construtiva. Busca Local. Meta-heurística. *Iterated Local Search*. GRASP. *Artificial Bee Colony*.

ABSTRACT

The production environment addressed herein is the Flexible Job Shop (FJS), a generalization of the Job Shop (JS). In the JS environment, the jobs scheduling problem is classified by Garey; Johnson and Sethi (1976) as NP-Hard and the FJS is at least as difficult as the JS. FJS is composed of a set of jobs, each consisting of operations. Each operation must be processed individually, without interruption, in a single machine of a subset of enabled machines. The main performance criterion is minimizing the jobs tardiness. Mixed Integer Linear Programming (MILP) models are presented. These models minimize the total tardiness and the completion time of the last operation, makespan. New priority rules of jobs are proposed, as well as adaptations of rules from the literature. These rules are used by constructive heuristics and are combined with strategies aimed at exploiting specific characteristics of FSJ. In order to improve the solutions initially obtained, local searches and other improvement mechanisms are proposed and used in the development of metaheuristics of three different categories. These metaheuristics are: Iterated Local Search (ILS), classified as trajectory metaheuristic; Greedy Randomized Adaptive Search (GRASP), constructive metaheuristic, and Artificial Bee Colony (ABC), recently proposed population metaheuristic. These methods were selected owing to their good results for various optimization problems in the literature. Computational experiments using 600 FJS instances are carried out to allow comparisons between the resolution methods. The results show that exploiting the characteristics of the problem allows one of the proposed priority rules to exceed the best literature rule in about 81% of instances. Metaheuristics ILS, GRASP and ABC achieve more than 31% improvement over the initial solutions and obtain an average tardiness only 2.24% higher than the optimal solutions. Modifications in metaheuristics are proposed to obtain even more significant improvements without increased execution time. Additionally, a version called Disassembly and Assembly FSJ (DAFJS) is studied and the experiments performed with a set of 150 instances also indicate good performance of the methods developed.

KEYWORDS: Scheduling. Flexible Job shop. Total tardiness. Mixed Integer Linear Programming. Constructive heuristic. Local search. Metaheuristic. Iterated Local Search. GRASP. Artificial Bee Colony.

LISTA DE ILUSTRAÇÕES

Figura 1 - Grafo disjuntivo do JS.	26
Figura 2 - Grafo disjuntivo do JSF.	28
Figura 3 - Exemplo de JSF.	28
Figura 4 - Solução do JSF.	30
Figura 5 - Gráfico de Gantt correspondente à solução do JSF.	30
Figura 6 - Hierarquia de problemas de <i>scheduling</i> determinísticos.	32
Figura 7 - Diferentes configurações de <i>jobs</i>	42
Figura 8 - Sequência de uma regra míope.	54
Figura 9 - Possível sequência de uma regra com antevisão.	54
Figura 10 - Convergência da BL para um ótimo local.	59
Figura 11 - Caminho crítico do <i>job</i> 1 a sofrer inversão de arco.	61
Figura 12 - Solução obtida após a inversão de arco.	61
Figura 13 - Gráfico de Gantt da solução obtida com a inversão de arco.	62
Figura 14 - Caminho crítico do <i>job</i> 2 a ser alterado pelo movimento de inserção.	67
Figura 15 - Solução obtida com a mudança de posição de uma operação por inserção.	68
Figura 16 - Solução obtida com a troca de máquina de uma operação.	68
Figura 17 - Gráfico de Gantt da solução obtida com as inserções.	69
Figura 18 - Fluxograma da meta-heurística ILS.	76
Figura 19 - Como a ILS evita a estagnação em ótimos locais.	77
Figura 20 - Exploração da região factível pela ILS.	78
Figura 21 - Fluxograma da meta-heurística GRASP.	84
Figura 22 - Exploração da região factível pela GRASP.	85
Figura 23 - Fluxograma da meta-heurística ABC.	89
Figura 24 - Exploração da região factível pela ABC.	90
Figura 25 - Multiplicadores usados para as datas de entrega.	100
Figura 26 - Gráfico de Gantt da solução ótima obtida para a instância MFJS4.	103
Figura 27 - Calibragem da regra DMA.	107
Figura 28 - Perfis de desempenho das melhores regras de prioridade.	116
Figura 29 - Atraso total em função do número de iterações da ILS.	122
Figura 30 - Exemplo de matriz de frequência.	124
Figura 31 - Melhoria percentual entre CPLEX e ILS por instância.	132

Figura 32 - Atraso total em função do número de iterações da GRASP.....	140
Figura 33 - Melhoria percentual entre CPLEX e GRASP por instância.	146
Figura 34 - Atraso total em função do número de iterações da ABC.....	154
Figura 35 - Melhoria percentual entre CPLEX e ABC por instância.	162
Figura 36 - Gráfico <i>boxplot</i> com as meta-heurísticas ILS, GRASP e ABC.....	168
Figura 37 - Perfis de desempenho de ILS, GRASP e ABC.....	169
Figura 38 - Diferentes porcentagens de tempo dedicado à ênfase construtiva.....	173
Figura 39 - Perfis de desempenho de ILS-EC, GRASP-EC e ABC-EC.	180
Figura 40 - Tipos dos <i>jobs</i> das instâncias JSFMD.	181
Figura 41 - <i>Jobs</i> e gráfico de Gantt de uma instância JSFMD.....	184
Figura 42 - Atraso total em função das iterações da ABC em instâncias JSFMD.	190
Figura 43 - Perfis de desempenho das meta-heurísticas com instâncias JSFMD.....	196
Figura 44 - Número de soluções de TSP, JS e JSF em função de n	211
Figura 45 - Número de soluções de TSP, JS e JSF em escala logarítmica.....	212

LISTA DE TABELAS

Tabela 1 - Valores dos componentes das instâncias JSF geradas.....	98
Tabela 2 - Resultados da minimização do <i>makespan</i>	102
Tabela 3 - Inícios e términos das operações da instância MFJS4.....	104
Tabela 4 - Resultados obtidos com o modelo que minimiza o atraso total.	105
Tabela 5 - Resultados das regras de prioridade básicas por dimensão.	108
Tabela 6 - Resultados das regras básicas por tipo de data de entrega.....	109
Tabela 7 - Resultados das regras básicas por nível de flexibilidade.....	109
Tabela 8 - Resultados das regras com estratégias por dimensão.	111
Tabela 9 - Resultados das regras com estratégias por tipo de data de entrega.	112
Tabela 10 - Resultados das regras com estratégias por nível de flexibilidade.	113
Tabela 11 - Melhorias obtidas ao explorar afinidade e flexibilidade.	114
Tabela 12 - Melhorias obtidas pelas estratégias por nível de flexibilidade.....	114
Tabela 13 - RDIs médios das principais regras implementadas.....	115
Tabela 14 - Resultados do CPLEX e das heurísticas construtivas.	117
Tabela 15 - Resultados das buscas locais propostas.....	119
Tabela 16 - Melhorias da ILS em relação à EDD-AAF.	127
Tabela 17 - Melhorias da ILS em relação à BL.....	129
Tabela 18 - Resultados do método exato e da ILS.	130
Tabela 19 - Resultados do CPLEX e da ILS por nível de flexibilidade.....	131
Tabela 20 - Melhorias da GRASP em relação à EDD-AAF.	143
Tabela 21 - Resultados do método exato e da GRASP.	144
Tabela 22 - Resultados do CPLEX e da GRASP por nível de flexibilidade.....	145
Tabela 23 - Melhorias da ABC em relação à EDD-AAF.....	158
Tabela 24 - Resultados do método exato e da ABC.....	160
Tabela 25 - Resultados do CPLEX e da ABC por nível de flexibilidade.....	161
Tabela 26 - Resultados das meta-heurísticas executadas com o mesmo tempo.....	163
Tabela 27 - Atrasos médios das meta-heurísticas por nível de flexibilidade.	165
Tabela 28 - Resultados das meta-heurísticas por tipo de data de entrega.	167
Tabela 29 - RDIs médios das meta-heurísticas.	167
Tabela 30 - Melhorias obtidas com a ênfase construtiva por nível de flexibilidade.	175
Tabela 31 - Resultados da ênfase construtiva em relação ao CPLEX.....	177

Tabela 32 - Resultados das meta-heurísticas com ênfase construtiva.	178
Tabela 33 - Valores dos componentes das instâncias JSFMD geradas.	183
Tabela 34 - Resultados do CPLEX para instâncias JSFMD.	185
Tabela 35 - Atrasos médios e melhorias das regras com sincronização de ramos.....	188
Tabela 36 - Resultados das meta-heurísticas para o JSFMD.	191
Tabela 37 - Melhorias das meta-heurísticas em relação à DMA-AF-MD.....	193
Tabela 38 - Resultados das meta-heurísticas em relação ao CPLEX com o JSFMD.	194
Tabela 39 - RDIs médios das meta-heurísticas para o JSFMD.	195
Tabela 40 - Meta-heurísticas com ênfase construtiva aplicadas ao JSFMD.....	197
Tabela 41 - Resultados de diferentes versões da ILS.	213
Tabela 42 - Resultados de diferentes versões da GRASP.....	215
Tabela 43 - Resultados de diferentes versões da ABC.	217
Tabela 44 - Atrasos médios da população inicial da ABC e da melhor solução inicial.	218

LISTA DE ABREVIATURAS E SIGLAS

ABC	<i>Artificial Bee Colony</i>
ABC-EC	ABC com Ênfase Construtiva
AG	Algoritmo Genético
BL	Busca Local
BL-ARC	BL baseada em inversão de Arcos
BL-ARC-AI	BL-ARC com Atualizações Internas
BL-INS	BL baseada em Inserção
BL-INS-AI	BL-INS com Atualizações Internas
BT	Busca Tabu
DMA	Data de Entrega Modificada com Antevisão
DMA-AF	DMA com Afinidade e Flexibilidade
DMA-AF-MD	DMA-AF com Montagem e Desmontagem
EDD	<i>Earliest Due Date</i>
EDD-AAF	EDD com Antevisão, Afinidade e Flexibilidade
EDD-AAF-MD	EDD-AAF com Montagem e Desmontagem
GRASP	<i>Greedy Randomized Adaptive Search Procedure</i>
GRASP-EC	GRASP com Ênfase Construtiva
ILS	<i>Iterated Local Search</i>
ILS-EC	ILS com Ênfase Construtiva
ILS-PV	ILS com Perturbação Variável
JS	<i>Job Shop</i>
JSF	<i>Job Shop</i> Flexível
JSFMD	JSF com Montagem e Desmontagem
LRC	Lista Restrita de Candidatas
MABC	<i>Modified Artificial Bee Colony</i>
MDD	<i>Modified Due Date</i>
MDD-AAF	MDD com Antevisão, Afinidade e Flexibilidade
MDD-AAF-MD	MDD-AAF com Montagem e Desmontagem
MOD	<i>Modified Operation Due date</i>
MST	<i>Minimum Slack Time</i>
PLIM	Programação Linear Inteira Mista

PRTTa	<i>Priority Rule for Total Tardiness adaptada</i>
RDI	<i>Relative Deviation Index</i>
SA	<i>Simulated Annealing</i>
SCR	<i>Smallest Critical Ratio</i>
SPT	<i>Shortest Processing Time</i>
TSP	<i>Travelling Salesman Problem</i>

LISTA DE SÍMBOLOS

n	número de <i>jobs</i>
m	número de máquinas
N	número de operações
J	conjunto de <i>jobs</i>
M	conjunto de máquinas
O	conjunto de operações
i	índice de <i>jobs</i>
k	índice de máquinas
j	índice de operações
O_i	conjunto de operações do <i>job i</i>
M_j	conjunto de máquinas alternativas para processar a operação <i>j</i>
t_{ijk}	tempo de processamento da operação O_{ij} na máquina <i>k</i>
L	número grande
$C_{\max}$	máximo instante de término de processamento dentre os <i>jobs</i>
d_i	data de entrega do <i>job i</i>
t	instante atual sendo considerado pelo <i>list scheduling algorithm</i>
G_i	conjunto das operações ainda não processadas do <i>job i</i>
t_{jk}	tempo de processamento da operação <i>j</i> na máquina <i>k</i>
d_i'	data de entrega modificada do <i>job i</i>
α	constante que pondera o primeiro termo das regras com antevisão
A_{jk}	afinidade entre a operação <i>j</i> a máquina <i>k</i>
F_j	flexibilidade da operação <i>j</i>
r_j	instante de liberação da operação <i>j</i>
s	solução factível de um problema de otimização
s^0	solução inicial
s^*	solução incumbente
S	conjunto de soluções factíveis
$N(s)$	vizinhança da solução <i>s</i>
$f(s)$	valor da função-objetivo para a solução <i>s</i>
δ	fator usado no cálculo das datas de entrega dos <i>jobs</i>
$\%Dif$	diferença relativa entre dois resultados médios

τ	fator de multiplicação da melhor solução nos perfis de desempenho
θ	valor que limita operações para LRC com base na diferença máxima
φ	valor que limita operações para LRC com base na candidata prioritária
Θ	conjunto de valores para θ
β	constante que pondera a penalização de ramos adiantados
R_i	número de estágios do <i>job i</i>

SUMÁRIO

1	INTRODUÇÃO	21
1.1	OBJETIVOS.....	21
1.2	JUSTIFICATIVAS	22
1.3	METODOLOGIA	22
1.4	ORGANIZAÇÃO DO TRABALHO	22
2	DESCRIÇÃO DO PROBLEMA	25
2.1	O PROBLEMA DE <i>JOB SHOP</i> CLÁSSICO	25
2.2	<i>JOB SHOP</i> FLEXÍVEL	27
2.3	REVISÃO BIBLIOGRÁFICA.....	33
3	FORMULAÇÕES MATEMÁTICAS.....	39
3.1	MODELOS PARA A MINIMIZAÇÃO DO <i>MAKESPAN</i>	39
3.2	MODELO ADAPTADO PARA A MINIMIZAÇÃO DO ATRASO TOTAL.....	44
4	HEURÍSTICAS CONSTRUTIVAS.....	47
4.1	REGRAS DE PRIORIDADE ADAPTADAS DA LITERATURA	48
4.2	IDENTIFICANDO AS CARACTERÍSTICAS DO PROBLEMA	52
4.2.1	Afinidade entre operações e máquinas	52
4.2.2	Grau de flexibilidade das operações	53
4.3	REGRAS DE PRIORIDADE PROPOSTAS	53
4.3.1	Data de Entrega Modificada com Antevisão.....	53
4.3.2	Data de Entrega Modificada com Antevisão, Afinidade e Flexibilidade.....	55
4.4	ESTRUTURA GERAL DAS HEURÍSTICAS CONSTRUTIVAS	55
5	BUSCAS LOCAIS	59
5.1	BUSCA LOCAL COM INVERSÃO DE ARCOS	60
5.2	BUSCA LOCAL COM INVERSÃO DE ARCOS E ATUALIZAÇÃO INTERNA	65
5.3	BUSCA LOCAL COM INSERÇÃO.....	66
5.4	BUSCA LOCAL COM INSERÇÃO E ATUALIZAÇÃO INTERNA.....	70
6	META-HEURÍSTICAS	73
6.1	<i>ITERATED LOCAL SEARCH</i>	74
6.1.1	Procedimento de perturbação	78
6.1.2	<i>Iterated Local Search</i> proposta	81
6.2	GRASP	82

6.2.1	Geração de soluções com aleatoriedade	85
6.2.2	GRASP proposta	86
6.3	<i>ARTIFICIAL BEE COLONY</i>	87
6.3.1	Componentes da meta-heurística	91
6.3.2	Artificial Bee Colony proposta.....	96
7	EXPERIMENTOS COMPUTACIONAIS COM O JSF	97
7.1	GERAÇÃO DE INSTÂNCIAS	97
7.2	FORMULAÇÕES MATEMÁTICAS.....	100
7.2.1	Minimização do <i>makespan</i>	101
7.2.2	Minimização do atraso total.....	104
7.3	HEURÍSTICAS CONSTRUTIVAS.....	106
7.4	BUSCAS LOCAIS	118
7.5	<i>ITERATED LOCAL SEARCH</i>	120
7.5.1	Seleção e ajuste de componentes.....	120
7.5.2	Resultados da ILS proposta	127
7.6	GRASP.....	133
7.6.1	Seleção e ajuste de componentes.....	133
7.6.2	Resultados da GRASP proposta	143
7.7	<i>ARTIFICIAL BEE COLONY</i>	146
7.7.1	Seleção e ajuste de componentes.....	146
7.7.2	Resultados da ABC proposta	157
7.8	COMPARAÇÕES ENTRE META-HEURÍSTICAS.....	162
7.9	META-HEURÍSTICAS COM ÊNFASE NA FASE CONSTRUTIVA	171
7.9.1	Implementações e ajustes da ênfase construtiva	172
7.9.2	Resultados das meta-heurísticas com ênfase construtiva.....	175
8	EXPERIMENTOS COMPUTACIONAIS COM O JSFMD	181
8.1	GERAÇÃO DE INSTÂNCIAS JSFMD.....	181
8.2	FORMULAÇÃO MATEMÁTICA.....	185
8.3	HEURÍSTICAS CONSTRUTIVAS PARA O JSFMD	186
8.4	META-HEURÍSTICAS APLICADAS AO JSFMD.....	189
9	CONCLUSÕES	199
	REFERÊNCIAS	203
	APÊNDICE A – Número de soluções do JSF em função de n e m	211
	APÊNDICE B – Resultados adicionais da ILS.....	213

APÊNDICE C – Resultados adicionais da GRASP	215
APÊNDICE D – Resultados adicionais da ABC	217
APÊNDICE E – Exemplo de instância do JSF e grafos do JSFMD	219

1 INTRODUÇÃO

O foco deste trabalho é a investigação e o desenvolvimento de modelos matemáticos, métodos heurísticos e meta-heurísticas para a resolução do problema de programação de tarefas *Job Shop* Flexível (JSF). Esse problema é uma generalização do clássico problema de *scheduling Job Shop* (JS), relacionado a ambientes produtivos com grande variedade de produtos e baixo volume de produção. O JS consiste de um conjunto de *jobs*, cada um formado por uma sequência de operações. Cada operação deve ser processada em uma máquina previamente determinada e o processo de determinação das sequências das operações nas máquinas deve otimizar algum critério pré-estabelecido. Com o critério mais explorado, a minimização do *makespan*, o problema é classificado por Garey; Johnson e Sethi (1976) como NP-Difícil. O JSF difere do JS em razão de cada operação possuir um subconjunto de máquinas capazes de efetuar seu processamento. Isso exige definir também em qual máquina cada operação deve ser processada.

O JSF pode ter diferentes critérios de desempenho e neste estudo o critério de desempenho utilizado é a minimização do atraso total dos *jobs*. O problema, assim como o JS, é classificado como NP-Difícil. Isso significa que não são conhecidos algoritmos exatos para sua resolução em tempo computacional aceitável, de acordo com Cormen *et al.* (2002). Por tal razão, métodos alternativos, como heurísticas e meta-heurísticas, são utilizados para que boas soluções sejam obtidas em tempo razoável.

1.1 OBJETIVOS

O objetivo principal deste trabalho é identificar e desenvolver métodos de resolução eficientes para o JSF utilizando o critério de minimização do atraso total. Entre os métodos desenvolvidos, este trabalho propõe e implementa modelos de Programação Linear Inteira Mista (PLIM) para o JSF para que soluções ótimas e limitantes superiores sejam utilizados na avaliação de procedimentos não exatos. O trabalho também envolve o desenvolvimento de métodos heurísticos construtivos, baseados em regras de prioridade propostas, para a geração rápida de soluções factíveis de qualidade aceitável. Essas heurísticas construtivas, além de serem eficientes, exploram características do ambiente JSF. Ao mesmo tempo, são propostas Buscas Locais (BL) baseadas em movimentos que geram vizinhanças mais amplas que as BLs mais utilizadas na literatura. Essas BLs propostas são utilizadas por métodos de melhoria de diferentes categorias. São desenvolvidas as meta-heurísticas: *Iterated Local Search* (ILS), a

qual é uma meta-heurística de trajetória; *Greedy Randomized Adaptive Search Procedure* (GRASP), uma meta-heurística construtiva; e *Artificial Bee Colony* (ABC), meta-heurística populacional. Cada uma dessas meta-heurísticas utiliza uma estratégia para buscar melhorias a partir das soluções iniciais fornecidas pelas heurísticas construtivas. Assim, além de desenvolver métodos eficientes para a resolução do JSF, esta pesquisa também investiga a existência de diferença de desempenho entre as meta-heurísticas de categorias distintas na resolução do mesmo problema. Adicionalmente, este trabalho estuda uma versão do problema que envolve operações de montagem e de desmontagem, sendo denominada JSF com Montagem e Desmontagem (JSFMD).

1.2 JUSTIFICATIVAS

A pesquisa se justifica pela dificuldade encontrada na resolução do JSF e pela aplicabilidade prática do problema. A aplicabilidade do JSF pode ser observada nos trabalhos de Alvarez-Valdes *et al.* (2005) envolvendo a indústria de vidro e de Vilcot e Billaut (2008) na indústria de impressão. Além disso, a pesquisa se justifica por existir uma lacuna na literatura sobre o estudo do problema. A lacuna ocorre especialmente em relação ao critério de desempenho atraso total, visto que na literatura o critério mais explorado é o *makespan*, conforme é abordado na seção sobre a revisão bibliográfica.

1.3 METODOLOGIA

Com relação à classificação metodológica, esta pesquisa utiliza a modelagem quantitativa, uma vez que o problema é abordado de modo matemático e não envolve subjetividade. A investigação engloba o desenvolvimento de modelos e de técnicas de resolução. Por isso, de acordo com Bertrand e Fransoo (2002), a linha a ser seguida é da pesquisa axiomática, com bifurcações nos campos normativo e descritivo.

1.4 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado em capítulos conforme segue:

Capítulo 2: o problema abordado é descrito. Primeiramente é feita uma descrição do problema clássico de JS e, então, o JSF é definido. Nesse capítulo também é apresentada uma revisão bibliográfica a respeito do JSF.

Capítulo 3: são apresentados modelos matemáticos para o JSF. Primeiramente é apresentado um modelo da literatura para a minimização do *makespan*. Depois são propostas

adaptações ao modelo da literatura que permitem resolver o JSFMD e utilizar o critério de minimização do atraso total.

Capítulo 4: as heurísticas construtivas são apresentadas. Primeiro são apresentadas regras de prioridade adaptadas da literatura. Depois são propostas novas regras, incluindo algumas que exploram características do JSF. Por fim é apresentada a estrutura geral da heurística construtiva baseada nas diferentes regras de prioridade.

Capítulo 5: as BLs utilizadas são descritas. São descritas duas BLs baseadas no movimento de inversão de arcos e duas BLs baseadas no movimento de inserção de operações. Duas dessas BLs, uma baseada em inversão e outra baseada em inserção, utilizam o conceito de subvizinhança.

Capítulo 6: são apresentadas as meta-heurísticas desenvolvidas ILS, GRASP e ABC. Juntamente com uma visão geral de cada uma das meta-heurísticas, são fornecidos detalhes a respeito dos elementos que as compõem.

Capítulo 7: os resultados obtidos com o JSF são analisados. Os resultados obtidos com instâncias JSF com os diversos métodos de resolução são apresentados e discutidos. São fornecidos detalhes a respeito da calibragem dos métodos, bem como a respeito de variações investigadas.

Capítulo 8: são apresentados os resultados alcançados com o JSFMD. Os resultados alcançados com os experimentos envolvendo as instâncias JSFMD são apresentados e analisados. Este capítulo também traz detalhes a respeito de uma proposta para as heurísticas construtivas melhorarem a sincronização de diferentes ramos de um mesmo *job*.

Capítulo 9: são apresentadas as conclusões da pesquisa.

2 DESCRIÇÃO DO PROBLEMA

Neste capítulo o problema em estudo é apresentado. Inicialmente é descrito o problema de programação de tarefas *Job Shop* (JS) clássico e, depois, é descrita sua versão flexibilizada. São feitas considerações a respeito da dificuldade de resolução do *Job Shop* Flexível (JSF), especialmente quando ele é comparado ao JS tradicional e são mencionados alguns dos critérios de otimização do problema flexibilizado. O capítulo também traz uma revisão bibliográfica sobre o JSF.

2.1 O PROBLEMA DE *JOB SHOP* CLÁSSICO

O ambiente de programação de tarefas JS tem como características a produção de alta variedade de produtos e baixo volume de produção. O problema pode ser enunciado conforme segue. Considere um conjunto independente de n peças, tarefas ou trabalhos, designados na língua inglesa por *jobs*, cada qual constituído por uma sequência ordenada específica de operações. Cada operação deve ser processada individualmente em uma única máquina, pertencente a um conjunto M formado por m máquinas, sem interrupção. Em um mesmo *job* cada operação deve ser executada em uma máquina diferente. Cada operação consome um diferente tempo de processamento, sendo que esse tempo pode ser adimensional e se referir genericamente a unidades de tempo. No caso do critério mais utilizado na literatura, a minimização do *makespan*, deve ser obtida uma sequência de processamentos de modo que o instante de término da última operação processada do último *job* finalizado seja o menor possível. Considerando tal critério de otimização o JS é classificado por Garey; Johnson e Sethi (1976) como NP-Difícil.

Além do *makespan*, outras medidas de desempenho também podem ser consideradas, entre elas: o atraso total, resultante da soma dos atrasos de todos os *jobs*; o atraso ponderado, semelhante ao atraso total mas com ponderações por *job*; o *lateness*, que é o distanciamento positivo ou negativo da data de entrega; a carga de trabalho total das máquinas; o balanceamento da carga de trabalho das máquinas; ou mesmo custos. Além da busca de satisfação isolada de um critério o JS ainda pode ter como objetivo otimizar vários critérios ao mesmo tempo, muitas vezes através de ponderações entre eles. Entre os critérios, segundo Pinedo (2012), são considerados regulares aqueles cujos valores da função-objetivo não diminuem à medida que os instantes de término dos *jobs* aumentam, como *makespan* e atraso. Outra característica do atraso é que, conforme Baker (1984), o mesmo não é uma função

linear do *makespan* e nem dos instantes de término dos *jobs*.

Vaessens; Aarts e Lenstra (1996) listam alguns métodos de tempo polinomial para o JS com minimização do *makespan*, mas os mesmos se restringem a problemas bastante limitados e específicos. Tais problemas devem ter como características: envolver somente 2 máquinas e, no máximo, 2 operações por *job*; envolver somente 2 máquinas e ter todos os tempos de processamento unitários; envolver somente 2 *jobs*; ou possuir um grafo disjuntivo cujo caminho crítico tenha comprimento máximo igual a 3.

Du e Leung (1990) demonstram que o problema de minimizar o atraso total em uma única máquina é NP-Difícil. Koulamas (1994) prova que minimizar o atraso no ambiente *flowshop* com duas máquinas é fortemente NP-Difícil e isso implica que minimizar o atraso no JS também é fortemente NP-Difícil. Lawer *et al.* (1993) asseguram que o JS é um dos problemas combinatoriais mais difíceis. Para exemplificar com clareza a dificuldade de resolução do JS, Bagchi (1999) afirma que se um computador moderno estivesse resolvendo um JS por enumeração explícita desde o início estimado do universo, cerca de 20 bilhões de anos atrás, teriam sido escalonados apenas 24 *jobs* em 1 máquina.

Um modo de representar o JS é através de um grafo, proposto por Balas (1969), denominado grafo disjuntivo. Nesse grafo cada operação de cada tarefa é representada por um vértice cujo peso é o tempo de processamento da respectiva operação. Além desses, há mais dois vértices artificiais que representam o início e o término dos *jobs*. Arcos ligam o vértice artificial de início à primeira operação de cada *job*, bem como ligam a operação final de cada *job* ao vértice artificial de término. Também existem arcos conjuntivos que ligam as operações consecutivas de cada *job*, determinando a ordem em que elas devem ser processadas. Finalmente são acrescentados arcos com orientação ainda não definida, chamados arcos disjuntivos, que conectam operações a serem executadas numa mesma máquina. A Figura 1 ilustra o grafo disjuntivo de um JS com 3 *jobs* e 4 máquinas.

Figura 1 - Grafo disjuntivo do JS.

Fonte: elaborada pelo autor.

No grafo disjuntivo da Figura 1, os vértices 0 e 11, tracejados, representam, respectivamente, as operações artificiais de início, ou origem, e de fim, ou destino. O primeiro *job* é composto pelas operações 1, 2, 3 e 4, nessa ordem; o segundo *job* é formado pelas operações 5, 6 e 7; e o terceiro, pelas operações 8, 9 e 10. Arcos disjuntivos unem as operações 2, 5 e 8, indicando que elas devem ser processadas na mesma máquina, M1. As operações 1, 7 e 9 possuem arcos disjuntivos entre si, pois todas demandam processamento em M2. A operação 3 é a única a ser executada em M3, máquina que não possui arcos disjuntivos. Já as operações 4, 6 e 10 precisam ser executadas na máquina M4. Desse modo o grafo disjuntivo possui m cliques, ou seja, um subgrafo completo por máquina.

A resolução do problema corresponde à definição das direções dos arcos disjuntivos de modo que sejam determinadas as ordens das operações a serem processadas em cada uma das máquinas. Assim se obtém um grafo orientado que, se for acíclico, representa uma solução factível do JS. A partir desse grafo orientado é identificado o maior caminho do vértice inicial ao final, denominado caminho crítico, o qual determina o valor do *makespan*.

2.2 JOB SHOP FLEXÍVEL

O JSF é uma extensão do JS na qual uma mesma operação pode ser executada alternativamente por mais de uma máquina. As máquinas capazes de processar cada operação podem ser idênticas ou não, conforme Brandimarte (1993), e cada operação pode ser atribuída somente a uma única máquina. Essa flexibilidade de escolha torna possível um *job* ser processado através de diversos caminhos alternativos através das máquinas, o que aumenta significativamente o número de soluções factíveis. Por essa razão, na resolução do JSF são empregadas diferentes abordagens e vários métodos de resolução, principalmente heurísticos.

Assim como o JS, o JSF pode ser representado por um grafo disjuntivo, conforme Dauzère-Pérès e Paulli (1997). A Figura 2 mostra como fica um JSF derivado do JS da Figura 1 pela flexibilização da operação 1, que passa a poder ser processada também pela máquina M1, e pela flexibilização das operações 6 e 9 que passam a poder ser executadas também pela máquina M3.

Figura 2 - Grafo disjuntivo do JSF.

Fonte: elaborada pelo autor.

O grafo disjuntivo da Figura 2 possui mais arcos disjuntivos que o grafo da Figura 1. Alguns vértices passam a ter, ao mesmo tempo, a incidência de arcos disjuntivos de diferentes máquinas. A Figura 3 ilustra como seria o ambiente produtivo representado pelo grafo da Figura 2. As setas, equivalentes aos arcos disjuntivos, indicam os caminhos possíveis dos *jobs* através das máquinas, ilustrando as alternativas de processamento das operações.

Figura 3 - Exemplo de JSF.

Fonte: elaborada pelo autor.

Na representação da Figura 3, a atribuição de cada operação a uma máquina alternativa implica em encontrar, para cada *job*, um caminho que inicie à esquerda da figura e termine à sua direita. Esses caminhos devem seguir as setas, passando por cada uma delas, no máximo, uma vez. Para se obter uma solução do JSF, além de definir as atribuições, ainda é necessário determinar as sequências das operações a serem processadas em cada máquina. No ambiente representado, o *job* 1 tem 4 operações e sua primeira operação pode ser processada

alternativamente nas máquinas 1 ou 2, conforme indicam as setas na cor azul. A segunda operação do *job* 1 deve ser processada na máquina 1. Dessa forma, caso a primeira operação seja processada também na máquina 1, ocorre recirculação. A terceira operação do *job* 1 deve ser executada na máquina 3 e a última operação, na máquina 4. O *job* 2 utiliza setas na cor vermelha e possui 3 operações, sendo que somente a segunda delas tem alternativas de processamento, as máquinas 3 ou 4. O *job* 3, com setas na cor verde, também possui 3 operações e sua segunda operação pode ser processada tanto na máquina 2 quanto na máquina 3. Com essa ilustração fica claro como os *jobs* do JSF podem percorrer caminhos alternativos através das máquinas.

De acordo com Dauzère-Pérès e Paulli (1997), obter uma solução viável a partir de um grafo disjuntivo do JSF significa encontrar a seleção $E = \bigcup_{k \in M} E_k$, onde E_k é a seleção dos arcos disjuntivos da máquina k , do seguinte modo:

Passo 1) Para cada operação, defina a direção de um ou dois arcos disjuntivos incidentes nela e elimine os demais arcos disjuntivos. A seleção E deve obedecer:

- i) Haver ao menos um arco conjuntivo que chegue a cada operação e ao menos um arco conjuntivo que parta de cada operação;
- ii) Se uma operação tem um arco disjuntivo que chega a ela e um que parte dela, ambos os arcos estão em E_k ;
- iii) Em todo $E_k \neq \emptyset$ deve haver uma única operação sem um arco disjuntivo que chegue a ela e uma única operação sem arco disjuntivo que parta dela.

Passo 2) Para cada $E_k \neq \emptyset$, inclua em E_k o arco que liga o vértice artificial de origem à primeira operação de E_k e o arco que liga a última operação de E_k ao vértice artificial de término. Para cada $E_k = \emptyset$ inclua em E_k um arco que ligue os vértices artificiais de origem e de destino.

O segundo passo desse método inclui arcos que ligam a operação artificial de início à primeira operação de cada máquina e a última operação de cada máquina ao vértice artificial final. Isso serve para manter a consistência com as notações utilizadas no trabalho de Dauzère-Pérès e Paulli (1997) nas quais cada operação deve ter um arco que chegue nela e um arco que parta dela. Neste trabalho esses arcos são mantidos para seguir a representação de Dauzère-Pérès e Paulli (1997). Contudo os mesmos poderiam ser suprimidos sem qualquer prejuízo, assim como o Passo 2 do algoritmo anterior. Restaria então um grafo disjuntivo que representaria uma solução de forma equivalente à proposta por Balas (1969). Dauzère-Pérès e Paulli (1997) ainda afirmam que o JSF engloba os ambientes *Flexible Manufacturing Systems*

e de máquinas paralelas idênticas ou não.

Assim como na representação clássica do JS, o grafo do JSF define uma solução viável se for acíclico. A Figura 4 exemplifica uma solução obtida pela definição das orientações dos arcos disjuntivos do grafo da Figura 2.

Figura 4 - Solução do JSF.

Fonte: elaborada pelo autor.

Nessa solução a máquina M1 executa as operações 1, 2, 5 e 8; M2 executa as operações 9 e 7; M3 processa as operações 6, e 3; e M4 recebe as operações 4 e 10. Nessa representação, arcos disjuntivos redundantes como o arco entre os vértices 1 e 5, por exemplo, são omitidos. A solução da Figura 4 é apresentada utilizando um gráfico de Gantt na Figura 5, no qual cada *job* é representado por uma cor. Cada operação é identificada por dois valores. O primeiro valor indica o *job* ao qual ela pertence e o segundo, entre parênteses, indica sua numeração na contagem geral das operações que identifica os vértices do grafo da Figura 4.

Figura 5 - Gráfico de Gantt correspondente à solução do JSF.

Fonte: elaborada pelo autor.

Através do gráfico de Gantt da Figura 5 se pode observar, em cada máquina, as mesmas sequências determinadas pelos arcos disjuntivos das respectivas máquinas no grafo da Figura 4. Nesse exemplo as datas de entrega d_i de cada *job* i são indicadas no eixo

horizontal. Assim o *job* 1 tem atraso igual a 8, pois sua data de entrega é igual a 13 e sua conclusão ocorre no instante 21. Os *jobs* 2 e 3 têm, respectivamente, atrasos 7 e 16 e, com isso, o atraso total é igual a 31.

Considerando a classificação de problemas de *scheduling* utilizando os campos $\alpha|\beta|\gamma$ propostos em Graham *et al.* (1979) e as nomenclaturas apresentadas em Pinedo (2012), o JSF abordado neste trabalho é representado por $FJ | rcrc | \Sigma T_i$. Em Pinedo (2012) a notação FJc é utilizada no campo α para representar um JSF com c centros de trabalho com máquinas idênticas paralelas. Como o JSF em estudo não utiliza centros de trabalho, o campo α é representado somente por FJ . No campo β é indicada a possibilidade de recirculação dos *jobs* pelas máquinas. Já no campo γ é indicado o critério de desempenho utilizado, a minimização do atraso total.

As abordagens de resolução do JSF se distinguem pela decomposição ou não do problema em subproblemas menos complexos que podem ser atacados isoladamente. O JSF pode ser visto como um problema constituído por duas fases. A primeira delas é a atribuição de cada uma das operações a uma das máquinas habilitadas a realizar seu processamento. Essa etapa, por vezes, é denominada roteamento, já que as atribuições especificam os caminhos que os *jobs* percorrem através das máquinas. Definidas tais atribuições, vem a segunda fase. Ela é constituída pelo sequenciamento das operações nas máquinas, ou seja, pela resolução do JS resultante após as atribuições da primeira fase. Esse tipo de resolução é denominada abordagem hierárquica. Em contraste com ela existe a abordagem integrada, pela qual um determinado método de resolução busca obter soluções para o JSF tratando, ao mesmo tempo, atribuições e sequenciamentos. Assim como no JS, o JSF pode trabalhar com diversos critérios de avaliação.

Com relação à complexidade, a posição do JSF frente a outros problemas de *scheduling* pode ser avaliada com o grafo da Figura 6. Esse grafo fornece uma visão geral da complexidade de diferentes problemas determinísticos, cada um representado por um vértice.

Figura 6 - Hierarquia de problemas de *scheduling* determinísticos.

Fonte: adaptada de Pinedo (2012).

A partir do problema menos complexo, com máquina única, se chega a problemas de maior complexidade pela inclusão de novas características. Desse modo um determinado problema é redutível a qualquer problema a partir do qual exista um caminho. Nesse grafo o JSF se encontra entre os problemas mais complexos. Neste contexto a definição do JSF considera que as máquinas alternativas de uma operação podem ter tempos de processamento diferentes.

A dificuldade de se encontrar a solução ótima no JSF pode ser compreendida a partir da dificuldade de resolução do JS, reconhecidamente um dos problemas combinatoriais mais complexos. No JS tradicional, no qual cada *job* passa uma única vez por cada máquina, existem $n!$ possibilidades de sequenciamento por máquina. Como o problema envolve m máquinas o JS tem, no total, $(n!)^m$ combinações possíveis de solução. Em um JSF com flexibilidade total no qual cada *job* tem m operações, um único *job* tem m^m possibilidades de roteamento. Considerando os n *jobs* do problema são, no total, $[(m)^m]^n$ possibilidades de roteamento. Como para cada opção de roteamento ainda existe um JS a ser resolvido, e supondo que na média cada máquina deve processar m operações, o JSF com flexibilidade total com m operações por *job* tem $(n!)^m \cdot m^{n \cdot m}$ combinações possíveis de solução. Evidentemente essas quantidades de combinações do JS e do JSF englobam soluções factíveis e infactíveis, aquelas que geram grafos disjuntivos com ciclos. Com essa explosão de combinações se pode notar que, dependendo do nível de flexibilidade e do número de operações por *job*, um JSF com 5 *jobs* e 5 máquinas pode ter aproximadamente $7,41 \cdot 10^{27}$ combinações enquanto um JS tradicional com o mesmo número de *jobs* e de máquinas tem ordem bastante menor de combinações, cerca de $2,48 \cdot 10^{10}$. Detalhes adicionais a respeito do número de soluções do JSF podem ser observados no Apêndice A.

2.3 REVISÃO BIBLIOGRÁFICA

Entre os primeiros trabalhos a mencionar o JSF estão Mishra e Pandey (1989) e Brucker e Schile (1990), embora usem nomenclaturas distintas para se referir ao problema. Mishra e Pandey (1989) utilizam simulação computacional para comparar um método heurístico proposto com diferentes regras de prioridade que visam minimizar o *makespan* e a carga de trabalho média das máquinas. Essas regras podem priorizar: a operação do *job* com maior processamento a ser efetuado; a operação disponível com menor tempo de processamento; o *job* com a operação de menor tempo de processamento em relação ao tempo de processamento total do *job*; ou a operação cujo *job* possua o maior tempo de espera no sistema. Brucker e Schile (1990) propõem um algoritmo de complexidade polinomial para o JSF com 2 *jobs*. Com relação a problemas mais realistas, com quantidades maiores de *jobs*, usualmente são utilizadas técnicas alternativas, como heurísticas. Brandimarte (1993) foi o primeiro trabalho a utilizar a abordagem hierárquica. Para tanto são empregadas regras de prioridade para estabelecer as rotas dos *jobs* pelas máquinas às quais suas operações são designadas. Na fase seguinte as sequências são obtidas através da heurística Busca Tabu (BT). Evidentemente, caso a abordagem hierárquica seja escolhida, a resolução ótima de ambos os subproblemas do JSF ainda não garante a solução ótima do problema composto.

Outro aspecto relevante a ser considerado nos problemas de JSF se refere ao grau de sua flexibilidade. Kacem; Hammadi e Borne (2002) classificam os problemas em JSF com flexibilidade total e JSF com flexibilidade parcial. No JSF com flexibilidade total cada uma das operações pode ser executada em qualquer uma das máquinas. Já no JSF com flexibilidade parcial cada uma das operações está associada a um subconjunto de máquinas aptas a realizar seu processamento. Naturalmente pode haver inúmeros graus de flexibilidade entre o JSF com flexibilidade total e o JS, no qual a flexibilidade inexistente. Kacem; Hammadi e Borne (2002) ainda trazem à tona a conjectura de que as restrições nas possibilidades de atribuição no JSF com flexibilidade parcial poderiam tornar sua resolução mais difícil que no JSF com flexibilidade total.

O fato do JS ser, de acordo com Lawer *et al.* (1993), um dos mais difíceis problemas de otimização combinatória e do JSF ser, no mínimo, tão difícil quanto um JS, conduz muitos pesquisadores a aplicar e desenvolver métodos heurísticos. Vários trabalhos empregam Algoritmos Genéticos (AG). Kacem; Hammadi e Borne (2002) utilizam a abordagem hierárquica associada a um método evolutivo. Primeiramente um procedimento é executado com o intuito de definir uma máquina para executar cada operação. Esse procedimento

procura a designação mais adequada levando em consideração os tempos de processamento e a carga de trabalho das máquinas. Em seguida um algoritmo evolutivo é empregado para gerar bons sequenciamentos. Isso é feito considerando *makespan* isoladamente ou *makespan* e carga de trabalho alternadamente no decorrer das gerações.

Zhang e Gen (2005) propõem um AG baseado numa representação multiestágio na qual as operações de um *job* descrevem um caminho através das máquinas. Os experimentos envolvem JSF com flexibilidade total e JSF com flexibilidade parcial tendo como objetivos a minimização do *makespan* e da carga de trabalho total e média das máquinas. Seus resultados são superiores aos de AGs que usam outras representações e aos de um método heurístico baseado na regra *Shortest Processing Time* (SPT). Chan; Wong e Chan (2006) utilizam um AG de dois níveis para resolver o JSF com flexibilidade total através da abordagem hierárquica. Em um primeiro momento o objetivo é alocar as operações minimizando o custo de ociosidade das máquinas. Em seguida essa alocação é o ponto de partida para a geração dos sequenciamentos com o objetivo de minimizar o *makespan*.

Ho; Tay e Lai (2006) desenvolvem uma arquitetura que funde AG e aprendizagem de máquina para uma aplicação do JSF na indústria eletrônica de circuito impresso cujo objetivo é minimizar o *makespan*. Pezzella; Morganti e Ciaschetti (2008) implementam um AG cuja população inicial é gerada por procedimentos que mesclam diferentes regras de atribuição, como a alocação da operação de menor tempo de processamento à respectiva máquina. Também são utilizados procedimentos que mesclam regras de prioridade. Esses procedimentos mistos, associados à aleatoriedade, permitem a obtenção de uma população inicial mais heterogênea. Gutiérrez e García-Magariño (2011) buscam minimizar o *makespan* usando AG e heurísticas de reparação. Inicialmente o AG define as atribuições das operações às máquinas e gera sequências rústicas, resultando em soluções que não são necessariamente factíveis. Então heurísticas de reparação especializadas em determinadas restrições adequam as soluções.

Além de AG, outros métodos heurísticos são utilizados por diferentes autores. Dauzère-Pérès e Paulli (1997) utilizam BT em uma vizinhança que não distingue alterações nas atribuições ou nos sequenciamentos para minimizar o *makespan*. Zhang *et al.* (2009) hibridizam a meta-heurística *Particle Swarm Optimization* (PSO) com BT em um método multicritério de abordagem integrada. O objetivo é, ponderadamente, minimizar o *makespan*, a carga de trabalho geral e a carga máxima de trabalho entre as máquinas. Li; Pan e Liang (2010) desenvolvem um método híbrido multiobjetivo que envolve BT e *Variable Neighborhood Search* (VNS). O *makespan*, a carga de trabalho total das máquinas e a carga

de trabalho da máquina crítica são minimizados pela abordagem hierárquica que ainda utiliza uma codificação alternativa de solução. Yuan e Xu (2013b) minimizam o *makespan* utilizando Evolução Diferencial e uma Busca Local (BL) focada no caminho crítico. Os resultados obtidos são competitivos com outros métodos da literatura. Yuan e Hu (2013a) minimizam o *makespan* com um algoritmo híbrido que mescla as técnicas *Harmony Search* e *Large Neighborhood Search*. Os autores alegam que o método visa resolver problemas de grande porte, mas somente são apresentados resultados de instâncias com até 20 *jobs* e 15 máquinas. As comparações com outros métodos da literatura indicam o melhor desempenho do método utilizado. Shao *et al.* (2013) aplicam uma PSO híbrida com *Simulated Annealing* (SA) para minimizar, simultaneamente, *makespan* carregamento máximo de máquina e carregamento total das máquinas. Seus resultados indicam um melhor desempenho que outras técnicas da literatura.

Alvarez-Valdes *et al.* (2005) utilizam métodos heurísticos construtivos em uma aplicação na indústria de vidro. Algumas operações críticas são efetuadas por trabalhadores e, para permitir a resolução, eles figuram no problema assim como as máquinas. O objetivo é minimizar os custos associados ao instante de término de cada *job*. Vilcot e Billaut (2008) fazem uso de BT e AG em uma aplicação industrial que minimiza o *makespan* e o *lateness* máximo de forma combinada. Uma aplicação do JSF minimizando o atraso na indústria de armamentos militares é apresentada por Chen *et al.* (2008). O ambiente abordado permite que as operações dos *jobs* tenham sequências alternativas e que os *jobs* tenham relações de precedência entre si. O método de resolução hierárquica envolve uma etapa de seleção de máquinas e outra de sequenciamento, ambas baseadas na regra *Earliest Due Date* (EDD). Outras regras também são utilizadas para desempate.

Scrich; Armentano e Laguna (2004) é, de acordo com o levantamento realizado neste estudo, o primeiro trabalho a considerar atraso total no JSF. Soluções iniciais são geradas pela aplicação de regras de prioridade e aprimoradas por BT. É adotada a abordagem hierárquica e o trabalho inclui um procedimento *multi-start*. Tay e Ho (2008) aliam regras de prioridade a *Genetic Programming* para resolver um JSF multiobjetivo que envolve, simultaneamente, *makespan*, atraso médio e *mean flow time*.

Gholami e Zandieh (2009) utilizam simulação para avaliar a robustez das soluções fornecidas por um AG. O ambiente inclui possíveis quebras baseadas nas meias-vidas das máquinas. Uma população é gerada aleatoriamente e cada indivíduo é submetido a uma etapa de simulação na qual as quebras podem ocorrer. Os indivíduos são avaliados pela simulação e, então, acontecem os cruzamentos e as mutações. As soluções são representadas por

cromossomos com duas partes, uma registrando as atribuições e outra, os sequenciamentos. As simulações se intercalam com os cruzamentos e com as mutações até que o critério de parada seja alcançado. O objetivo do trabalho é minimizar isoladamente o atraso médio ou o *makespan*, encontrando uma solução suficientemente robusta para não ser muito afetada por eventuais quebras de máquinas. Desse modo, nem sempre a solução que tenha o menor atraso médio ou o menor *makespan* em um ambiente sem quebras será a melhor solução para o objetivo proposto por Gholami e Zandieh (2009).

Mousakhani (2013) minimiza o atraso total em um JSF com tempos de *setup* dependentes da sequência utilizando *Iterated Local Search* (ILS). É utilizado um esquema de codificação das soluções composto por duas listas, uma para as sequências dos *jobs* e outra para as atribuições às máquinas. A partir dessa codificação são realizados os movimentos aleatórios da BL que se baseia na inserção de cada *job* em uma posição aleatória diferente da atual. Os experimentos envolvem instâncias com até 20 *jobs* e 10 máquinas, as quais os autores consideram de grande porte. Os resultados obtidos são comparados com os fornecidos por uma BT desenvolvida para minimizar o *makespan* e uma VNS desenvolvida para minimização multiobjetivo, indicando melhor desempenho do método proposto.

Baykasoğlu e Ozbakir (2010) propõem representar as soluções através de uma simbologia chamada gramatical. A representação gramatical envolve símbolos que representam, por exemplo, os planos alternativos dos *jobs* e as máquinas que podem processar cada operação. São associadas probabilidades às escolhas desses planos e das máquinas, de acordo com algum critério como, por exemplo, seus tempos de processamento. O trabalho usa a representação gramatical nas soluções de uma BT multiobjetivo e somente resolve as 4 instâncias propostas por Kacem; Hammadi e Borne (2002). Cada solução envolve um plano, as máquinas e uma regra de prioridade. Uma solução vizinha é obtida ao se sortear um plano alternativo e outras máquinas. O objetivo é identificar a influência da variação da flexibilidade do JSF sobre os resultados das regras de prioridade. O artigo avalia diferentes regras para o atraso comparando-as em pares e apresenta apenas as diferenças percentuais entre cada duas delas, omitindo os valores absolutos alcançados por cada regra. O trabalho conclui que, à medida que a flexibilidade aumenta, os resultados de diferentes regras de prioridade se aproximam. Pelo levantamento bibliográfico realizado se constatou que na literatura existe carência de trabalhos sobre JSF que envolvam atraso, em especial, que envolvam atraso total.

Em contraste com o grande volume de estudos que evocam técnicas heurísticas, a quantidade de trabalhos que propõem formulações matemáticas é mais limitada. Nesse campo

o artigo de Fattahi; Mehrabad e Jolai (2007) é relevante, já que nele é proposto um modelo matemático para o JSF com o objetivo de minimizar o *makespan*. A modelagem utiliza variáveis binárias que determinam a posição de cada operação na máquina, como em Wagner (1959). Além do modelo, Fattahi; Mehrabad e Jolai (2007) propõem um conjunto de 20 instâncias específicas do JSF, sendo que 10 delas têm de 2 a 4 *jobs* e 10 têm de 5 a 12 *jobs*. O modelo matemático elaborado é utilizado para resolver tais instâncias com o tempo de execução computacional limitado a 1 hora. As soluções ótimas, obtidas para os problemas de menor porte, e os limitantes superiores e inferiores, obtidos nos demais problemas, são utilizados em comparações com resultados alcançados por métodos heurísticos. Os métodos heurísticos utilizados por Fattahi; Mehrabad e Jolai (2007) são BT e SA. O trabalho explora a abordagem integrada aplicando BT e SA isoladamente. Já a abordagem hierárquica usa BT e SA isolados ou combinados. Neste último caso uma meta-heurística gera as atribuições e a outra, os sequenciamentos.

Özgüven; Özbakir e Yavuz (2010) propõem outro modelo matemático para a minimização do *makespan* no JSF a partir de modificações do modelo de Manne (1960) para o JS que permitem incorporar a flexibilidade de máquinas. Esse modelo, consideravelmente mais conciso, é confrontado com o proposto por Fattahi; Mehrabad e Jolai (2007) por meio da resolução de suas 20 instâncias através de um *software* de programação matemática. Os resultados, apesar de serem obtidos em computadores com configurações distintas, indicam que o modelo de Özgüven; Özbakir e Yavuz (2010) tem melhor desempenho. Esse modelo obtém solução ótima em todos os 10 problemas de menor porte, assim como seu concorrente, porém em menor tempo. Além disso, alcança a solução ótima em 5 dos 10 outros problemas, enquanto Fattahi; Mehrabad e Jolai (2007) não consegue nenhum ótimo nesse subconjunto. Nas 5 instâncias em que nenhum dos modelos atinge a solução ótima, o modelo de Özgüven; Özbakir e Yavuz (2010) fornece melhores limitantes.

Demir e İşleyen (2013) fazem um estudo comparativo envolvendo três categorias de modelos matemáticos para o JSF. Na primeira categoria as variáveis binárias determinam, como em Wagner (1959), a posição que uma dada operação ocupa em uma determinada máquina. A segunda categoria, assim como o modelo de Manne (1960) para o JS, utiliza variáveis binárias que assumem valores com base na precedência entre operações que estejam em uma mesma máquina. Na terceira categoria as variáveis binárias definem se dada operação é processada em certa máquina em determinado período, como em Bowman (1959). Demir e İşleyen (2013) testam modelos das três categorias e os resultados indicam que o modelo mais

rápido e com os melhores resultados é o proposto por Özgüven; Özbakir e Yavuz (2010), da segunda categoria.

Um novo modelo para a minimização do *makespan* é proposto em Birgin *et al.* (2013). A formulação minimiza o término da última operação processada sem distinguir os *jobs* aos quais as mesmas pertencem. Os experimentos computacionais indicam um melhor desempenho desse modelo em relação ao proposto por Özgüven; Özbakir e Yavuz (2010).

Evidentemente problemas com a complexidade do JS ou do JSF exigem técnicas sofisticadas para que resultados razoáveis possam ser obtidos em um tempo aceitável. Por tal razão a exploração de métodos não exatos é observada com tão alta frequência. Em muitas aplicações práticas é preferível obter boas soluções rapidamente, mesmo que não sejam ótimas, a esperar muito tempo pela resolução exata. Contudo, é necessário que os resultados oriundos desses métodos não exatos, especialmente heurísticos, possam ser avaliados de modo que determinados níveis de qualidade de solução sejam atendidos. Por esse motivo o desenvolvimento e o aprimoramento de modelos que permitam a resolução exata são fundamentais. Modelos bem elaborados, aliados à grande evolução dos recentes pacotes comerciais baseados em métodos como *branch-and-bound* e *branch-and-cut*, podem ser capazes de fornecer, senão a solução ótima, limitantes satisfatórios para muitos problemas. Esses limitantes então podem ser utilizados para balizar a avaliação de métodos heurísticos, distinguindo, com mais clareza, bons e maus resultados. Assim, os resultados obtidos através de modelos matemáticos do JSF devem servir para avaliar os métodos heurísticos em desenvolvimento, os quais visam amenizar a escassez existente na literatura a respeito do JSF com o critério de minimização do atraso total.

3 FORMULAÇÕES MATEMÁTICAS

Neste capítulo são apresentados os modelos matemáticos de Programação Linear Inteira Mista (PLIM) para o *Job Shop* Flexível (JSF), sendo que dois deles objetivam minimizar o *makespan* e um, o atraso total. A primeira formulação foi proposta por Özgüven; Özbakir e Yavuz (2010) e minimiza o *makespan*. A segunda formulação é uma adaptação da primeira para incorporar a declaração explícita das relações de precedência. Já a terceira formulação tem alterações que permitem minimizar o atraso total. Essas formulações são apresentadas na sequência.

3.1 MODELOS PARA A MINIMIZAÇÃO DO *MAKESPAN*

O primeiro modelo para minimização do *makespan* é o proposto por Özgüven; Özbakir e Yavuz (2010). Ele deriva da formulação de Manne (1960) e trabalha com um menor número de variáveis e com muito menos restrições que o modelo de Fattahi; Mehrabad e Jolai (2007). Além disso, conforme experimentos computacionais realizados por Demir e İşleyen (2013), possui um desempenho superior. O objetivo é minimizar o *makespan* garantindo que todas as operações sejam processadas, uma por vez em cada máquina, na ordem determinada pelos *jobs*.

Índices e conjuntos:

J	conjunto de <i>jobs</i> ;
M	conjunto de máquinas;
O	conjunto de operações;
i, i'	<i>jobs</i> ($i, i' \in J$);
j, j'	operações ($j, j' \in O$);
k	máquinas ($k \in M$);
O_i	conjunto ordenado das operações do <i>job</i> i ($O_i \subseteq O$);
O_{ij}	operação j do <i>job</i> i ($O_{ij} \in O_i$);
$O_{if(i)}$	primeiro elemento de O_i ;
$O_{il(i)}$	último elemento de O_i ;
M_j	conjunto de máquinas alternativas para processar a operação j ($M_j \subseteq M$);
$M_j \cap M_{j'}$	conjunto de máquinas onde ambas as operações j e j' podem ser processadas.

Parâmetros:

t_{ijk} tempo de processamento da operação O_{ij} na máquina k ;

L número grande;

Variáveis de decisão:

X_{ijk} assume 1 se a máquina k é selecionada para a operação O_{ij} e 0, em caso contrário;

S_{ijk} instante de início de processamento da operação O_{ij} na máquina k ;

C_{ijk} instante de término de processamento da operação O_{ij} na máquina k ;

$Y_{iji'j'k}$ assume 1 se a operação O_{ij} precede $O_{i'j'}$ na máquina k e 0, caso contrário;

C_i instante de término de processamento do *job* i ;

$C_{\max}$ máximo instante de término de processamento dentre os *jobs*, *makespan*.

Modelo matemático:

Função-objetivo:

$$\text{Minimizar } C_{\max} \quad (1)$$

Sujeito a:

$$\sum_{k \in M_j} X_{ijk} = 1 \quad \forall i \in J, \forall j \in O_i \quad (2)$$

$$S_{ijk} + C_{ijk} \leq (X_{ijk}) \cdot L \cdot 2 \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (3)$$

$$C_{ijk} \geq S_{ijk} + t_{ijk} - (1 - X_{ijk}) \cdot L \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (4)$$

$$S_{ijk} \geq C_{i'j'k} - (Y_{iji'j'k}) \cdot L \quad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \quad (5)$$

$$S_{i'j'k} \geq C_{ijk} - (1 - Y_{iji'j'k}) \cdot L \quad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \quad (6)$$

$$\sum_{k \in M_j} S_{ijk} \geq \sum_{k \in M_{j-1}} C_{i,j-1,k} \quad \forall i \in J, \forall j \in O_i - \{O_{if(i)}\} \quad (7)$$

$$C_i \geq \sum_{k \in M_{O_{il(i)}}} C_{i,O_{il(i)},k} \quad \forall i \in J \quad (8)$$

$$C_{\max} \geq C_i \quad \forall i \in J \quad (9)$$

$$X_{ijk} \in \{0,1\} \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (10)$$

$$S_{ijk} \geq 0 \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (11)$$

$$C_{ijk} \geq 0 \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (12)$$

$$Y_{iji'j'k} \in \{0,1\} \quad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \quad (13)$$

$$C_i \geq 0 \quad \forall i \in J \quad (14)$$

A função-objetivo (1) minimiza o máximo dos instantes de término dentre todos os *jobs*, que é exatamente o *makespan*. O conjunto de restrições do tipo (2) garante que cada operação seja atribuída a exatamente uma máquina dentre as habilitadas para seu processamento. As restrições do tipo (3) asseguram que, caso uma determinada operação não seja atribuída a certa máquina, seus instantes de início e término de processamento nessa máquina sejam nulos. As restrições do tipo (4) fazem com que, caso uma determinada operação seja atribuída a certa máquina, seu instante de término seja, no mínimo, a soma de seu instante de início de processamento e de seu tempo de processamento naquela máquina. As restrições dos tipos (5) e (6) evitam que operações alocadas a uma mesma máquina sejam colocadas em execução concorrentemente. As restrições do tipo (7) exigem que as operações sejam processadas na ordem determinada por cada *job*. As restrições do tipo (8) forçam que o instante de término de cada *job* seja maior ou igual ao instante de término de sua última operação. As restrições do tipo (9) definem o *makespan*. As restrições de (10) a (14) definem os domínios das variáveis, sendo que as restrições (10) e (13) tratam a condição de integralidade. O modelo é apresentado com correções tipográficas nos somatórios das restrições dos tipos (7) e (8) com relação ao mostrado em Özgüven; Özbakir e Yavuz (2010) e com a inclusão de um fator de multiplicação nas restrições (3) a ser discutido na sequência juntamente com a definição do valor de L .

O segundo modelo para minimização do *makespan*, apresentado a seguir e descrito em Birgin *et al.* (2011), resulta de adaptações e aprimoramentos do modelo original de Özgüven; Özbakir e Yavuz (2010). O modelo original trabalha com uma lista ordenada de operações para cada *job*. Isso fica especialmente evidente nas restrições (7) e (8). Porém, a representação clássica do *Job Shop* (JS) e do JSF envolve os arcos de precedência. Assim as adaptações têm o intuito de fazer com que o modelo trabalhe com as precedências declaradas explicitamente. Com isso o modelo se torna capaz de resolver problemas em que uma determinada operação possua mais de uma operação precedente imediata. Tal situação ocorre quando existem operações de montagem. A Figura 7 ilustra diferentes configurações de *jobs* com as quais o modelo adaptado pode lidar.

Figura 7 - Diferentes configurações de *jobs*.

Fonte: elaborada pelo autor.

Na Figura 7, o *job 1* possui a distribuição clássica dos arcos do JSF e é constituído por uma única sequência linear de operações em que, a partir da segunda operação, cada operação tem exatamente uma única operação precedente. O modelo de Özgüven; Özbakir e Yavuz (2010) foi desenvolvido para esse esquema de *job*, mencionando inclusive as operações iniciais, $O_{if(i)}$, e finais, $O_{il(i)}$.

O *job 2* da Figura 7 possui três sequências paralelas de operações formadas pelas operações de 1 a 3, de 4 a 6 e de 7 a 9. Esses três ramos do *job 2* se encontram na operação 10, uma operação de montagem, a partir da qual segue uma única sequência de operações até a finalização do *job*. Esta configuração de *job* é chamada por Conway; Maxwell e Miller (1967) de *assembly shop* e podem existir diversos níveis de montagem. O conceito de operação de montagem pode ser útil em situações em que partes processadas independentemente são unidas para formar um único produto final.

No *job 3*, em vez de montagem, existe a operação 2 de desmontagem, a partir da qual seguem dois ramos independentes. Ela possui uma única operação predecessora, mas, diferentemente das operações anteriores, possui duas operações sucessoras. Um exemplo de aplicação para este tipo de configuração de *job* seria o uso de uma peça que, depois de alguns processamentos, seria dividida e daria origem a dois produtos de um mesmo pedido. Já o *job 4*, além de ter a operação de montagem 7 ainda possui uma operação de desmontagem, a

operação 2. Por fim o *job* 5 apresenta dois ramos de operações não conectados por qualquer arco conjuntivo. Esta configuração poderia ser usada para representar um pedido com data de entrega única composto por vários produtos independentes. As configurações dos *jobs* 3 e 4 são apresentadas por Birgin *et al.* (2013). No presente trabalho se denomina *Job Shop Flexível com Montagem e Desmontagem* (JSFMD) problemas que possuem, simultaneamente, *jobs* com operações de desmontagem, como os *jobs* 2 e 4 da Figura 7, e montagem, como os *jobs* 3 e 4.

Problemas que possuam operações distribuídas similarmente às operações dos *jobs* 2, 3, 4 ou 5 da Figura 7 não podem ser resolvidos pelo modelo de Özgüven; Özbakir e Yavuz (2010). Contudo, no ambiente industrial é comum haver *jobs* nos quais partes processadas separadamente são unidas em um único produto. Assim o objetivo principal do modelo adaptado é atender essas situações. Os ambientes descritos por Chen *et al.* (2008) e Vilmont e Billaut (2008) são exemplos de onde esse modelo poderia ser utilizado. Para compor o modelo adaptado devem ser considerados adicionalmente:

Índices e conjuntos:

k' máquinas ($k' \in M$);

P_{ij} conjunto das operações que devem preceder a operação O_{ij} .

Parâmetro:

L número adequadamente grande fornecido por $\sum_{O_{ij} \in O} \max_{k \in M_j} (t_{ijk})$.

Özgüven; Özbakir e Yavuz (2010) afirmam que L deve ser um número grande, mas seu valor não é indicado. Contudo, valores excessivamente grandes podem incorrer em erros numéricos. Por outro lado, valores não suficientemente grandes podem fazer com que as restrições (4), (5) e (6) inviabilizem a obtenção de soluções. Então, é conveniente definir um meio pelo qual valores adequados de L possam ser calculados para cada instância. Neste trabalho é proposto que L seja a soma do tempo máximo de processamento de todas as operações dentre as máquinas que podem executar cada uma delas. Com tal valor de L , as restrições do tipo (3) sem multiplicação pelo fator 2, como apresentado originalmente em Özgüven; Özbakir e Yavuz (2010), poderiam eliminar soluções ótimas ou mesmo impedir que qualquer solução factível fosse obtida. Como exemplo, se pode considerar um problema com um único *job* e nenhuma flexibilidade. L seria a soma dos tempos de processamento, o que equivaleria ao *makespan* mínimo. Então bastaria que a última operação fosse antecedida por qualquer operação com tempo de processamento não nulo para que qualquer solução se

inviabilizasse. Isso aconteceria pois, nesse caso, C_{ijk} jamais seria menor que L e, com um S_{ijk} não nulo, as restrições do tipo (3) sem a multiplicação por 2 não poderiam ser atendidas. Para evitar isso, a restrição (3) deve ser substituída pelas restrições (15) e (16):

$$S_{ijk} \leq X_{ijk} \cdot L \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (15)$$

$$C_{ijk} \leq X_{ijk} \cdot L \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (16)$$

Com esse desmembramento da restrição (3), mesmo na situação mencionada anteriormente, a solução ótima seria alcançada.

Para que o modelo possa lidar com os arcos de precedência, as restrições (7) e (8) devem ser substituídas, respectivamente, pelas restrições a seguir:

$$\sum_{k \in M_j} S_{ijk} \geq \sum_{k' \in M_{j'}} C_{ij'k'} \quad \forall i \in J, \forall j \in O_i, \forall O_{ij'} \in P_{ij} \quad (17)$$

$$C_i \geq \sum_{k \in M_j} C_{ijk} \quad \forall i \in J, \forall j \in O_i \quad (18)$$

Desse modo, em vez do modelo considerar que cada operação de um *job* deve iniciar após o término da operação listada anteriormente na série, como nas restrições do tipo (7), as restrições do tipo (17) exigem que cada operação inicie seu processamento após a finalização das operações que a precedem. As restrições do tipo (18), em vez de basear o instante de término dos *jobs* na última operação de uma lista, como em (8), verifica todas as operações do *job*.

3.2 MODELO ADAPTADO PARA A MINIMIZAÇÃO DO ATRASO TOTAL

Diferentemente dos dois modelos anteriormente descritos, o modelo a seguir tem o objetivo de minimizar o atraso total. Ele deriva do modelo adaptado que minimiza o *makespan* e, portanto, é capaz de encontrar a solução ótima para problemas com operações de montagem e de desmontagem. As alterações efetuadas no modelo adaptado envolvem a função-objetivo e algumas novas restrições, além de exigir novos parâmetros e variáveis de decisão. Por essa razão ele é apresentado integralmente. Os parâmetros adicionais necessários para a resolução do JSF são as datas de entrega, ou *due dates*, solicitadas para cada *job*.

Parâmetros:

d_i data de entrega do *job* i ;

Variáveis de decisão:

T_i atraso do *job* i ;

Modelo matemático:

Função-objetivo:

$$\text{Minimizar } \sum_{i \in J} T_i \quad (19)$$

Sujeito a:

$$\sum_{k \in M_j} X_{ijk} = 1 \quad \forall i \in J, \forall j \in O_i \quad (20)$$

$$S_{ijk} \leq X_{ijk} \cdot L \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (21)$$

$$C_{ijk} \leq X_{ijk} \cdot L \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (22)$$

$$C_{ijk} \geq S_{ijk} + t_{ijk} - (1 - X_{ijk}) \cdot L \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (23)$$

$$S_{ijk} \geq C_{i'j'k} - (Y_{iji'j'k}) \cdot L \quad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \quad (24)$$

$$S_{i'j'k} \geq C_{ijk} - (1 - Y_{iji'j'k}) \cdot L \quad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \quad (25)$$

$$\sum_{k \in M_j} S_{ijk} \geq \sum_{k \in M_{j'}} C_{ij'k} \quad \forall i \in J, \forall j \in O_i, \forall O_{ij'} \in P_{ij} \quad (26)$$

$$C_i \geq \sum_{k \in M_j} C_{ijk} \quad \forall i \in J, \forall j \in O_i \quad (27)$$

$$T_i \geq C_i - d_i \quad \forall i \in J \quad (28)$$

$$X_{ijk} \in \{0,1\} \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (29)$$

$$S_{ijk} \geq 0 \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (30)$$

$$C_{ijk} \geq 0 \quad \forall i \in J, \forall j \in O_i, \forall k \in M_j \quad (31)$$

$$Y_{iji'j'k} \in \{0,1\} \quad \forall i < i', \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \quad (32)$$

$$C_i \geq 0 \quad \forall i \in J \quad (33)$$

$$T_i \geq 0 \quad \forall i \in J \quad (34)$$

A função-objetivo (19) minimiza o atraso total, obtido pela soma dos atrasos de todos os *jobs*. Os conjuntos de restrições de (20) a (27) têm as mesmas funções que lhes cabe no modelo adaptado. As restrições do tipo (28) fazem com que o atraso de cada *job* seja maior ou igual à diferença entre seu instante de término e sua data de entrega. Contudo, existe a exigência de que as variáveis T_i sejam não negativas, além do fato da função-objetivo minimizar o somatório de todas as variáveis T_i . A associação dessas três condições faz com que, caso o término do processamento de um *job* extrapole sua data de entrega, seu atraso seja

dado pela diferença entre seu instante de término e sua data de entrega. Em caso contrário, esse atraso assume valor igual a zero. Os conjuntos de restrições de (29) a (34) determinam os domínios das variáveis envolvidas.

Esse modelo para minimização do atraso total pode ainda ser facilmente alterado de modo a minimizar o atraso ponderado. Para tanto basta que a função-objetivo minimize o somatório, para todo $i \in J$, da multiplicação de T_i pelo respectivo peso atribuído ao *job* i .

4 HEURÍSTICAS CONSTRUTIVAS

Este capítulo detalha os métodos construtivos implementados para o *Job Shop* Flexível (JSF). Tais métodos são capazes de fornecer, com rapidez, soluções factíveis para o problema com o critério de minimização do atraso total. São descritas as regras de prioridade obtidas na literatura e as regras de prioridade propostas. Além disso, são propostas estratégias de exploração de características do JSF que permitem encontrar melhores soluções. Também são discriminados os passos do algoritmo construtivo, baseado em regras de prioridade, proposto neste trabalho.

Considerando que o *Job Shop* (JS), de acordo com Garey; Johnson e Sethi (1976), pertence à classe de problemas NP-Difíceis e que o JSF é, no mínimo, tão difícil quanto o JS, o emprego de métodos exatos encontra limitações computacionais. Nessa classe de problemas os métodos de enumeração implícita como *branch-and-bound* e *branch-and-cut* são úteis quando se trabalha com instâncias de porte reduzido. Contudo, à medida que o número de elementos envolvidos no problema aumenta, essas técnicas se tornam inadequadas, pois não conseguem prover soluções satisfatórias em tempo computacional aceitável. Por essa razão é necessário lançar mão de métodos alternativos, especialmente heurísticos.

Os métodos heurísticos são amplamente difundidos na literatura que trata dos mais complexos problemas de otimização e são classificados em categorias, conforme Zanakis; Evans e Vazacopoulos (1989). Entre as categorias de métodos heurísticos as mais comuns são as heurísticas construtivas e as heurísticas de melhoria. Na sua grande maioria, as heurísticas de melhoria, apesar de serem capazes de obter bons resultados, precisam de um ponto de partida. Esse ponto de partida é uma solução inicial, a partir da qual melhores soluções são alcançadas. Assim, a fase de melhoria usualmente é precedida por uma fase construtiva, geralmente constituída por passos menos complexos e, por isso, mais rápida. Sempre que possível, se procura fornecer uma solução inicial factível às heurísticas de melhoria, embora existam aquelas que iniciam seu processo a partir de soluções infactíveis a serem reparadas.

Pelo fato dos problemas de programação de tarefas estarem entre os mais complexos da literatura é comum que métodos simples e efetivos sejam empregados na geração de soluções. Dentre esses métodos se pode citar o *list scheduling algorithm*, ou *dispatch procedure*, método de programação de operações bastante utilizado em ambientes de produção complexos como o JS. Nesse método, cada vez que uma máquina se torna ociosa, é selecionada para ocupá-la a operação de maior prioridade dentre as operações que podem ser

processadas por ela. A definição da operação prioritária se dá através de uma regra de prioridade. Conforme Cheng; Gen e Tsujimura (1996), regras de prioridade provavelmente são os métodos heurísticos mais utilizados na resolução de problemas de programação da produção. Segundo Panwalkar e Iskander (1977), o desempenho das regras de prioridade está fortemente ligado ao critério de desempenho utilizado.

Nesta pesquisa somente métodos construtivos que geram soluções factíveis são considerados. Na sequência são apresentadas as regras de prioridade e as estratégias propostas para exploração das características do problema. Também é descrita a estrutura da heurística construtiva utilizada neste trabalho.

4.1 REGRAS DE PRIORIDADE ADAPTADAS DA LITERATURA

Esta seção traz as regras de prioridade já exploradas em outros trabalhos da literatura, não necessariamente com o JSF, mas também com outros problemas de programação de tarefas envolvendo a minimização do atraso. Tais regras são descritas na sequência juntamente com as propostas de adaptação ao JSF, quando necessárias.

i) *Shortest Processing Time*

A *Shortest Processing Time* (SPT), conforme Panwalkar e Iskander (1977), seleciona para ser alocada a operação que, dentre as disponíveis, possui o menor tempo de processamento iminente. Desse modo, sempre que uma máquina se torna ociosa ela passa a ser ocupada pela operação ainda não alocada que primeiramente poderá ser finalizada. Baker (1984) menciona que, apesar da SPT não considerar as datas de entrega, muitas vezes é capaz de atendê-las.

ii) *Minimum Slack Time*

A regra *Minimum Slack Time* (MST) seleciona a operação do *job i* com menor espaço de tempo disponível para processamento no instante t , $MST(i,t)$, que é fornecido pela equação:

$$MST(i,t) = d_i - t - \sum_{j \in G_i} t_j \quad (35)$$

onde $d_i - t$ fornece a margem de tempo restante para o processamento do *job i*; t é o instante atual considerado pelo *list scheduling algorithm*; e G_i é o conjunto das operações ainda não processadas do *job i*. Para a resolução do JSF, para cada operação, o tempo de processamento t_j foi substituído pela média $\overline{t_{jk}}$, obtida pela divisão da soma dos tempos de processamento

dessa operação nas máquinas capazes de processá-la, tempos t_{jk} , pela quantidade dessas máquinas alternativas.

iii) *Smallest Critical Ratio*

A *Smallest Critical Ratio* (SCR) é semelhante à MST, mas, em vez de uma subtração, utiliza uma razão, conforme segue:

$$SCR(i,t) = \frac{d_i - t}{\sum_{j \in G_i} t_j} \quad (36)$$

Assim como na MST, para aplicação ao JSF os tempos de processamento t_j foram substituídos pelas médias $\overline{t_{jk}}$.

iv) *Earliest Due Date*

A regra *Earliest Due Date* (EDD) se baseia somente nas datas de entrega dos *jobs*. Conforme Vepsalainen e Morton (1987), quando uma máquina fica disponível, é selecionada para ocupá-la a primeira operação ainda não alocada do *job* cuja data de entrega esteja mais próxima. Seu objetivo é favorecer os *jobs* que se estima estarem sob maior risco de sofrerem atraso.

v) *Modified Due Date*

Semelhantemente à EDD, a regra de prioridade *Modified Due Date* (MDD), usada por Baker e Bertrand (1982), também se baseia nas datas de entrega, porém sem se apoiar fixamente à data de entrega inicial, d_i , do *job* i . Em vez disso, a regra avalia se, em determinado instante, um dado *job* tem condição ou não de ser entregue no prazo. Se d_i for menor que a soma do instante atual aos tempos de processamento das operações subsequentes do *job*, a data de entrega d_i , necessariamente, não poderá ser cumprida. Então essa data de entrega é modificada para a data de entrega mais próxima possível, fornecida pela soma anteriormente mencionada. Com isso se evita trabalhar com uma data de entrega inicial, mais apertada, que não poderá ser atendida. Cada data de entrega modificada, MDD, é fornecida por:

$$d_i' = \max \left\{ d_i, t + \sum_{j \in G_i} t_j \right\} \quad (37)$$

No JSF o somatório da Equação (37) considera, para cada operação, o menor dentre os seus tempos de processamento nas diversas máquinas alternativas, conforme segue:

$$d_i' = \max[d_i, t + \sum_{j \in G_i} \min(t_{jk})] \quad (38)$$

Desse modo, quando d_i não pode mais ser atendido, d_i' sempre indica o menor instante em que o *job* poderá ser finalizado. Sempre é escolhida a operação cujo *job* possua o menor valor d_i' .

vi) *Modified Operation Due Date*

A regra de prioridade *Modified Operation Due date* (MOD), proposta por Baker (1984), é a MDD alterada para trabalhar com datas de entrega por operação. A data de entrega modificada estimada para cada operação, d_{ij}' , do *job* i é obtida do seguinte modo:

$$d_{ij}' = \max\{d_{ij}, t + t_{ij}\} \quad (39)$$

onde as datas de entrega d_{ij} são fornecidas por:

$$d_{ij} = d_{i,j-1} + \frac{(d_i - t) \cdot t_{ij}}{\sum_{O_{ij} \in O_i} t_{ij}}, \quad (40)$$

sendo que, somente na MOD, j é a j -ésima operação do *job* i que no instante t está prestes a ser processada, com tempo de processamento t_{ij} ; e $(d_i - t)$ é a margem restante de tempo. Se todos os *jobs* estão disponíveis no instante inicial zero, nesse instante se tem a margem igual a data de entrega. Nesse caso, devido à convenção de que d_{i0} é o instante de liberação do *job* i , d_{i0} também é zero. Em suma, cada operação recebe como tempo para sua execução uma fração do prazo do *job* que é proporcional à fração que seu tempo de processamento representa no somatório de todos os tempos de processamento de todas as operações do *job*. Neste trabalho os tempos t_{ij} de cada operação na regra MOD aplicada ao JSF são substituídos pelos tempos médios $\overline{t_{ijk}}$. Em cada alocação é selecionada a operação com menor MOD.

vii) *Priority Rule for Total Tardiness adaptada*

A *Priority Rule for Total Tardiness* adaptada (PRTTa), proposta por Mainieri e Ronconi (2013), é uma adaptação para *flowshop* com máquinas paralelas da regra *Priority Rule for Total Tardiness* proposta por Chu e Portman (1992) para máquina única. Essa regra de prioridade, além de considerar as operações disponíveis em um dado instante, antevê operações que se tornarão disponíveis em instantes posteriores. Com isso a PRTTa permite a inserção de espera. Em comparações com outras regras, como a MDD, a PRTTa propiciou resultados melhores no ambiente *flowshop* flexível. Nela se considera adicionalmente:

h índice de estágio do ambiente *flowshop*;

r_{ih}	instante de liberação do <i>job i</i> para o estágio <i>h</i> ;
d_{ih}	data de entrega do estágio <i>h</i> do <i>job i</i> ;
t_{ih}	tempo de processamento do estágio <i>h</i> do <i>job i</i> ;
α	constante que pondera o primeiro termo da regra.

Cada valor d_{ih} é obtido pela subtração dos tempos dos estágios posteriores da data de entrega do *job*. A regra PRTTa então pode ser usada para calcular o valor $PRTTa(i,h,t)$ que determina a prioridade do *job i*, no estágio *h*, no instante *t*, do seguinte modo:

$$PRTTa(i,h,t) = \alpha \cdot \max(r_{ih}, t) + \max[\max(r_{ih}, t) + t_{ih}, d_{ih}] \quad (41)$$

O primeiro termo da regra, considerando $\alpha > 0$, busca penalizar os *jobs* que não estão disponíveis no instante *t*. O segundo termo visa, ao mesmo tempo, considerar a priorização da atualização da data de entrega e penalizar *jobs* não disponíveis. Se $r_{ih} < t$, o segundo termo se torna idêntico à outra regra proposta por Mainieri e Ronconi (2013) para o *flowshop*, uma adaptação da MDD com datas de entrega por estágio que faz atualização de data de entrega. A todas as prioridades é acrescentado o valor $\alpha \cdot t$, mas somente nos casos em que a liberação é posterior a *t* há o acréscimo do valor $\alpha \cdot (r_{ih} - t)$, penalizando liberações mais distantes de *t*. O estágio do *job* perde prioridade se possuir data de entrega muito à frente de *t* ou se demorar muito para terminar. Em Mainieri e Ronconi (2013) pode haver d_{ih} negativo.

No JSF, em vez de estágios, são consideradas operações. Portanto a adaptação proposta neste trabalho define como d_j a data de entrega da operação *j* e como r_j seu instante de liberação. O cálculo de cada d_j é similar ao usado por Mainieri e Ronconi (2013), ou seja, a data de entrega do seu *job* menos o tempo médio de processamento das operações restantes pode haver d_j negativo. Nesse processo os tempos de processamento por operação são as médias de seus processamentos nas diversas máquinas alternativas, como na MOD. A PRTTa aplicada ao JSF antevê como operações candidatas aquelas cujas predecessoras já estão alocadas em alguma máquina, mas que ainda não foram finalizadas e ainda não estão disponíveis. Nesses casos o instante de liberação r_j é o instante em que a operação predecessora será concluída. O cálculo da adaptação proposta é feito do seguinte modo:

$$PRTTa(i,j,k,t) = \alpha \cdot \max(r_j, t) + \max[\max(r_j, t) + t_{jk}, d_j] \quad (42)$$

No primeiro termo α penaliza operações ainda não disponíveis. Operações cujas predecessoras estão concluídas têm r_j igual a *t*. Operações que possuem predecessoras ainda não alocadas não entram na lista de operações candidatas. No segundo termo, o potencial

instante de término da operação é comparado com sua data de entrega estimada. É selecionada a operação com menor valor $PRTTa(i,j,k,t)$.

4.2 IDENTIFICANDO AS CARACTERÍSTICAS DO PROBLEMA

Além de utilizar métodos efetivos e rápidos para a geração de soluções, também é extremamente interessante, quando possível, usar informações a respeito do problema para aprimorar os resultados obtidos por tais métodos. Assim, identificar e explorar algum conhecimento a respeito do problema pode permitir que soluções de qualidade bastante superior sejam alcançadas sem aumento significativo do tempo computacional. Desse modo, na sequência são introduzidas duas estratégias para explorar características específicas do JSF com o intuito de obter soluções de melhor qualidade. Essas estratégias podem ser associadas a diferentes regras de prioridade ou mesmo a outros métodos de resolução, guiando o processo de definição das atribuições das operações às máquinas e de sequenciamentos.

4.2.1 Afinidade entre operações e máquinas

A primeira estratégia de exploração de características do JSF se relaciona à intuitiva conveniência de que pode ser vantajoso atribuir as operações às máquinas que possam processá-las consumindo menos tempo. Entretanto, para que bons resultados sejam alcançados, é preciso que essa percepção seja explorada na medida correta. Assim, a alocação de certa operação a uma máquina pode ser induzida, mas não deve ser artificialmente imposta. Neste trabalho é denominada afinidade, A_{jk} , a conveniência de se alocar uma operação j a uma máquina k tendo como base a relação existente entre o tempo de processamento da operação j na máquina k , t_{jk} , e os tempos de processamento dessa mesma operação j em suas demais máquinas alternativas. A afinidade A_{jk} entre a operação j e a máquina k é expressa pela relação:

$$A_{jk} = \frac{t_{jk}}{\sum_{k \in M_j} t_{jk}} \quad (43)$$

Essa relação varia de 0 a 1 e quanto menor seu valor, maior é a afinidade existente entre a operação j e sua máquina alternativa k . Mantendo a coerência, a afinidade entre uma operação com uma única máquina alternativa tem valor máximo, igual a 1, inclusive porque, com escolha única, não existe razão para favorecer uma atribuição obrigatória. A afinidade entre uma operação e uma máquina não habilitada para ela não é definida, pois tal atribuição não seria permitida. A estratégia de afinidade pode ser usada para direcionar cada operação para alguma máquina que possa efetuar seu processamento gastando menos tempo.

4.2.2 Grau de flexibilidade das operações

Outra característica do JSF a ser explorada é o grau de flexibilidade das operações candidatas a serem alocadas em uma máquina que ficou ociosa, conforme Melo e Ronconi (2013). Considerar o grau de flexibilidade permite favorecer as operações com menos alternativas de processamento, ou seja, com menor grau de flexibilidade. Isso pode ser conveniente pois tais operações possuem possibilidades de alocação mais restritivas e, se não escolhidas, podem dificultar o restante da resolução do problema. A flexibilidade F_j de uma operação j é fornecida por:

$$F_j = \frac{|M_j|}{m} \quad (44)$$

onde $|M_j|$ indica a cardinalidade do conjunto M_j , o que equivale ao número de máquinas habilitadas para a operação j . Assim como a afinidade, a flexibilidade se encontra no intervalo fechado limitado pelos valores 0 e 1. Quanto menor o valor de F_j , maior a possibilidade da operação j ser selecionada.

4.3 REGRAS DE PRIORIDADE PROPOSTAS

A seguir são apresentadas as regras de prioridade desenvolvidas neste trabalho. As mesmas têm alguns elementos de regras da literatura, sendo que uma delas incorpora as estratégias de exploração anteriormente descritas. A associação dessas estratégias às regras conhecidas da literatura será abordada juntamente com os experimentos computacionais no Capítulo 7.

4.3.1 Data de Entrega Modificada com Antevisão

A primeira regra proposta é denominada Data de entrega Modificada com Antevisão (DMA) e utiliza elementos das regras PRTTa e MDD. O intuito da DMA é, com alterações, aliar o bom desempenho da regra MDD na literatura com a antevisão da regra PRTTa. A antevisão permite que, quando uma máquina se torna ociosa, uma operação ainda não disponível, mas com instante de liberação conhecido, seja atribuída a ela. Como exemplo se pode imaginar a situação descrita a seguir. Existe uma operação O_{ij} com instante de liberação $r_j=20$, com tempo de processamento numa dada máquina k sendo $t_{jk}=20$ e que essa é a operação final de um *job* i com data de entrega $d_i=70$. Adicionalmente existe outra operação $O_{i'j'}$ com $r_{j'}=21$, $t_{j'k}=20$ e que é a última operação de um *job* i' com $d_{i'}=50$. No instante $t=20$ a máquina k se torna ociosa e uma operação deve ser selecionada para ocupá-la. Uma regra que

não é capaz de antever as operações a serem liberadas, aqui denominada regra míope, somente consideraria a operação O_{ij} como candidata. Assim, a operação O_{ij} , representada na cor azul, seria selecionada primeiramente, conforme a Figura 8.

Figura 8 - Sequência de uma regra míope.

Fonte: elaborada pelo autor.

Com isso a operação $O_{i'j'}$, na cor verde, seria iniciada somente após o término de O_{ij} . Então, $O_{i'j'}$ seria finalizada no instante 60, causando um atraso de 10 unidades de tempo ao *job* i' . Uma regra com antevisão consideraria também a operação $O_{i'j'}$ como candidata e poderia gerar a sequência da Figura 9.

Figura 9 - Possível sequência de uma regra com antevisão.

Fonte: elaborada pelo autor.

Desse modo, com a inserção de 1 unidade de tempo ocioso na sequência da máquina k , seria possível aguardar a liberação da operação $O_{i'j'}$, escalonando-a antes de O_{ij} . A consequência seria a finalização de ambos os *jobs* i e i' dentro de suas datas de entrega.

Como a antevisão gera tempo ocioso na máquina, ela deve ser utilizada com cautela. Isso é feito através de uma constante α que pondera a penalização da escolha de uma operação ainda não liberada. Além disso, a data de entrega de cada *job* é atualizada quando o mesmo não pode ser entregue no prazo e essa atualização leva em consideração a possibilidade de que as operações candidatas ainda não estejam disponíveis. A regra então pode ser usada para calcular o valor DMA (j,k,t) que determina a prioridade da operação j , no instante t , da maneira que segue:

$$DMA(i, j, t) = \alpha \cdot \max(r_j, t) + \max \left[d_i, \max(r_j, t) + \sum_{j' \in G_i} \min_{k \in M_{j'}} t_{j'k} \right] \quad (45)$$

O primeiro termo da regra, com $\alpha > 0$, busca penalizar as operações que não estão disponíveis no instante t , impedindo que ociosidades excessivas sejam inseridas no *scheduling*. O segundo termo atualiza a data de entrega do *job* i . Para tanto o tempo mínimo de processamento das operações restantes do *job* i é somado ao máximo entre t e r_j . Com isso

a regra DMA é capaz de considerar como candidatas não somente as operações já disponíveis, mas também aquelas cujas predecessoras estejam alocadas, embora não finalizadas, antevendo em que instante no futuro cada operação se tornará disponível. Operações disponíveis em t , aquelas cujas predecessoras já estejam concluídas, têm $r_j \leq t$. Operações que possuem predecessoras ainda não alocadas não entram na lista de operações candidatas. Sempre é selecionada a operação com menor valor $DMA(j,k,t)$.

4.3.2 Data de Entrega Modificada com Antevisão, Afinidade e Flexibilidade

Uma derivação dessa regra proposta associa a DMA à afinidade e à flexibilidade das operações em análise. O objetivo é tentar induzir o *list scheduling algorithm* a atribuir cada operação a uma das máquinas que a processem com maior rapidez, além de favorecer operações com menor número de máquinas alternativas. Essa regra é denominada DMA com Afinidade e Flexibilidade (DMA-AF). A regra proposta incorpora as estratégias relacionadas à afinidade e à flexibilidade do seguinte modo:

$$DMA-AF(i,j,k,t) = \alpha \cdot \max(r_j, t) + \max \left[d_i, \max(r_j, t) + \sum_{j' \in G_i} \min_{k' \in M_{j'}} t_{j'k'} \right] \cdot A_{jk} \cdot F_j \quad (46)$$

Tanto A_{jk} quanto F_j variam de 0 a 1. Deve ser selecionada a operação com menor valor $DMA-AF(i,j,k,t)$. Assim, quanto maior for a afinidade entre j e k , menor será A_{jk} e, portanto, menor resultará o valor $DMA-AF(i,j,k,t)$, favorecendo a alocação da operação j à máquina k . Do mesmo modo, quanto menor for a flexibilidade da operação j , menor será F_j , tornando $DMA-AF(i,j,k,t)$ menor, o que também favorecerá a alocação da operação j à máquina em análise. Da mesma maneira que a regra proposta DMA foi associada às estratégias de afinidade e flexibilidade, outras regras também podem ser, inclusive incorporando a antevisão.

4.4 ESTRUTURA GERAL DAS HEURÍSTICAS CONSTRUTIVAS

Baseada em algumas das regras de prioridade clássicas mencionadas anteriormente, foi implementada uma estrutura geral de heurística construtiva que, como tal, objetiva obter soluções viáveis e de qualidade razoável com rapidez. Essas soluções viáveis geradas pelo algoritmo implementado constituem um futuro ponto de partida para métodos mais sofisticados, como meta-heurísticas, que devem buscar soluções melhores. A heurística construtiva aqui proposta é derivada das apresentadas por Scrich (1997) e Mainieri e Ronconi (2013) e utiliza a abordagem de resolução integrada.

O método tem início no instante igual a zero, no qual todas as máquinas estão disponíveis e todas as operações são conhecidas. À medida que o tempo avança, as operações são atribuídas às máquinas, até que todas as operações estejam alocadas. Uma alocação é definida a cada vez que uma máquina se torna disponível, ou seja, sempre que uma operação é finalizada. Nesse momento são levantadas todas as operações candidatas para ocupar a máquina que se tornou vaga, sendo selecionada a operação de maior prioridade, conforme a regra de prioridade vigente. Neste trabalho, a operação de maior prioridade sempre é aquela que permite à regra gerar o menor valor. Se houver mais de uma máquina disponível no mesmo instante, a primeira máquina a receber uma operação é a que possuir o menor carregamento estimado. Esse carregamento é uma estimativa baseada em toda a carga de trabalho que uma dada máquina k pode vir a receber a partir do instante atual. Ele é dado pela soma dos tempos de processamento de todas as operações ainda não alocadas que podem ser executadas na máquina k .

No caso das regras PRTTa, DMA e DMA-AF, operações ainda não liberadas podem ser alocadas, causando ociosidade na máquina em análise. Caso isso ocorra, a heurística construtiva tenta preencher esse tempo ocioso com operações que possam ser processadas nessa máquina sem interferir no instante de início de processamento da operação não liberada que foi alocada. As operações a serem inseridas no tempo ocioso são aquelas com menor instante de liberação, r_j . O método utiliza as seguintes notações:

t	instante da programação que está sendo considerado pelo algoritmo;
R_k	instante em que a máquina k se torna disponível;
m_j	máquina selecionada para processar a operação j ;
t_{jk}	tempo de processamento da operação j na máquina k ;
r_j	instante de liberação da operação j ;
s_j	instante de início de processamento da operação j ;
u_k	assume 1 se a máquina k , ociosa, não possui operação candidata e 0, caso contrário;
m	número de máquinas;
N	número total de operações do problema;
M_t	conjunto das máquinas disponíveis no instante t ;
$ M_t $	número de máquinas disponíveis no instante t ;
$M_t[w]$	indica a w -ésima máquina de M_t com menor carregamento estimado;
A	conjunto das operações já alocadas a alguma máquina;
$ A $	número de operações já alocadas a alguma máquina;

T^{ocioso}	tempo ocioso gerado pela alocação de uma operação ainda não disponível;
t'	instante no qual tem início um tempo ocioso;
L	número grande.

A heurística construtiva implementada (*Heur_Constr*) é detalhada na sequência. Suas entradas são uma instância do problema (*Inst_JSF*) e uma regra de prioridade (*Regra*).

Algoritmo Heur_Constr (entrada: *Inst_JSF*, *Regra*)

```

1. início;
2.  $t \leftarrow 0$ ;
3.  $A \leftarrow \emptyset$ ;
4. para  $k \leftarrow 1$  até  $m$  faça:
5.    $R_k \leftarrow 0$ ;
6.    $u_k \leftarrow 0$ ;
7. fim para;
8. para  $j \leftarrow 1$  até  $N$  faça:
9.   se (a operação  $j$  não tem predecessora no job) faça:
10.     $r_j \leftarrow 0$ ;
11.  senão;
12.     $r_j \leftarrow \infty$ ;
13.  fim senão;
14. fim para;
15. enquanto ( $|A| \neq N$ ) faça:
16.   $M_t \leftarrow \emptyset$ ;
17.  para  $k \leftarrow 1$  até  $m$  faça:
18.    se ( $(R_k \leq t)$  ou  $(u_k = 1)$ ) faça:
19.       $M_t \leftarrow M_t \cup \{k\}$ ;
20.    fim se;
21.  fim para;
22. Ordene as máquinas de  $M_t$  por carregamento estimado crescente;
23.  para  $j \leftarrow 1$  até  $N$  faça:
24.    se (as predecessoras de  $j$  estiverem concluídas) faça:
25.       $r_j \leftarrow$  instante de término da última predecessora;
26.    fim se;
27.  fim para;
28.  para  $w \leftarrow 1$  até  $|M_t|$  faça:
29.     $k = M_t[w]$ ;
30.    Identifique as operações candidatas para  $k$  em  $t$  segundo Regra;
31.    se (houver operação candidata) faça:
32.      Calcule as prioridades das operações candidatas segundo Regra;
33.      Selecione a operação  $j$  de maior prioridade;
34.       $m_j \leftarrow k$ ;
35.       $s_j \leftarrow \max\{t, r_j\}$ ;
36.       $A \leftarrow A \cup \{j\}$ ;
37.       $R_k \leftarrow s_j + t_{jk}$ ;
38.       $u_k \leftarrow 0$ ;
39.      Retire a operação  $j$  da lista de candidatas;
40.    se ( $r_j > t$ ) faça:
41.       $T^{\text{ocioso}} \leftarrow r_j - t$ ;
42.       $t' \leftarrow t$ ;
43.      enquanto (houver operação candidata  $j'$  com  $t_{j',k} \leq T^{\text{ocioso}}$ ) faça:
44.        Selecione a operação candidata  $j'$  com menor  $r_{j'}$ ;
45.         $m_{j'} \leftarrow k$ ;
46.         $s_{j'} \leftarrow \max\{t', r_{j'}\}$ ;
47.         $A \leftarrow A \cup \{j'\}$ ;
48.         $t' \leftarrow t' + s_j + t_{j',k}$ ;
49.         $T^{\text{ocioso}} \leftarrow r_j - t'$ ;

```

```

50.           Retire a operação  $j'$  da lista de candidatas;
51.       fim enquanto;
52.   fim se;
53.   senão
54.        $u_k \leftarrow 1$ ;
55.        $R_k \leftarrow L$ ;
56.   fim senão;
57. fim para;
58.  $t \leftarrow \min_{k=1 \dots m} \{R_k\}$ ;
59. fim enquanto;
60. Devolva (solução estabelecida por  $m_j, s_j, j=1, \dots, N$ );
61. fim.

```

Essa heurística construtiva descreve os passos seguidos para a geração de soluções para o JSF através de diferentes regras de prioridade a partir de uma instância do problema. As operações candidatas para uma máquina k no instante t são identificadas na linha 30 do algoritmo e podem variar conforme a regra utilizada. Se a regra não possuir antevisão somente as operações liberadas se tornam candidatas. Caso contrário, as operações com instante de liberação conhecido também se tornam candidatas. Havendo operações candidatas, a prioridade de cada uma delas é calculada segundo a regra utilizada, conforme a linha 32. De acordo com a regra, a operação prioritária j pode variar. Desse modo se pode considerar que o algoritmo apresentado é capaz de fornecer diferentes soluções para o problema, representando diversas heurísticas construtivas. Caso ocorra empate na seleção da linha 33, o desempate é feito priorizando a operação que possa liberar a máquina a ser ocupada mais rapidamente. Quando regras com antevisão são utilizadas, as linhas de 40 a 52 preenchem os tempos ociosos gerados pelas alocações de operações ainda não liberadas. Isso ocorre enquanto houver, entre as candidatas, operações que possam ser alocadas sem interferir no início de processamento da operação previamente alocada. Entre essas, é selecionada a operação j' com menor instante de liberação. Em caso de empate na seleção da linha 44, é escolhida a operação com maior prioridade. O algoritmo fornece como saída uma solução para o problema. Uma solução é representada pelas atribuições das operações às máquinas e por seus instantes de início de processamento.

5 BUSCAS LOCAIS

Este capítulo traz as descrições de quatro Buscas Locais (BL) propostas para o *Job Shop* Flexível (JSF). Uma BL é uma heurística de melhoria, conforme Zanakis; Evans e Vazacopoulos (1989), que parte de uma solução inicial e, iterativamente, tenta encontrar soluções melhores dentro de uma região limitada do espaço de soluções factíveis. A solução inicial normalmente é fornecida por um método construtivo. O processo é definido por Aarts e Lenstra (2003) como melhoria iterativa. Frequentemente uma BL é a base de métodos de melhoria mais sofisticados.

Definido um problema de otimização combinatória e instâncias para esse problema, existe um conjunto de soluções factíveis S , eventualmente vazio. Cada solução $s \in S$ possui um subconjunto formado por soluções que estão, de algum modo, próximas a ela. Esse subconjunto de soluções próximas a s é definido através de um mapeamento a partir de s , determinando assim sua vizinhança $N(s)$, sendo que $N(s) \subseteq S$.

Uma forma de efetuar uma BL a partir de uma solução inicial s^0 de um problema de otimização combinatória é verificar, dentre todos os elementos de $N(s^0)$, a solução s^1 que promova a maior melhoria na função-objetivo. Então a solução inicial s^0 é substituída por s^1 e, realizando o mesmo mapeamento, a vizinhança de s^1 é explorada. O processo segue até que se encontre uma solução cuja vizinhança não contenha qualquer solução melhor que ela, recaindo em um ótimo local. A Figura 10 ilustra a exploração sucessiva de vizinhanças.

Figura 10 - Convergência da BL para um ótimo local.

Fonte: elaborada pelo autor.

As vizinhanças são exploradas partindo de s^0 até que um ótimo local, s^4 , seja alcançado. Consolidados tais conceitos, a seguir podem ser apresentadas as quatro propostas de BLs deste trabalho.

5.1 BUSCA LOCAL COM INVERSÃO DE ARCOS

Na BL baseada em Inversão de Arcos (BL-ARC), dada uma solução inicial, são identificados os *jobs* atrasados e também os caminhos críticos desses *jobs*. Para tanto é utilizado um procedimento proposto por Taillard (1994) e presente em Scrich (1997) melhor detalhado na sequência. Então, cada arco disjuntivo do caminho crítico de um *job* atrasado se torna candidato a realizar o movimento de inversão, alterando a sequência das operações na máquina correspondente. Esse movimento se baseia nas propriedades enunciadas por Balas (1969), descritas a seguir:

- i) Inverter um arco disjuntivo de um caminho crítico não gera ciclo no grafo disjuntivo;
- ii) Se a inversão de um arco disjuntivo, não pertencente ao caminho crítico, gerar um grafo acíclico, o maior caminho do grafo obtido não é menor que o maior caminho do grafo original. Isso ocorre pois esse caminho persiste no novo grafo.

Com essas propriedades fica evidente que as investigações para redução do instante de finalização dos *jobs* devem se concentrar nos seus caminhos críticos. Por tal razão a BL-ARC somente avalia as inversões de arcos disjuntivos de caminhos críticos de *jobs* atrasados. Balas (1969) enuncia tais propriedades em um ambiente relacionado à minimização do *makespan*, onde se considera somente o caminho crítico do grafo disjuntivo. Apesar disso as mesmas podem ser usadas quando as inversões são realizadas nos caminhos críticos dos *jobs*. Na literatura são encontrados trabalhos que utilizam essas propriedades ao propor BLs baseadas em inversão, como em Scrich (1997) e Scrich; Armentano e Laguna (2004).

Para cada *job* atrasado são identificadas as soluções obtidas por cada uma das inversões de arcos disjuntivos de seus caminhos críticos. É executada a inversão que leve à maior redução do atraso total, *best move*, alterando a solução atual. Enquanto houver melhoria novas inversões são investigadas. Em suma, na BL-ARC a vizinhança de uma dada solução é composta por todas as soluções geradas pela inversão de cada um dos arcos disjuntivos dos caminhos críticos de todos os *jobs* atrasados.

A execução de um movimento de inversão de arco disjuntivo é exemplificada nas Figuras 11 e 12. A Figura 11 retoma a solução da Figura 4, mas destaca com linhas tracejadas os arcos do caminho crítico do *job* 1, finalizado com atraso e formado pelas operações 1, 2, 3 e 4.

Figura 11 - Caminho crítico do *job* 1 a sofrer inversão de arco.

Fonte: elaborada pelo autor.

A partir da identificação do caminho crítico de um *job* atrasado são avaliadas as possíveis inversões de arcos. Essas inversões se restringem aos arcos disjuntivos, uma vez que os arcos conjuntivos não podem ser alterados. No caminho crítico do *job* destacado os candidatos à inversão são os arcos disjuntivos (2, 5) e (6, 3). Na Figura 12 é exibida a solução obtida pela inversão do arco (6, 3).

Figura 12 - Solução obtida após a inversão de arco.

Fonte: elaborada pelo autor.

Considerando que a operação 6 termina depois da operação 2, se nota que o caminho crítico do *job* 1 no grafo obtido pela inversão foi reduzido, já que agora ele somente envolve arcos conjuntivos. Assim, em nenhuma ocasião o processamento desse *job* precisa ser interrompido para aguardar uma operação de outro *job* ser executada. É importante destacar que, com a inversão do arco (6, 3), os arcos (0, 3) e (6, 11) que antes eram omitidos por serem redundantes passam a ser exibidos. Já os arcos (0, 6) e (3, 11) se tornaram redundantes e, por isso, não estão presentes nessa figura. A solução obtida com tal inversão de arcos é representada pelo gráfico de Gantt da Figura 13.

Figura 13 - Gráfico de Gantt da solução obtida com a inversão de arco.

No gráfico de Gantt da Figura 13, a inversão efetuada fez com que o *job* 1, que na Figura 5 terminava no instante 21, passasse a terminar no instante 11. Com isso seu atraso foi anulado. O *job* 2, antes concluído no instante 18, passou a terminar no instante 20, o que fez seu atraso subir para 9. Já o atraso do *job* 3 sofre redução para 7, pois passou a terminar no instante 15 em vez de 24, como antes. Desse modo o atraso total foi reduzido de 31 para 16.

Nessa busca todas as inversões de arcos de caminhos críticos de *jobs* atrasados são investigadas simultaneamente até que nenhuma melhoria seja mais possível, sendo esse o critério de parada. Nesse momento a BL-ARC estará em um ótimo local. Eventualmente, caso o atraso total da solução se torne nulo durante o processo, a BL é interrompida.

Para a identificação dos caminhos críticos é utilizado o procedimento elaborado por Taillard (1994) descrito na sequência. Nesse procedimento o conjunto Q contém as operações ainda não processadas que não possuem operações precedentes no *job* ou na máquina ainda não concluídas; $PJ[j]$ indica a operação imediatamente antes da operação j no seu *job*; $PM[j]$ indica a operação imediatamente antes de j na sua máquina atual; r^J e r^M são variáveis; $r[1, \dots, N]$ é um vetor que armazena os instantes de liberação das operações; e $Cam_Cri[1, \dots, N]$ é um vetor que identifica o caminho crítico de uma operação indicando qual operação está imediatamente antes dela. Para sua execução é fornecida uma solução (Sol_Atual). O procedimento é apresentado logo a seguir.

Procedimento Ide_Cam_Cri (entrada: Sol_Atual)

```

1. início;
2.  $Q \leftarrow \emptyset$ ;
3. para  $j = 1$  até  $N$  faça:
4.    $r[j] \leftarrow 0$ ;
5.    $Cam\_cri[j] \leftarrow 0$ ;
6.   se (operação  $j$  não tem precedente no job nem na máquina) faça:
7.      $Q \leftarrow Q \cup \{j\}$ ;
8.   fim se;
9. fim para;
10. enquanto ( $Q \neq \emptyset$ ) faça:
11.  Marque um  $j \in Q$ ;
12.   $Q \leftarrow Q \setminus \{j\}$ ;

```

```

13.  $r^J \leftarrow r[PJ[j]] + t[PJ[j]];$ 
14.  $r^M \leftarrow r[PM[j]] + t[PM[j]];$ 
15. se ( $r^J \geq r^M$ ) faça:
16.    $r[j] \leftarrow r^J;$ 
17.    $Cam\_cri[j] \leftarrow PJ[j];$ 
18. senão
19.    $r[j] \leftarrow r^M;$ 
20.    $Cam\_cri[j] \leftarrow PM[j];$ 
21. fim senão;
22. se (( $PM[SJ[j]]$  não existe) ou ( $PM[SJ[[j]]$  está marcado)) faça:
23.    $Q \leftarrow Q \cup \{SJ[j]\};$ 
24. fim se;
25. se (( $PJ[SM[j]]$  não existe) ou ( $PJ[SM[[j]]$  está marcado)) faça:
26.    $Q \leftarrow Q \cup \{SM[j]\};$ 
27. fim se;
28. fim enquanto;
29. Devolva ( $Cam\_Cri, r$ );
30. fim.

```

O conjunto Q possui somente as operações que não precisam aguardar o término de uma operação precedente no *job* ou o término de uma operação precedente na máquina. Para cada operação $j \in Q$ o instante de liberação é calculado e a operação que a faz esperar para iniciar seu processamento é identificada em $Cam_Cri[j]$, sendo que $Cam_Cri[j]=0$ indica que j não tem operação alguma em seu caminho crítico, exceto ela mesma. Então as operações que sucedem j no *job*, $SJ[j]$, e na máquina, $SM[j]$, cujas precedentes não existam ou estejam concluídas são inseridas em Q . O procedimento segue até que as operações finais dos *jobs* que estão nas últimas posições das máquinas tenham seus instantes de liberação e caminhos críticos identificados. Assim, para identificar todo o caminho crítico de um *job* finalizado por uma operação j basta percorrer o vetor iniciando por $j=Cam_Cri[j]$, atualizando $j=Cam_Cri[j]$ até que se tenha $j=0$. Os valores que a variável j assume nesse processo identificam todo seu caminho crítico. Além desse procedimento, a BL-ARC utiliza os procedimentos auxiliares descritos a seguir:

Ava_Sol(*Sol_Atual*): calcula o atraso total da solução *Sol_Atual* e devolve tal valor;

Inverter(*Sol_Entr*,*e*): realiza a inversão do arco disjuntivo *e* em uma solução de entrada *Sol_Entr* e devolve a solução obtida, *Sol_Saída*.

O pseudocódigo da BL-ARC é apresentado a seguir. Além dos elementos anteriormente discriminados, são utilizadas variáveis para armazenar o atraso total da solução atual ($Atr_Tot^{Sol_Atual}$), da solução em processo de melhoria ($Atr_Tot^{Sol_Mel}$) e da solução que sofre inversão ($Atr_Tot^{Sol_Inv}$). Também é usada uma variável para identificar a ocorrência de melhorias (*Melhoria*). Estruturas de dados armazenam a solução inicial (*Sol_Ini*), a solução em processo de melhoria (*Sol_Mel*) e a solução com inversão (*Sol_Inv*). Tais estruturas

armazenam as atribuições das operações às máquinas e seus instantes de início de processamento.

Procedimento BL-ARC (entrada: *Sol_Ini*)

```

1. início;
2. Sol_Atual ← Sol_Ini;
3. Ide_Cam_Cri(Sol_Atual);
4. Atr_TotSol_Atual ← Avaliar(Sol_Atual);
5. Atr_TotSol_Melh ← Atr_TotSol_Atual;
6. Melhoria ← Verdadeiro;
7. enquanto ((Melhoria = Verdadeiro) e (Atr_TotSol_Atual ≠ 0)) faça:
8.     Melhoria ← Falso;
9.     para i ← 1 até n faça:
10.        se (o job i está atrasado) faça:
11.            para cada arco disjuntivo e do caminho crítico do job i faça:
12.                Sol_Inv ← Inverter(Sol_Atual, e);
13.                Atr_TotSol_Inv ← Avaliar(Sol_Inv);
14.                se (Atr_TotSol_Inv < Atr_TotSol_Atual) faça:
15.                    Melhoria ← Verdadeiro;
16.                    Sol_Melh ← Sol_Inv;
17.                    Atr_TotSol_Atual ← Atr_TotSol_Inv;
18.                fim se;
19.            fim para;
20.        fim se;
21.    fim para;
22.    se (Melhoria = Verdadeiro) faça:
23.        Sol_Atual ← Sol_Melh;
24.        Ide_Cam_Cri(Sol_Atual);
25.    fim se;
26. fim enquanto;
27. Devolva (Sol_Atual);
28. fim.
```

Dada uma solução inicial, as inversões dos arcos disjuntivos dos caminhos críticos de todos os *jobs* atrasados são investigadas. A maior melhoria é selecionada e as investigações se repetem sucessivamente até que um ótimo local seja obtido. É preciso evidenciar que a BL-ARC não troca operações de máquina. A ordem pela qual os *jobs* atrasados têm seus caminhos críticos explorados, linha 9 do pseudocódigo, segue a ordem de seus índices, do menor ao maior. Essa definição advém de Scrich (1997) que afirma não ter observado diferença significativa de resultados ao iniciar as inversões pelos *jobs* com maiores atrasos ou mesmo ao realizar escolha aleatória.

A complexidade total da BL-ARC não pode ser estimada, pois o número de iterações do laço iniciado na linha 7 não pode ser previamente determinado. Contudo, a partir do pseudocódigo apresentado, é possível estimar a complexidade da exploração de uma única vizinhança pela BL-ARC. Primeiramente se nota que as operações externas ao laço da linha 7, bem como as operações executadas entre as linhas 22 e 24, têm complexidade $O(N)$. Essa complexidade é inferior à complexidade do laço da linha 9, como será mostrado na sequência. Assim, a análise é feita com base nas operações executadas dentro do laço iniciado na linha 9,

iniciando com as operações mais internas para, depois, chegar às mais externas. As operações das linhas 12 a 18, internas ao laço da linha 11, tem complexidade $O(N)$. Basta notar que a inversão de um arco, no pior caso, pode exigir a determinação dos novos instantes de início e término de todas as operações do problema. As demais operações, linhas 13 a 18, não tem complexidade superior a essa. No pior caso, o laço iniciado na linha 11 pode ser executado $N-1$ vezes e isso, associado à complexidade interna do laço, resulta em complexidade $O(N^2)$. Já o laço da linha 9 é executado n vezes. Assim a exploração de uma vizinhança da BL-ARC tem complexidade $O(n \cdot N^2)$.

5.2 BUSCA LOCAL COM INVERSÃO DE ARCOS E ATUALIZAÇÃO INTERNA

A BL-ARC com Atualizações Internas (BL-ARC-AI) é uma versão da BL-ARC na qual a solução atual é alterada após as inversões dos arcos do caminho crítico de cada *job* atrasado, desde que exista melhoria. Assim, para um dado *job* atrasado são identificadas as soluções obtidas por cada uma das inversões de arco disjuntivo de seu caminho crítico. É executada a inversão que leve à maior redução do atraso total, *best move*, alterando a solução atual. Essa nova solução atual se torna o ponto de partida da investigação das inversões dos arcos do próximo *job* atrasado. Essa atualização intermediária da solução atual diferencia a BL-ARC-AI da BL-ARC, implantando o conceito de subvizinhança. Nesse caso uma subvizinhança é composta por todas as soluções geradas pelas inversões de arcos disjuntivos de um único *job* atrasado, enquanto a vizinhança compreende todas as subvizinhanças. Em suma, na BL-ARC-AI, uma subvizinhança é composta por todas as soluções geradas pela inversão de cada um dos arcos disjuntivos do caminho crítico de um único *job* que esteja atrasado. A atualização da solução atual é realizada internamente do laço que percorre os *jobs*. Os *jobs* atrasados são investigados, um após o outro, até que todos sejam percorridos e nenhuma melhoria seja obtida, recaindo em um ótimo local. É utilizada uma variável adicional (*Melhoria_Interna*) para identificar as melhorias intermediárias. A BL-ARC-AI é apresentada a seguir.

Procedimento BL-ARC-AI (entrada: *Sol_Ini*)

```

1. início;
2. Sol_Atual ← Sol_Ini;
3. Ide_Cam_Cri(Sol_Atual);
4. Atr_TotSol_Atual ← Avaliar(Sol_Atual);
5. Atr_TotSol_Melh ← Atr_TotSol_Atual;
6. Melhoria ← Verdadeiro;
7. enquanto ((Melhoria = Verdadeiro) e (Atr_TotSol_Atual ≠ 0)) faça:
8.   Melhoria ← Falso;
9.   para i ← 1 até n faça:
10.    Melhoria_Interna ← Falso;
```

```

11.   se (o job  $i$  está atrasado) faça:
12.     para cada arco disjuntivo  $e$  do caminho crítico do job  $i$  faça:
13.        $Sol\_Inv \leftarrow Inverter(Sol\_Atual, e)$ ;
14.        $Atr\_Tot^{Sol\_Inv} \leftarrow Avaliar(Sol\_Inv)$ 
15.       se ( $Atr\_Tot^{Sol\_Inv} < Atr\_Tot^{Sol\_Atual}$ ) faça:
16.          $Melhoria \leftarrow Verdadeiro$ ;
17.          $Melhoria\_Interna \leftarrow Verdadeiro$ ;
18.          $Sol\_Melh \leftarrow Sol\_Inv$ ;
19.          $Atr\_Tot^{Sol\_Atual} \leftarrow Atr\_Tot^{Sol\_Inv}$ ;
20.       fim se;
21.     fim para;
22.   fim se;
23.   se ( $Melhoria\_Interna = Verdadeiro$ ) faça:
24.      $Sol\_Atual \leftarrow Sol\_Melh$ ;
25.      $Ide\_Cam\_Cri(Sol\_Atual)$ ;
26.   fim se;
27. fim para;
28. fim enquanto;
29. Devolva ( $Sol\_Atual$ );
30. fim.

```

As inversões dos arcos disjuntivos do caminho crítico de cada *job* atrasado são exploradas sucessivamente, conforme as linhas 9 a 27. A diferença em relação à BL-ARC é que a atualização da solução atual ocorre na linha 24, dentro do laço que percorre os *jobs*. A complexidade da BL-ARC-AI é $O(n \cdot N^2)$, idêntica à complexidade da BL anterior. Isso ocorre porque o acréscimo de operações $O(N)$ dentro do laço da linha 9, fora do laço da linha 12, não altera a complexidade geral, uma vez que o laço da linha 9 já possuía operações internas com complexidade $O(N^2)$.

5.3 BUSCA LOCAL COM INSERÇÃO

A BL baseada em Inserção (BL-INS), partindo de uma solução inicial, identifica os *jobs* atrasados e também os caminhos críticos desses *jobs* com o procedimento de Taillard (1994). Cada operação que esteja no caminho crítico de um *job* atrasado, independentemente a qual *job* pertença, se torna candidata a realizar um movimento. Nesse contexto um movimento constitui na inserção de uma operação candidata em uma posição diferente de sua posição original na sequência de operações. Essa nova posição pode ser na sequência de operações de sua máquina atual, ou em alguma posição de uma de suas máquinas alternativas.

A BL-INS somente avalia as inserções que permitam que a operação candidata inverta sua posição em relação a alguma operação de seu caminho crítico. Isso se deve à propriedade enunciada por Balas (1969), de que inverter um arco não pertencente a caminho crítico não reduz o instante de término do *job*. Porém, em vez de explorar a inversão de um único arco de caminho crítico, a BL-INS tem a proposta de trabalhar com movimentos mais abrangentes, especificamente com a inserção. Tal movimento, no entanto, pode inverter ao

mesmo tempo mais de um arco disjuntivo do caminho crítico, o que foge à garantia de factibilidade de Balas (1969). Assim é utilizado um procedimento proposto por Dell’Amico e Trubian (1993) que permite verificar se uma inserção gerará uma solução factível.

O procedimento de Dell’Amico e Trubian (1993) considera o instante de término de processamento da operação j , $C[j]$, e seu tempo de processamento, $t[j]$. Dada uma operação j ele verifica se ela pode formar a sequência $(O^{PM[j]}, j, O^{SM[j]})$ onde $O^{PM[j]}$ é o conjunto de tarefas que precedem j na mesma máquina e $O^{SM[j]}$ é o conjunto de operações que a sucedem nessa máquina. Para que a sequência $(O^{PM[j]}, j, O^{SM[j]})$ torne a solução infactível é necessário que, para algum $j' \in O^{PM[j]}$, seja verdadeira a inequação $C[SJ[j]] \leq C[PJ[j']] - t[PJ[j']]$ ou que, para algum $j' \in O^{SM[j]}$, seja verdadeira a inequação $C[SJ[j']] \leq C[PJ[j]] - t[PJ[j]]$. No primeiro caso a operação $SJ[j]$ estaria sequenciada antes de uma operação $PJ[j']$ e isso geraria um ciclo por j' estar sequenciada antes de j . Na segunda inequação, uma operação $SJ[j']$ viria antes de uma operação $PJ[j]$ gerando um ciclo, pois j estaria sequenciada antes de j' . O procedimento de Dell’Amico e Trubian (1993) eventualmente pode não garantir a factibilidade para alguns movimentos que geram soluções factíveis. Porém os autores afirmam que tais movimentos, no melhor caso, não promoveriam melhoria e, assim, não seriam escolhidos. Com relação aos movimentos indicados como viáveis, o procedimento fornece plena segurança sobre sua factibilidade.

Para exemplificar um movimento de inserção, a partir da Figura 4 foi criada a Figura 14, na qual está destacado o caminho crítico do *job* 2, atrasado e formado pelas operações 5, 6 e 7.

Figura 14 - Caminho crítico do *job* 2 a ser alterado pelo movimento de inserção.

Fonte: elaborada pelo autor.

O caminho crítico em destaque envolve as operações 1, 2, 5, 6 e 7, além das operações artificiais 0 e 11. Nessa solução a operação 5 deve esperar as operações 1 e 2, pertencentes ao *job* 1, serem processadas para então poder ser executada na máquina 1. A

Figura 15 indica a solução obtida pelo movimento de inserção que colocou a operação 5 antes da operação 1 na sequência da máquina 1.

Figura 15 - Solução obtida com a mudança de posição de uma operação por inserção.

Fonte: elaborada pelo autor.

Nota-se que na solução da Figura 15 o caminho crítico do *job* 2 passou a envolver somente suas próprias operações. Em nenhum momento o processamento desse *job* fica interrompido por ser preciso aguardar o processamento de operações de outros *jobs*. Isso foi possível porque a operação 5 deixou a terceira posição na sequência de operações da máquina 1 para ocupar a primeira posição. Desse modo os arcos disjuntivos (1, 5) e (2, 5) sofreram inversão, implicando que a propriedade de Balas (1969) não pode ser evocada. É preciso esclarecer que na Figura 14 o arco (1, 5) é omitido por ser redundante e o mesmo acontece com o arco (5, 2) na Figura 15. O movimento realizado fez com que os arcos (0, 1) e (5, 8) também possam ser omitidos e que os arcos (0, 5) e (2, 8) precisem ser exibidos. A partir da solução da Figura 15 é exemplificado outro movimento de inserção, na Figura 16, desta vez realizando a troca de máquina da operação 6.

Figura 16 - Solução obtida com a troca de máquina de uma operação.

Fonte: elaborada pelo autor.

No grafo disjuntivo da Figura 16 a operação 6 passa a ser processada na máquina 4 em vez da máquina 3. A troca de máquina é explicitada no grafo pela exclusão dos arcos disjuntivos em cor amarela, da máquina 3, que incidiam no vértice 6 e pela inclusão dos arcos

em azul, da máquina 4, que agora incidem nessa operação. Essa mudança não provoca alteração no caminho crítico do *job 2* porque a operação 6 se torna a primeira a ser processada pela máquina 4. Contudo tal troca pode reduzir o instante de término do *job 2*, caso a nova máquina seja mais rápida que a anterior. Ela também pode provocar alterações nos caminhos críticos de outros *jobs*. A Figura 17 mostra o gráfico de Gantt gerado a partir da solução representada pelo grafo da Figura 16.

Figura 17 - Gráfico de Gantt da solução obtida com as inserções.

Fonte: elaborada pelo autor.

O gráfico de Gantt da Figura 17 mostra como essa solução permite que o *job 1* seja concluído exatamente em sua data de entrega e que o *job 2* tenha seu término antecipado para o instante 15. Já o *job 3* teve seu atraso aumentado em uma unidade de tempo. Contudo os movimentos de inserção permitiram que o atraso total, igual a 31 na solução da Figura 5 e igual a 16 na Figura 13, fosse reduzido para 12.

Na BL-INS, para cada operação candidata, são identificados os possíveis movimentos em cada uma de suas máquinas alternativas e, logicamente, as soluções geradas por tais movimentos. Dentre esses movimentos é realizado aquele que permita obter a solução com menor atraso total, *best move*, alterando a solução atual. Após uma atualização, uma nova vizinhança envolvendo todos os movimentos possíveis de todas as operações candidatas é investigada. Essas vizinhanças seguem sendo investigadas sucessivamente. O critério de parada é a exploração de uma vizinhança sem qualquer melhoria. Além de outros procedimentos já mencionados, a BL-INS utiliza o procedimento auxiliar descrito a seguir:

Movimentar(*Sol_Entr*,*j*,*z*): realiza o movimento *z* da operação *j* em uma solução de entrada (*Sol_Entr*) e devolve a solução obtida (*Sol_Saída*).

O pseudocódigo da BL-INS segue.

Procedimento BL-INS (entrada: *Sol_Ini*)

1. início;
2. *Sol_Atual* $\leftarrow$ *Sol_Ini*;
3. *Ide_Cam_Cri*(*Sol_Atual*);
4. *Atr_Tot*^{*Sol_Atual*} $\leftarrow$ *Avaliar*(*Sol_Atual*);
5. *Atr_Tot*^{*Sol_Melh*} $\leftarrow$ *Atr_Tot*^{*Sol_Atual*};

```

6. Melhoria ← Verdadeiro;
7. enquanto ((Melhoria = Verdadeiro) e (Atr_TotSol_Atual ≠ 0)) faça:
8.   Melhoria ← Falso;
9.   para j ← 1 até N faça:
10.    se (j está no caminho crítico de algum job atrasado) faça:
11.     para cada movimento possível z da operação j faça:
12.      Sol_Mov ← Movimentar(Sol_Atual, j, z);
13.      Atr_TotSol_Mov ← Avaliar(Sol_Mov)
14.      se (Atr_TotSol_Mov < Atr_TotSol_Atual) faça:
15.        Melhoria ← Verdadeiro;
16.        Sol_Melh ← Sol_Mov;
17.        Atr_TotSol_Atual ← Atr_TotSol_Mov;
18.      fim se;
19.    fim para;
20.  fim se;
21. fim para;
22. se (Melhoria = Verdadeiro) faça:
23.   Sol_Atual ← Sol_Melh;
24.   Ide_Cam_Cri(Sol_Atual);
25. fim se;
26. fim enquanto;
27. Devolva (Sol_Atual);
28. fim.

```

Fornecida uma solução inicial, os movimentos de inserção de operações de caminho crítico de *jobs* atrasados são utilizados para gerar as vizinhanças. Essas são exploradas sequencialmente até que o método recaia em um ótimo local, quando tal solução é devolvida. A análise de complexidade da exploração de uma vizinhança pela BL-INS é realizada com base nas operações executadas dentro do laço da linha 9, pois as demais operações têm complexidade inferior. Essa análise se inicia das operações mais internas ao laço para, então, considerar operações mais externas. O movimento de inserção da linha 12, assim como uma inversão da BL-ARC, pode exigir a determinação dos novos instantes de início e término de todas as operações envolvidas, resultando $O(N)$. As demais operações internas ao laço iniciado na linha 11 não tem complexidade superior a essa. O laço da linha 11 pode ser executado até $N-1$ vezes o que, juntamente com a complexidade interna do laço, resulta em complexidade $O(N^2)$. Como o laço da linha 9 é executado N vezes a exploração de uma vizinhança da BL-INS tem complexidade $O(N^3)$.

5.4 BUSCA LOCAL COM INSERÇÃO E ATUALIZAÇÃO INTERNA

A BL-INS com Atualizações Internas (BL-INS-AI) é uma versão da BL-INS na qual a solução atual pode ser atualizada a cada vez que uma parte da vizinhança é explorada. Assim como na BL-INS, a BL-INS-AI identifica os *jobs* atrasados e seus caminhos críticos, tornando candidatas a movimento as operações que estejam em um desses caminhos. O movimento de inserção é o mesmo da BL-INS, inclusive envolvendo trocas de máquina. A diferença é que, para cada operação candidata, são identificados os possíveis movimentos em

suas máquinas. Dentre tais movimentos é realizado aquele que permita obter a solução com menor atraso total, alterando a solução atual. Essa atualização intermediária da BL-INS-AI, assim como a BL-ARC-AI, remete ao uso de subvizinhanças.

Na BL-INS-AI uma subvizinhança é delimitada pelas posições que a operação candidata pode ocupar em suas máquinas alternativas, incluindo a máquina atual. Em cada subvizinhança é selecionada a melhor solução, *best move*, e pode haver, no máximo, uma atualização da solução atual por subvizinhança. Após uma atualização, outras subvizinhanças geradas por movimentos de outras operações são investigadas. Essas subvizinhanças seguem sendo investigadas até que todas as inserções da operação candidata sejam exploradas. A ordem pela qual as operações candidatas são investigadas segue seus índices, começando pela operação inicial do primeiro *job* e terminando com a operação final do último *job*.

Sob certo ponto de vista se pode afirmar que a vizinhança de uma solução envolve todas as subvizinhanças formadas por cada operação candidata. O critério de parada é a exploração de uma vizinhança, equivalente a todas as subvizinhanças de todas as operações, sem qualquer melhoria. O pseudocódigo da BL-INS-AI é apresentado a seguir.

Procedimento BL-INS-AI (entrada: *Sol_Ini*)

```

1. início;
2. Sol_Atual  $\leftarrow$  Sol_Ini;
3. Ide_Cam_Cri(Sol_Atual);
4. Atr_TotSol_Atual  $\leftarrow$  Avaliar(Sol_Atual);
5. Atr_TotSol_Melh  $\leftarrow$  Atr_TotSol_Atual;
6. Melhoria  $\leftarrow$  Verdadeiro;
7. enquanto ((Melhoria = Verdadeiro) e (Atr_TotSol_Atual  $\neq$  0)) faça:
8.   Melhoria  $\leftarrow$  Falso;
9.   para j  $\leftarrow$  1 até N faça:
10.    Melhoria_Interna  $\leftarrow$  Falso;
11.    se (j está no caminho crítico de algum job atrasado) faça:
12.     para cada movimento possível z da operação j faça:
13.      Sol_Mov  $\leftarrow$  Movimentar(Sol_Atual, j, z);
14.      Atr_TotSol_Mov  $\leftarrow$  Avaliar(Sol_Mov);
15.      se (Atr_TotSol_Mov < Atr_TotSol_Atual) faça:
16.       Melhoria  $\leftarrow$  Verdadeiro;
17.       Melhoria_Interna  $\leftarrow$  Verdadeiro;
18.       Sol_Melh  $\leftarrow$  Sol_Mov;
19.       Atr_TotSol_Atual  $\leftarrow$  Atr_TotSol_Mov;
20.     fim se;
21.   fim para;
22. fim se;
23. se (Melhoria_Interna = Verdadeiro) faça:
24.   Sol_Atual  $\leftarrow$  Sol_Melh;
25.   Ide_Cam_Cri(Sol_Atual);
26. fim se;
27. fim para;
28. fim enquanto;
29. Devolva (Sol_Atual);
30. fim.
```

A BL-INS-AI se difere da BL-INS pelo momento em que a solução é atualizada. Na BL-INS-AI essa atualização ocorre na linha 24, dentro do laço que percorre as operações candidatas, linhas 9 a 27. Já na BL-INS a atualização é feita fora deste laço. Assim a amostra considerada para a atualização da BL-INS-AI é menos abrangente que a considerada para as atualizações da BL-INS. A complexidade da BL-INS-AI, assim como a da BL-INS, é $O(N^3)$. Isso ocorre porque as operações internas ao laço iniciado na linha 12 são executadas, no máximo, $N-1$ vezes, igualmente o laço da linha 11 da BL-INS. Basta notar que em ambas as situações, no pior caso, a operação sendo movimentada pode ser inserida antes ou depois de cada outra operação. Além disso, assim como ocorreu com a BL-ARC-AI, a inclusão das operações de atualização dentro do laço da linha 9, fora do laço da linha 12, não altera a complexidade do método.

6 META-HEURÍSTICAS

Neste capítulo são descritas as meta-heurísticas propostas para a minimização do atraso total no ambiente *Job Shop* Flexível (JSF). São propostos métodos que utilizam as Buscas Locais (BL) apresentadas em processos de melhoria que começam com as soluções iniciais fornecidas pelas heurísticas construtivas.

Uma meta-heurística, segundo Osman (2002), é um processo iterativo principal que guia e modifica processos subordinados para eficientemente produzir soluções de alta qualidade. Ela pode combinar diferentes conceitos de inteligência para explorar o espaço de soluções utilizando aprendizagem adaptativa e informações estruturadas.

As meta-heurísticas podem ser classificadas, conforme Osman (2002), em três categorias: meta-heurísticas baseadas em BL, também chamadas de meta-heurísticas de trajetória ou de busca sequencial; meta-heurísticas baseadas em construção de soluções; e meta-heurísticas populacionais. Silver (2004) também afirma que as meta-heurísticas tentam evitar a estagnação prematura em ótimos locais. Blum e Roli (2003) ainda enfatizam o contraste entre as meta-heurísticas de trajetória, as quais se concentram em aprimorar uma única solução, e as meta-heurísticas populacionais, as quais trabalham simultaneamente com um conjunto de soluções. A ideia geral do método ainda deve ser suficientemente generalista para ser aplicada a diversos problemas em diferentes contextos.

As meta-heurísticas são utilizadas para resolver problemas para os quais não são conhecidos algoritmos de resolução eficiente, os problemas NP-Difíceis, e normalmente não garantem a otimalidade de suas soluções. Em muitos casos as meta-heurísticas são inspiradas em fenômenos naturais, como, por exemplo, Algoritmos Genéticos (AG), *Simulated Annealing* (SA) e *Ant Colony Optimization*. Em outras situações não, como é o caso da *Greedy Randomized Adaptive Search Procedure* (GRASP) que incorpora aleatoriedade a uma heurística construtiva e utiliza BL.

Neste trabalho são propostas três meta-heurísticas para a resolução do JSF, cada uma delas pertencente a uma diferente categoria de meta-heurísticas. As três meta-heurísticas propostas utilizam as mesmas ferramentas para a resolução do problema. As diferenças existentes entre elas se devem à maneira pela qual cada uma organiza e utiliza as ferramentas disponibilizadas.

A primeira delas é a *Iterated Local Search* (ILS), um método de trajetória clássico que foi selecionado por sua simplicidade e pelos seus bons resultados para problemas de otimização, como em Laurent e Hao (2009) e Mousakhani (2013). Como método de trajetória, a ILS concentra todos os seus esforços em uma única solução de trabalho, a qual é modificada no decorrer da resolução com o auxílio de elaborados mecanismos de melhoria.

A segunda é a GRASP, uma meta-heurística baseada em construção de soluções que tem bom desempenho com problemas de otimização, conforme Rajkumar *et al.* (2011) e Heath; Bard e Morrice (2013). A estratégia da GRASP é distribuir seus esforços entre uma grande quantidade de pontos de partida distribuídos pela região factível, a partir dos quais sempre são buscadas melhorias com o auxílio de procedimentos menos sofisticados.

A terceira é a *Artificial Bee Colony* (ABC), um recente método populacional que tem superado meta-heurísticas tradicionais em aplicações, como Wang *et al.* (2012) e Pan *et al.* (2011). Como método populacional, a ABC divide seus esforços em uma quantidade limitada de soluções, visando tirar proveito da diversidade da população utilizando mecanismos de melhoria elaborados.

A implementação desses métodos tem como objetivo permitir utilizar diferentes estratégias para a resolução do JSF. Além disso, a utilização de três meta-heurísticas de categorias diferentes para resolver o mesmo problema pode evidenciar os pontos fortes e as fragilidades de cada uma dessas abordagens no contexto considerado, permitindo identificar em quais circunstâncias cada categoria se mostra mais adequada. Nas próximas seções são descritas as três meta-heurísticas desenvolvidas neste trabalho.

6.1 ITERATED LOCAL SEARCH

A meta-heurística ILS é um método de trajetória desenvolvido em torno de uma BL, conforme Lourenço; Martin e Stützle (2010). Ela consegue obter melhores resultados que a BL isolada ao utilizar mecanismos para escapar da estagnação em ótimos locais e, assim, explorar novas áreas da região factível. A ideia do método foi introduzida por Baxter (1981) e se mostrou eficiente, embora simples.

A meta-heurística ILS começa sua busca a partir de uma solução inicial sobre a qual ela realiza uma BL. No entanto a solução fornecida pela execução de uma BL, por definição, é um ótimo local. Assim, para que a ILS não fique estagnada nesse ótimo local e possa explorar novas vizinhanças a solução fornecida pela BL é, de algum modo, modificada. Com isso a BL pode ser novamente executada, indicando um novo ótimo local. A maneira usual da ILS alterar uma solução para sair de um ótimo local é utilizar um procedimento de

perturbação. Essa perturbação pode ser baseada na troca aleatória de alguns elementos da solução. Então a BL e o procedimento de perturbação são executados de forma intercalada iterativamente até que o critério de parada seja alcançado.

Aplicações recentes da ILS a problemas de *scheduling* que alcançaram bons resultados são mencionadas na sequência. Em Pan e Ruiz (2012) é proposto um método híbrido que mescla ILS e *Iterated Greedy* para a minimização do *flowtime* no ambiente *flowshop*. Um problema envolvendo máquinas paralelas é resolvido pela ILS em Della Croce; Garaix e Grosso (2012) com o critério de minimização do atraso total. Em Kampke; Arroyo e Santos (2010) a meta-heurística trabalha com o critério de minimização simultânea do total de recursos alocados e da soma dos instantes de término. O ambiente envolve máquinas paralelas e tempo de *setup* dependente da sequência. Em Essafi; Mati e Dauzère-Pérès (2008) o objetivo da ILS combinada a um AG é minimizar o atraso ponderado no ambiente *Job Shop* (JS). Mousakhani (2013) minimiza o atraso total no JSF com tempos de *setup* dependentes da sequência. A BL realiza movimentos aleatórios e são resolvidas instâncias com até 20 *jobs* e 10 máquinas. Contudo o método é comparado a uma Busca Tabu (BT) desenvolvida para minimizar o *makespan* e uma *Variable Neighborhood Search* (VNS) desenvolvida para minimização multiobjetivo.

Envolvendo outros problemas de otimização, a ILS foi aplicada à minimização de custos do Problema de Roteamento de Veículos com Frota Heterogênea em Subramanian *et al.* (2012); à minimização de custos no Problema de Escalonamento de Veículos com Múltiplos Depósitos em Laurent e Hao (2009); e à minimização do *makespan* em um problema que envolve, simultaneamente, escalonamento de máquinas e de veículos em um ambiente *Flexible Manufacturing Systems* em Deroussi; Gourgand e Tchernev (2008).

Pelo levantamento bibliográfico realizado não foram encontrados trabalhos que apliquem a meta-heurística ILS exatamente ao JSF abordado neste estudo. Por essa razão, aliado ao fato de ser um método simples e eficiente, essa meta-heurística foi selecionada para este trabalho.

Os principais passos das versões mais comuns da meta-heurística ILS para problemas de minimização, conforme Lourenço; Martin e Stützle (2010), estão no fluxograma da Figura 18.

Figura 18 - Fluxograma da meta-heurística ILS.

Fonte: elaborada pelo autor.

No fluxograma da Figura 18 o critério de aceitação pode simplesmente verificar se s'^* é melhor que s . Com isso a solução s sempre seria idêntica à solução incumbente. Segundo Lourenço; Martin e Stützle (2010) existem versões da meta-heurística ILS que utilizam informações históricas. Essas informações servem, por exemplo, para alterar o nível de perturbação de acordo com as soluções obtidas previamente, adaptando alguns parâmetros durante a resolução. Os parâmetros normalmente necessários para a ILS são: o número de iterações, sendo usualmente o critério de parada; e o nível de perturbação.

Durante a execução da ILS as perturbações são fundamentais para que novas áreas da região factível sejam exploradas com o intuito de alcançar o ótimo global ou, ao menos, um bom ótimo local. A Figura 19 ilustra os valores da função-objetivo na região factível de um problema de minimização. Ela apresenta uma situação na qual a perturbação seguida da execução da BL permitiu o sucesso do processo de melhoria da ILS.

Figura 19 - Como a ILS evita a estagnação em ótimos locais.

Fonte: adaptada de Lourenço; Martin e Stützle (2010).

Na Figura 19 uma solução inicial s^0 é obtida por uma heurística construtiva. Então, conforme o fluxograma da Figura 18, uma BL é executada e se obtém a solução s^* . A trajetória da melhoria obtida pela primeira execução da BL que leva a solução incumbente de s^0 para s^* é destacada em azul. Para escapar do ótimo local s^* a ILS efetua uma perturbação sobre tal solução, chegando a uma solução de maior custo s' . Novamente é aplicada a BL, desta vez sobre a solução perturbada s' . Essa execução da BL descreve a trajetória de melhoria destacada em verde, a qual leva a um novo ótimo local que, nesse exemplo, é também o ótimo global.

Segundo Lourenço; Martin e Stützle (2010) a ILS concentra seus esforços em soluções fornecidas por um método de suporte em vez de investigar todo o espaço de soluções candidatas, como outras meta-heurísticas. Com isso, a busca por melhorias se torna relativamente controlável. Isso ocorre porque, com uma perturbação moderada, o foco da busca do método não vai se deslocar para qualquer lugar da região factível. Ao contrário, o foco se mantém em regiões promissoras relativamente próximas às exploradas nas iterações anteriores, mais especificamente da solução incumbente. A Figura 20 ilustra como a ILS explora o espaço de soluções. Nesse exemplo as perturbações sempre são realizadas a partir da solução incumbente.

Figura 20 - Exploração da região factível pela ILS.

Fonte: elaborada pelo autor.

Na Figura 20 a primeira execução da BL sobre a solução inicial chegou ao ótimo local s^* após explorar três vizinhanças. Como primeiro ótimo local encontrado, s^* se tornou solução incumbente. O procedimento de perturbação foi aplicado sobre s^* para escapar do ótimo local e isso gerou a nova solução s' , destacada em vermelho, localizada numa região próxima à vizinhança de s^* . A segunda execução da BL partiu de s' e chegou a outro ótimo local, s'^* . Como esse novo ótimo local não satisfaz o critério de aceitação, a solução incumbente não foi atualizada. Então o procedimento de perturbação foi novamente aplicado sobre s'^* para gerar a solução perturbada s'' , também situada nas redondezas da vizinhança da solução incumbente. A BL descreveu a trajetória que parte de s'' e chega a s''^* após explorar cinco vizinhanças. Como s''^* se mostrou melhor que s^* , ela se tornou a nova solução incumbente, sendo destacada em verde. Apesar do foco da busca ter se deslocado dentro da região factível isso não aconteceu de forma completamente aleatória. Ao contrário, isso ocorreu porque a cada vizinhança uma solução melhor era encontrada, o que mostra como esse deslocamento através do espaço de soluções é orientado. Caso o critério de parada ainda não estivesse satisfeito uma nova perturbação seria aplicada, desta vez sobre s''^* , e o processo seguiria.

6.1.1 Procedimento de perturbação

Para que a ILS consiga se evadir de ótimos locais, este trabalho propõe um procedimento de perturbação que emprega o mesmo movimento de inserção, o mesmo utilizado na BL baseada em Inserção (BL-INS). Para tanto uma operação qualquer é selecionada aleatoriamente e então é inserida em uma posição diferente da original. Essa nova

posição pode ser na sequência da máquina original ou na sequência de uma máquina alternativa. Para isso é utilizado o mesmo procedimento de verificação de factibilidade de Dell'Amico e Trubian (1993). O nível de perturbação é definido por dois parâmetros de entrada. O primeiro deles determina o número de trocas de posição (NTP), que define quantas operações são reinseridas em posições diferentes nas mesmas máquinas de origem. O outro é o número de trocas de máquina (NTM), que define quantas operações são trocadas de máquina.

Quando uma operação é sorteada para ser reinserida em outra posição da mesma máquina, também é definida aleatoriamente a posição na qual ela será colocada. Do mesmo modo, assim que uma operação é sorteada para ser trocada de máquina, também é selecionada aleatoriamente a máquina alternativa para a qual ela será deslocada, bem como a posição que ocupará nessa nova máquina. Uma operação que tem sua posição alterada não pode mais ser sorteada. Portanto, uma mesma operação não sofre troca de posição e de máquina numa mesma execução do procedimento de perturbação. Eventualmente é possível que o procedimento não consiga realizar as quantidades de perturbações estabelecidas, por exemplo, por não haver movimentos factíveis suficientes. Nesse caso o procedimento realiza o número possível de perturbações. Todos os sorteios envolvidos utilizam o gerador de números aleatórios de Taillard (1993). O procedimento mencionado é detalhado na sequência. Além das variáveis anteriormente descritas, são utilizadas variáveis para contar as trocas efetuadas de posição ($Cont_TP$) e de máquina ($Cont_TM$) e para sinalizar uma troca efetuada ($Trocada$). Uma estrutura de dados representa a solução a ser perturbada (Sol_Pert). Conjuntos são empregados para representar potenciais movimentos (Z) e as operações que podem sofrer troca de posição (O^{TP}) ou de máquina (O^{TM}).

Procedimento Perturbar (entrada: Sol_Entr , NTP , NTM)

```

1. início;
2.  $Sol\_Pert \leftarrow Sol\_Entr$ ;
3.  $Cont\_TP \leftarrow 0$ ;
4.  $Cont\_TM \leftarrow 0$ ;
5. Coloque no conjunto  $O$  todas as operações;
6.  $O^{TP} \leftarrow O$ ;
7. enquanto (( $Cont\_TP < NTP$ ) e ( $O^{TP} \neq \emptyset$ )) faça:
8.   Sorteie uma operação  $j \in O^{TP}$ ;
9.    $O^{TP} \leftarrow O^{TP} / \{j\}$ ;
10.   $Trocada \leftarrow \text{Falso}$ ;
11.  Coloque em  $Z$  os movimentos factíveis  $z$  de  $j$  na máquina atual;
12.  enquanto (( $Z \neq \emptyset$ ) e ( $Trocada = \text{Falso}$ )) faça:
13.    Sorteie um movimento  $z \in Z$ ;
14.     $Z \leftarrow Z / \{z\}$ ;
15.    se (o movimento  $z$  gera solução factível) faça:
16.       $Sol\_Pert \leftarrow \text{Movimentar}(Sol\_Pert, j, z)$ ;
17.       $Cont\_TP \leftarrow Cont\_TP + 1$ ;

```

```

18.       $O \leftarrow O/\{j\};$ 
19.       $Trocada \leftarrow \text{Verdadeiro};$ 
20.      fim se;
21.      fim enquanto;
22. fim enquanto;
23.  $O^{TM} \leftarrow O;$ 
24. enquanto  $((Cont\_TM < NTM) \text{ e } (O^{TM} \neq \emptyset))$  faça:
25.   Sorteie uma operação  $j \in O^{TM};$ 
26.    $O^{TM} \leftarrow O^{TM}/\{j\};$ 
27.    $Trocada \leftarrow \text{Falso};$ 
28.   Coloque em  $M_j$  as máquinas alternativas de  $j$  com exceção da atual;
29.   enquanto  $((M_j \neq \emptyset) \text{ e } (Trocada = \text{Falso}))$  faça:
30.    Sorteie uma máquina alternativa  $k \in M_j;$ 
31.     $M_j \leftarrow M_j/\{k\};$ 
32.    Coloque em  $Z$  os potenciais movimentos  $z$  de  $j$  na máquina  $k;$ 
33.    enquanto  $((Z \neq \emptyset) \text{ e } (Trocada = \text{Falso}))$  faça:
34.     Sorteie um movimento  $z \in Z;$ 
35.      $Z \leftarrow Z/\{z\};$ 
36.     se (o movimento  $z$  gera solução factível) faça:
37.       $Sol\_Pert \leftarrow \text{Movimentar}(Sol\_Pert, j, z);$ 
38.       $Cont\_TM \leftarrow Cont\_TM + 1;$ 
39.       $O \leftarrow O/\{j\};$ 
40.       $Trocada \leftarrow \text{Verdadeiro};$ 
41.     fim se;
42.    fim enquanto;
43.   fim enquanto;
44. fim enquanto;
45. Devolva  $(Sol\_Pert);$ 
46. fim.

```

Com o laço iniciado na linha 7, o procedimento efetua trocas aleatórias de posição enquanto o número requerido (NTP) não é alcançado, desde que existam operações habilitadas para tanto. Nesse laço, uma operação ainda não investigada é sorteada na linha 8. Os potenciais movimentos para ela são sorteados até que um movimento factível seja encontrado ou até que se esgotem as possibilidades para a operação sorteada. Na linha 23 as operações candidatas a mudança de máquina se tornam aquelas que não sofreram troca de posição. No laço iniciado na linha 24 operações são trocadas de máquina até a quantidade previamente definida ser alcançada, desde que seja possível. No interior desse laço uma operação ainda não movimentada j é sorteada e uma máquina k , diferente da atual, também é escolhida aleatoriamente. Então o método procura definir aleatoriamente uma posição factível para a operação j na nova máquina k . Caso não seja possível transferir j para k , outras máquinas alternativas de j podem ser investigadas. Não sendo possível movimentar j para qualquer máquina alternativa, outra operação é sorteada. O processo se encerra quando as trocas de máquina requeridas são realizadas ou quando não existem mais operações que possam sofrer tal ação. Nesse momento a solução perturbada é fornecida como saída. O pseudocódigo do procedimento deixa claro que mesmo após as perturbações a solução de entrada, normalmente incumbente, deve ser preservada.

6.1.2 Iterated Local Search proposta

A ILS proposta neste trabalho utiliza uma heurística construtiva, uma das BLs e o procedimento de perturbação anteriormente descritos. Primeiramente é gerada uma solução inicial com a heurística construtiva utilizando uma das regras de prioridade apresentadas. Em seguida é aplicada uma BL, procedimento genericamente chamado neste momento de $BL(Sol_Ent)$. Então tem início um laço que, iterativamente, realiza a perturbação de um ótimo local seguida pela execução da BL. Nesse processo a solução incumbente é preservada, sendo ela o ponto de partida para as perturbações. Assim, a cada iteração é explorada uma nova região relativamente próxima à solução incumbente. Esse processo segue por um número determinado de iterações (NI). O pseudocódigo da ILS proposta vem a seguir. São utilizadas variáveis para armazenar o atraso total da solução incumbente ($Atr_Tot^{Sol_Inc}$) e também para controlar as iterações ($Cont_NI$). Uma estrutura de dados armazena a solução incumbente (Sol_Inc).

Algoritmo ILS (entrada: $Inst_JSF$, $Regra$, NI , NTP , NTM)

```

1. início;
2.  $Sol\_Ini \leftarrow Heur\_Constr(Inst\_JSF, Regra)$ ;
3.  $Sol\_Inc \leftarrow BL(Sol\_Ini)$ ;
4.  $Atr\_Tot^{Sol\_Inc} \leftarrow Avaliar(Sol\_Inc)$ ;
5. para  $Cont\_NI \leftarrow 1$  até  $NI$  faça:
6.    $Sol\_Melh \leftarrow Perturbar(Sol\_Inc, NTP, NTM)$ ;
7.    $Sol\_Melh \leftarrow BL(Sol\_Melh)$ ;
8.    $Atr\_Tot^{Sol\_Melh} \leftarrow Avaliar(Sol\_Melh)$ ;
9.   se ( $Atr\_Tot^{Sol\_Melh} < Atr\_Tot^{Sol\_Inc}$ ) faça:
10.     $Sol\_Inc \leftarrow Sol\_Melh$ ;
11.     $Atr\_Tot^{Sol\_Inc} \leftarrow Atr\_Tot^{Sol\_Melh}$ ;
12.   fim se;
13. fim para;
14. Devolva ( $Sol\_Inc$ );
15. fim.
```

Os parâmetros de entrada que precisam ser ajustados na ILS proposta são o número de iterações e as quantidades de trocas aleatórias do procedimento de perturbação. Também deve ser indicada a regra a ser utilizada pela heurística construtiva, além da instância a ser resolvida. Uma solução inicial é gerada com a heurística construtiva proposta. A primeira execução da BL é realizada sobre essa solução inicial para se obter a primeira solução incumbente. Então, na linha 9, tem início o laço que, por NI iterações, executa o procedimento de perturbação e, sobre a solução perturbada, aplica a BL. Ao final é fornecida a melhor solução obtida durante o processo.

6.2 GRASP

A meta-heurística baseada em construção de soluções GRASP foi proposta por Feo e Resende (1989) para a resolução do Problema de Cobertura de Conjuntos. Desde então tem obtido bons resultados com diversos outros problemas de otimização. O método é um procedimento *multi-start* no qual várias soluções são geradas através da incorporação de aleatoriedade a um processo guloso. A aleatoriedade tem o objetivo de possibilitar uma exploração mais ampla da região factível do que a permitida pelo determinismo de um processo puramente guloso. Contudo, criar soluções completamente aleatórias, apesar de propiciar grande diversidade, conduz a soluções ruins, as quais normalmente não são bons pontos de partida para procedimentos de melhoria. Para evitar soluções excessivamente ruins, mas preservando certa diversidade, os autores propuseram a utilização de uma randomização controlada. Essa randomização, conforme Feo e Resende (1995), faz com que um método construtivo primariamente guloso, a cada passo, incorpore em sua solução em construção não exatamente o componente que traga maior ganho, mas um dos componentes que promovam maiores melhorias. Dessa seleção não completamente gulosa surge o conceito de Lista Restrita de Candidatos (LRC).

Em cada passo de um procedimento construtivo guloso o componente que mais favoreça a função-objetivo é escolhido. Em um procedimento construtivo aleatório, a cada etapa, qualquer componente pode ser sorteado. Já com a utilização da LRC, a escolha aleatória fica restrita a um grupo formado pelos componentes que mais tragam benefício imediato. Segundo Feo e Resende (1995), a LRC pode ter um número fixo de elementos que tragam o maior benefício, sendo denominada restrição por cardinalidade. Outra opção é gerar a LRC com base na diferença entre escolher o melhor ou o pior candidato. Nesse caso é definido um percentual dessa diferença e esse percentual é somado ao benefício obtido com a escolha gulosa, indicando um limite. Então são admitidos na LRC somente os elementos cuja escolha implique em benefício que esteja na faixa que se inicia com a escolha gulosa e se encerra com o valor limite. Esse tipo de delimitação é denominada restrição por valor. Além de definir quantos elementos devem ser admitidos na LRC, também é possível determinar diferentes probabilidades de escolha para cada um deles. Na GRASP básica todos os elementos da LRC têm a mesma probabilidade de sorteio. Contudo diferentes probabilidades podem ser atribuídas para aumentar as chances de sorteio de candidatos que gerem maior benefício.

A meta-heurística GRASP tem sido utilizada com sucesso em diversos trabalhos recentes que envolvem problemas de *scheduling*. Especificamente no ambiente JSF, o trabalho de Rajkumar *et al.* (2011) considera a minimização do *makespan*, carregamento máximo e carregamento total de máquinas. O método define as atribuições das operações às máquinas minimizando o carregamento máximo e o carregamento total. Então os sequenciamentos são determinados de modo a minimizar o *makespan*. Os resultados indicam o melhor desempenho da GRASP em relação a um AG. Rajkumar; Asokan e Vamsikrishna (2010) abordam um JSF que envolve paradas para manutenção preventiva. A resolução é feita como em Rajkumar *et al.* (2011) e os múltiplos objetivos são os mesmos.

A meta-heurística GRASP é utilizada para resolver um *flowshop* permutacional em Arroyo e Pereira (2011). São minimizados dois ou três objetivos, sendo eles *makespan* e atraso máximo ou *makespan*, atraso máximo e *flowtime*. São efetuadas resoluções que combinam os diferentes objetivos através de ponderações e resoluções que objetivam encontrar a fronteira Pareto-ótima. Os resultados são comparados aos de um AG da literatura, indicando um desempenho superior da GRASP. Ronconi e Henriques (2009) aplicam GRASP isoladamente e associada a *path relinking* para minimizar o atraso total no *flowshop* com bloqueio. Rodriguez *et al.* (2012) também mesclam GRASP e *path relinking* para minimizar a soma ponderada dos instantes de término dos *jobs* em um ambiente com máquinas paralelas não idênticas. Damodaran; Vélez-Gallego e Maya (2011) utilizam GRASP para resolver um problema de processamento paralelo de lotes. Primeiramente os *jobs* são divididos em lotes e depois os mesmos são sequenciados com o objetivo de minimizar o *makespan*. Heath; Bard e Morrice (2013) utilizaram a meta-heurística GRASP para resolver uma aplicação de grande porte de um problema de *scheduling* na indústria eletrônica. O problema, denominado *flexible assembly lines*, envolve custos de *setup* e consiste em atribuir famílias de produtos às linhas de produção, além de sequenciar tais linhas minimizando custos. Pelo levantamento realizado neste estudo essa meta-heurística ainda não foi utilizada para minimizar o atraso total no JSF.

O fluxograma da meta-heurística GRASP para um problema de minimização, segundo as etapas propostas por Feo e Resende (1995), é apresentado na Figura 21.

Figura 21 - Fluxograma da meta-heurística GRASP.

Fonte: elaborada pelo autor.

Nesse fluxograma, inicialmente o valor da função-objetivo da solução ótima é infinito, pelo fato da mesma ainda não existir. Então tem início o laço dentro do qual o método se desenvolve. A cada iteração é gerada uma nova solução gulosa s^0 com um grau de aleatoriedade previamente definido. Sobre cada nova solução gerada é aplicada a BL, atingindo um ótimo local s^1 . Caso esse ótimo local, segundo o critério de avaliação, seja melhor que a solução incumbente s^* , a mesma é substituída. O ciclo descrito segue até que o critério de parada seja atingido. Os principais parâmetros a serem definidos na GRASP são: o número de iterações, o qual normalmente determina o critério de parada; e o número de elementos da LRC. A Figura 22 ilustra a atuação da GRASP sobre a região factível.

Figura 22 - Exploração da região factível pela GRASP.

Fonte: elaborada pelo autor.

A cada iteração i é gerada uma nova solução inicial $s^{0,i}$. Cada solução ocupa uma posição diferente na região factível devido à aleatoriedade envolvida na seleção da fase construtiva. As múltiplas soluções geradas e a sua dispersão evidenciam o viés multiconstrutivo desta meta-heurística, a qual é capaz de investigar diversas regiões do espaço de soluções. Sobre cada solução gerada é aplicada a BL e o processo segue até o critério de parada ser alcançado, quando a melhor solução obtida é devolvida.

6.2.1 Geração de soluções com aleatoriedade

Neste trabalho a geração de soluções da meta-heurística GRASP se baseou na heurística construtiva proposta no Capítulo 4. A incorporação de certo grau de aleatoriedade visa proporcionar diversidade dos pontos de partida para as execuções da BL, permitindo que se chegue a melhores resultados.

A heurística construtiva com aleatoriedade utilizada pela GRASP se diferencia da heurística construtiva proposta anteriormente por não selecionar diretamente a operação de maior prioridade. Em vez disso é sorteada uma das operações com maior prioridade, as quais compõem a LRC. Para tanto a linha 33 da *Heur_Constr*, da Seção 4.4 deve ser substituída pelas seguintes instruções destacadas a seguir.

Algoritmo Heur_Constr_Aleat (entrada: *Inst_JSf*, *Regra*)

⋮ *Instruções anteriores de Heur_Constr (linha 1 a linha 32)*

33. Elabore a LRC;

34. Calcule as probabilidades de sorteio das operações da LRC;

35. Sorteie uma operação j da LRC;

⋮ *Instruções posteriores de Heur_Constr (linha 34 a linha 61)*

Desse modo, a heurística construtiva com aleatoriedade, denominada *Heur_Constr_Aleat*, segue as mesmas instruções da *Heur_Constr* até a linha 32. Contudo, na linha 33 da *Heur_Constr_Aleat* é preciso elaborar a LRC. Para isso é utilizado um critério para definir quais operações liberadas, ou com instante de liberação conhecido, devem ser admitidas na LRC. Em seguida deve ser calculada a probabilidade de sorteio de cada operação admitida na LRC. Por fim o sorteio é realizado. Então as instruções a partir da linha 36 da *Heur_Constr_Aleat* são as mesmas que a *Heur_Constr* possui a partir da linha 34. Pelo fato da *Heur_Constr*, apresentada integralmente no Capítulo 4, ter um código extenso cuja maior parte é comum a *Heur_Constr_Aleat*, se optou por destacar os trechos que diferenciam os métodos, evitando repetições. A elaboração da LRC e o cálculo das probabilidades de sorteio de suas operações são abordados juntamente com os experimentos computacionais no Capítulo 7.

6.2.2 GRASP proposta

A partir da descrição da adaptação da heurística construtiva para a GRASP e das BLs apresentadas anteriormente é possível fornecer uma visão geral da GRASP proposta. O critério de parada é um dado número de iterações (*NI*). O pseudocódigo da meta-heurística vem a seguir.

Algoritmo GRASP (entrada: *Inst_JSF*, *Regra*, *NI*)

```

1. início;
2.  $Atr\_Tot^{Sol\_Inc} \leftarrow \infty$ ;
3. para Contador  $\leftarrow 1$  até NI faça:
4.    $Sol\_Ini \leftarrow Heur\_Constr\_Aleat(Inst\_JSF, Regra)$ ;
5.    $Sol\_Melh \leftarrow BL(Sol\_Ini)$ ;
6.    $Atr\_Tot^{Sol\_Melh} \leftarrow Avaliar(Sol\_Melh)$ ;
7.   se ( $Atr\_Tot^{Sol\_Melh} < Atr\_Tot^{Sol\_Inc}$ ) faça:
8.      $Sol\_Inc \leftarrow Sol\_Melh$ ;
9.      $Atr\_Tot^{Sol\_Inc} \leftarrow Atr\_Tot^{Sol\_Melh}$ ;
10.  fim se;
11. fim para;
12. Devolva ( $Sol\_Inc$ );
13. fim.
```

A regra de prioridade a ser utilizada e o parâmetro *NI* são dados de entrada, assim como a instância. A cada uma das *NI* iterações o algoritmo *Heur_Constr_Aleat* fornece uma nova solução, sobre a qual é aplicada uma BL. O algoritmo GRASP fornece como saída a melhor solução obtida durante o processo.

6.3 ARTIFICIAL BEE COLONY

A meta-heurística ABC é um método populacional bioinspirado baseado no comportamento dos indivíduos de uma colônia de abelhas. Numa colônia de abelhas real os indivíduos desempenham diferentes funções no processo de busca por alimento. A ABC se baseia nesses comportamentos para encontrar soluções para problemas de otimização. Essa meta-heurística foi proposta em Karaboga (2005) e em Karaboga e Basturk (2007) para resolver problemas de otimização contínua.

Akay e Karaboga (2009) aplicam a ABC para otimizar funções numéricas que utilizam variáveis inteiras. Para isso os valores das variáveis das soluções são arredondados para o inteiro mais próximo. Recentemente foram apresentadas algumas aplicações da ABC que obtiveram bons resultados em otimização discreta. Szeto; Wu e Ho (2011) utilizam essa meta-heurística para obter soluções de baixo custo para o Problema de Roteamento de Veículos Capacitados. Pal *et al.* (2011) empregam a ABC para resolver um problema relacionado à cadeia de suprimentos que integra aquisição, planejamento e carregamento. Problemas de *scheduling* também são resolvidos por implementações da ABC. Han *et al.* (2012) minimizam o *makespan* em um ambiente *flowshop* com *buffer* zero, superando resultados obtidos com Evolução Diferencial Discreta, AG e BT. Tasgetiren *et al.* (2011) minimizam o *flowtime* em um *flowshop* permutacional. A minimização de adiantamentos e atrasos ponderados é o critério de Pan *et al.* (2011) em um *lot-streaming flowshop*. Wang *et al.* (2012) minimizam o *makespan* no JSF utilizando cruzamentos e mutações, além de BL, numa abordagem hierárquica, mas seus experimentos envolvem somente 15 instâncias.

A meta-heurística ABC foi escolhida como representante da categoria de meta-heurísticas populacionais por ser relativamente recente e, mesmo assim, ter se colocado em evidência na literatura. Por exemplo, o trabalho de Karaboga e Basturk (2008) que compara a ABC a outros métodos de otimização acumula mais de quatro centenas de citações em uma das mais conceituadas bases de artigos científicos. Isso se deve aos bons resultados alcançados pela ABC, inclusive superando meta-heurísticas clássicas, como AG e BT. Além disso, pelo levantamento realizado, ela ainda não foi utilizada para minimizar o atraso total no ambiente JSF.

A meta-heurística ABC se baseia em três comportamentos distintos que os indivíduos de uma colônia real de abelhas desempenham durante o processo de procura por néctar, seu alimento. Cada um desses comportamentos está relacionado a uma função exercida dentro da colônia e um mesmo indivíduo pode exercer funções distintas em

diferentes momentos. É importante ressaltar que na meta-heurística ABC as soluções do problema guardam analogia com as fontes de alimento das colônias reais, enquanto os mecanismos de melhoria utilizados pelo método de resolução seriam equivalentes às abelhas. Assim, neste trabalho, o termo população se refere às soluções, ou fontes de alimento, e não às abelhas. A ABC trabalha em um processo iterativo no qual os indivíduos da colônia exercem suas funções sequencialmente. As três funções incorporadas pela meta-heurística são detalhadas a seguir:

- a) ***Employed bee***: essa função é exercida por metade dos indivíduos da colônia e suas posições iniciais no espaço de busca são definidas com aleatoriedade. Cada *employed bee* é responsável por explorar regiões, ou vizinhanças, próximas a uma solução da população e por transmitir informações a respeito dessas soluções às *onlooker bees*. Na proposta inicial da literatura existe uma *employed bee* para cada fonte de alimento, ou solução.
- b) ***Onlooker bee***: função exercida pela outra metade da colônia e cujo objetivo é intensificar a exploração de regiões promissoras. As regiões candidatas a serem exploradas pelas *onlooker bees* são as circunvizinhanças das soluções obtidas pelas *employed bees*. Os mecanismos de busca de novas soluções das *onlooker bees* são os mesmos das *employed bees*, mas a probabilidade de uma solução ser explorada por uma *onlooker bee* é proporcional à sua qualidade. Como na ABC original da literatura há uma *onlooker bee* para cada solução, pode ocorrer que em uma iteração duas ou mais *onlooker bees* escolham uma mesma solução para explorar. Nesse caso outras soluções podem não ser visitadas por qualquer *onlooker bee*. Assim as funções *employed bee* e *onlooker bee* se diferenciam somente pela seleção das soluções que exploram.
- c) ***Scout bee***: o objetivo dessa função é encontrar novas soluções, ou novas fontes de alimento quando se decide que uma fonte já explorada deve ser abandonada. Nessa ocasião uma *employed bee* descarta sua solução atual e encontra uma nova solução com o auxílio de aleatoriedade. O critério usado para definir se uma solução deve ser abandonada normalmente está relacionado a um determinado número de iterações consecutivas sem que a mesma sofra melhoria.

Na ABC aplicada para otimização contínua, uma solução usualmente é representada por um vetor multidimensional que armazena os valores de variáveis do tipo real. Para encontrar novas soluções a partir de uma solução de partida, as funções *employed bee* e *onlooker bee* primeiramente sorteiam na população uma solução guia. Também são sorteados uma dimensão do vetor e um valor real entre -1 e 1. Então é calculada a diferença entre os valores da solução de partida e da solução guia na dimensão sorteada. Para se chegar à nova

solução, essa diferença é multiplicada pelo valor sorteado entre -1 e 1 e somado ao valor original da respectiva dimensão da solução de partida. Já a função *scout bee* gera soluções aleatórias guiadas somente pelos limitantes inferiores e superiores das variáveis. Por outro lado, as aplicações da ABC a problemas combinatoriais não podem fazer uso desses mesmos artifícios. Por essa razão as versões discretas da ABC normalmente utilizam operadores que permitem chegar a novas soluções investigando as vizinhanças das soluções atuais.

Com as funções envolvidas na ABC definidas é possível apresentar o fluxograma da meta-heurística, presente na Figura 23. Neste fluxograma a vizinhança $N(s^i)$ é utilizada para representar a região próxima a uma solução s^i , independentemente da definição de tal vizinhança. Essa representação visa adequar o método à resolução de problemas inteiros. O parâmetro número de soluções, NS , determina o tamanho da população de trabalho da ABC. A solução incumbente é representada por s^* .

Figura 23 - Fluxograma da meta-heurística ABC.

Fonte: elaborada pelo autor.

Seguindo o fluxograma da Figura 23, o primeiro passo da ABC é gerar uma população inicial, o que pode ser feito aleatoriamente ou com o auxílio de métodos

específicos. Essa população é composta por NS soluções e, na analogia com as colônias reais, cada solução seria uma fonte de alimento. Geradas as soluções iniciais, tem início o processo iterativo de busca de melhorias. Esse processo intercala as funções *employed bee*, *onlooker bee* e *scout bee*, nessa ordem. A cada iteração todas as NS *employed bees* investigam as regiões de suas respectivas soluções. Em toda iteração cada uma das NS *onlooker bees* sorteia uma solução e também investiga sua vizinhança. A probabilidade de uma solução ser sorteada por uma *onlooker bee* é proporcional à sua qualidade. Eventualmente podem ser definidas quantidades de *onlooker bees* diferentes de NS . A cada iteração se verifica quais soluções devem ser abandonadas e substituídas por novas soluções obtidas por *scout bees*. A solução incumbente deve ser preservada durante todo o processo, mesmo que a fonte correspondente seja excluída da população e substituída por outra fornecida pela função *scout bee*. O número de soluções abandonadas por iteração pode variar, uma vez que está relacionado à estagnação das mesmas. Ainda é possível que seja estipulado um número mínimo de abandonos por iteração para que se garanta maior diversificação. Os parâmetros a serem definidos na ABC são: o número de soluções da população (NS); o número de iterações, o que normalmente define o critério de parada; e o número de iterações consecutivas sem melhoria que uma solução deve alcançar antes de ser abandonada. A Figura 24 ilustra como a ABC explora a região factível.

Figura 24 - Exploração da região factível pela ABC.

Fonte: elaborada pelo autor.

Na Figura 24 as soluções obtidas são indicadas por $s^{atualização, solução}$, onde *solução* identifica cada uma das NS soluções da população e *atualização* indica as atualizações que essa solução sofreu durante o processo não exprimindo, necessariamente, as iterações. A

resolução pela ABC começa com a população inicial formada pelas soluções $s^{0,1}$, $s^{0,2}$, $s^{0,3}$ e $s^{0,4}$, as quatro na cor preta. À medida que as iterações ocorrem, soluções melhores são obtidas pela função *employed bee*, destacadas em cinza, e pela função *onlooker bee*, destacadas em azul. Nem sempre tais funções obtêm melhoria. É possível observar que soluções de melhor qualidade, como as originadas em $s^{0,3}$, têm maiores chances de serem exploradas por *onlooker bees* e, por isso, têm mais atualizações na cor azul. Do mesmo modo, soluções de qualidade inferior, como as da cadeia originada em $s^{0,1}$, têm menores probabilidades de serem sorteadas pelas *onlooker bees*, não possuindo atualizações em cor azul. O processo segue até que, após algumas iterações consecutivas sem melhoria, a solução $s^{2,1}$ deve ser abandonada. Nesse momento a função *scout bee* gera uma nova solução a partir da então solução incumbente, $s^{3,3}$. Desse modo é obtida a solução $s^{3,1}$, na cor vermelha, não muito distante da então solução incumbente. As iterações continuam e, a partir dessa nova solução, a ABC chega à solução $s^{7,1}$, destacada em verde, a melhor encontrada pela meta-heurística. Com esse exemplo fica claro que a ABC visa, assim como outras meta-heurísticas populacionais, explorar simultaneamente diversas áreas da região factível. Essa estratégia contrasta com a tática mais focada da ILS e com as construções iterativas da GRASP.

6.3.1 Componentes da meta-heurística

Em otimização contínua a ABC normalmente chega a novas soluções ao multiplicar componentes de soluções atuais por valores aleatórios menores que 1. Assim são conseguidos deslocamentos limitados a uma região relativamente próxima às soluções de partida. Em otimização discreta esse artifício normalmente não pode ser aplicado diretamente e, por isso, é necessário encontrar outras maneiras de se buscar novas soluções em regiões próximas às soluções atuais. Então, a seguir são descritos os componentes utilizados na meta-heurística ABC desenvolvida para o JSF, desde a geração da população inicial até os procedimentos de melhoria.

i) Geração da população inicial

Neste trabalho a geração da população inicial da ABC proposta utiliza o algoritmo construtivo descrito no Capítulo 4 aplicando diferentes estratégias de seleção das operações candidatas. Essa geração visa obter soluções diferentes entre si para que a população possua diversidade. Após a geração e o armazenamento das soluções, tem início o processo iterativo de melhoria. O procedimento utilizado para a geração da população inicial, denominado *Gerar_Pop_Ini*, tem como entrada o número de soluções a serem geradas (NS). Um procedimento, aqui genericamente chamado *Gerar_Sol*, é utilizado para fornecer a r -ésima

solução da população. O procedimento *Gerar_Sol* pode ser o algoritmo *Heur_Constr* utilizando diferentes regras de prioridade ou outra forma de se obter uma solução. Maiores detalhes sobre a geração das soluções são fornecidos no Capítulo 7, referente aos experimentos computacionais. O pseudocódigo do procedimento *Gerar_Pop_Ini* vem a seguir. Uma estrutura em formato de matriz ($Sol_Pop[r]$) é utilizada para armazenar, em cada linha r , a r -ésima solução da população. Dois vetores ($Atr_Tot[r]$ e $Ite_Sem_Mel[r]$) são utilizados, respectivamente, para armazenar os atrasos de cada solução r da população e controlar o número de iterações consecutivas que cada solução r permaneceu sem melhoria.

Procedimento Gerar_Pop_Ini (entrada: NS)

```

1. início;
2.  $Atr\_Tot^{Sol\_Inc} \leftarrow \infty$ ;
3. para  $r \leftarrow 1$  a  $NS$  faça:
4.    $Sol\_Pop[r] \leftarrow Gerar\_Sol(Inst\_JSF, r)$ ;
5.    $Atr\_Tot[r] \leftarrow Avaliar(Sol\_Pop[r])$ ;
6.   se ( $Atr\_Tot[r] < Atr\_Tot^{Sol\_Inc}$ ) faça:
7.      $Atr\_Tot^{Sol\_Inc} \leftarrow Atr\_Tot[r]$ ;
8.      $Sol\_Inc \leftarrow Sol\_Pop[r]$ ;
9.   fim se;
10.   $Ite\_Sem\_Mel[r] \leftarrow 0$ ;
11. fim para;
12. fim.
```

O procedimento *Gerar_Pop_Ini* gera NS soluções com o auxílio do procedimento *Gerar_Sol*. Tais soluções são armazenadas e avaliadas para que sejam utilizadas nas próximas etapas da meta-heurística. A melhor solução obtida nesse processo se torna a solução incumbente.

ii) *Employed bee*

As funções *employed bee* e *onlooker bee* da ABC, originalmente aplicada à otimização contínua, foram adaptadas para o JSF com o uso dos movimentos descritos no Capítulo 5, especificamente o movimento de inserção. Assim, a obtenção de uma nova solução a partir de uma solução da população se dá pela inserção de uma ou mais operações em posições diferentes das atuais. Essa inserção pode ocorrer aleatoriamente, como no procedimento *Perturbar*, ou com determinado favorecimento à melhoria, como na exploração de toda uma vizinhança seguida pela seleção da melhor solução encontrada. Uma vez que a função *employed bee* da literatura emprega aleatoriedade associada a certo direcionamento, a proposta inicial deste trabalho para tal função é utilizar consecutivamente o procedimento de perturbação e a exploração de uma vizinhança. Essa vizinhança é obtida do mesmo modo que ocorre em uma BL, mas, em vez de investigações sucessivas, somente uma única vizinhança é explorada. Desse modo existe um favorecimento moderado às melhorias sem a execução

completa de uma BL, o que distanciaria a proposta deste trabalho da ABC encontrada na literatura. Além disso, essa exploração de vizinhanças pode percorrer caminhos distintos daqueles que a BL baseada em Inserção com Atualizações Internas (BL-INS-AI) percorreria, não estacionando em ótimos locais. Como exemplo, a seguir é apresentado o pseudocódigo da Exploração de Vizinhança única baseada em Inserção e Atualizações Internas (EV-INS-AI), o qual é baseado na BL-INS-AI. O EV-INS-AI também utiliza o procedimento *Ide_Cam_Cri*, o qual identifica o caminho crítico e os instantes de início de cada operação.

Procedimento EV_INS_AI(entrada: *Sol_Entr*)

```

1. início;
2. Sol_Atual ← Sol_Entr;
3. Ide_Cam_Cri(Sol_Atual);
4. Atr_TotSol_Atual ← Avaliar(Sol_Atual);
5. Atr_TotSol_Melh ← Atr_TotSol_Atual;
6. para j ← 1 até N faça:
7.   Melhoria_Interna ← Falso;
8.   se (j está no caminho crítico de algum job atrasado) faça:
9.     para cada máquina alternativa k da operação j (k ∈ Mj) faça:
10.      para cada movimento possível z da operação j em k faça:
11.        Sol_Mov ← Movimentar(Sol_Atual, j, z);
12.        Atr_TotSol_Mov ← Avaliar(Sol_Mov);
13.        se (Atr_TotSol_Mov < Atr_TotSol_Atual) faça:
14.          Melhoria_Interna ← Verdadeiro;
15.          Sol_Melh ← Sol_Mov;
16.          Atr_TotSol_Atual ← Atr_TotSol_Mov;
17.        fim se;
18.      fim para;
19.    fim para;
20.  fim se;
21.  se (Melhoria_Interna = Verdadeiro) faça:
22.    Sol_Atual ← Sol_Melh;
23.    Ide_Cam_Cri(Sol_Atual);
24.  fim se;
25. fim para;
26. Devolva (Sol_Atual);
27. fim.

```

O procedimento EV-INS-AI se difere da BL-INS-AI pela ausência do laço *enquanto*, iniciado na linha 7 da BL-INS-AI, evidenciando que somente uma vizinhança completa, com todas subvizinhanças, é explorada. Definido tal procedimento se pode apresentar o pseudocódigo da função *employed bee*, conforme segue. Os parâmetros de entrada determinam o número de trocas aleatórias de posição (*NTP_EO*) e o número de trocas de máquina (*NTM_EO*) feitas pelo procedimento de perturbação executado pelas funções *employed bee* e *onlooker bee*. Esses parâmetros são as entradas do procedimento *Employed_bee* juntamente com a variável *r*, a qual indica a solução da população a ser explorada.

Procedimento Employed_bee (entrada: r, NTM_{EO}, NTP_{EO})

```

1. início;
2. Sol_Melh ← Perturbar(Sol_Pop[r], NTM_EO, NTP_EO)
3. Sol_Melh ← EV-INS-AI(Sol_Melh);
4. Atr_TotSol_Melh ← Avaliar(Sol_Melh);
5. se (Atr_TotSol_Melh < Atr_Tot[r]) faça:
6.   Sol_Pop[r] ← Sol_Melh;
7.   Atr_Tot[r] ← Atr_TotSol_Melh;
8.   Ite_Sem_Mel[r] ← -1;
9.   se (Atr_Tot[r] < Atr_TotSol_Inc) faça:
10.    Atr_TotSol_Inc ← Atr_Tot[r];
11.    Sol_Inc ← Sol_Pop[r];
12. fim se;
13. fim se;
14. fim.

```

A cada vez que é executada, a função *employed bee* proposta investiga a região próxima a uma solução, sendo que, a cada iteração, a ABC aplica tal função sobre todas as NS soluções da população. A investigação é realizada com a execução de uma perturbação moderada seguida pela exploração de uma única vizinhança completa. Nesse processo, uma dada solução da população é substituída se, a partir dela, for obtida uma solução melhor que ela mesma. Do mesmo modo, quando aplicável, a solução incumbente é atualizada.

iii) *Onlooker bee*

Na fase *onlooker bee* a exploração de uma região próxima a uma dada solução é realizada do mesmo modo que na *employed bee*. Contudo a escolha de cada solução a ser explorada pela função *onlooker bee* utiliza o método *roulette wheel*. Assim, em uma dada iteração, a probabilidade de escolha da r -ésima solução da população é fornecida por:

$$Probabilidade(s^r) = \frac{[Fitness(s^r)]^{-1}}{\sum_{r'=1}^{NS} [Fitness(s^{r'})]^{-1}} \quad (47)$$

onde $Fitness(s^r)$ indica o *fitness*, neste trabalho igual ao atraso total, da r -ésima solução da população, s^r . Como o problema é de minimização, o cálculo das probabilidades utiliza o inverso de *fitness*. Com isso o pseudocódigo da função *onlooker bee* pode ser apresentado na sequência.

Procedimento Onlooker_bee (entrada: NTM_{EO}, NTP_{EO})

```

1. início;
2. Calcule as probabilidades de sorteio baseadas em fitness;
3. Sorteie uma solução  $r'$  da população;
4. Sol_Melh ← Perturbar(Sol_Pop[r'], NTM_EO, NTP_EO)
5. Sol_Melh ← EV-INS-AI(Sol_Melh);
6. Atr_TotSol_Melh ← Avaliar(Sol_Melh);
7. se (Atr_TotSol_Melh < Atr_Tot[r']) faça:
8.   Sol_Pop[r'] ← Sol_Melh;
9.   Atr_Tot[r'] ← Atr_TotSol_Melh;

```

```

10.   $Ite\_Sem\_Mel[r'] \leftarrow -1;$ 
11.  se ( $Atr\_Tot[r'] < Atr\_Tot^{Sol\_Inc}$ ) faça:
12.     $Atr\_Tot^{Sol\_Inc} \leftarrow Atr\_Tot[r'];$ 
13.     $Sol\_Inc \leftarrow Sol\_Pop[r'];$ 
14.  fim se;
15. fim se;
16. fim.

```

O procedimento *Onlooker_bee* se diferencia do procedimento *Employed_bee* por apresentar, nas linhas 2 e 3, o cálculo das probabilidades e o sorteio da solução a ser explorada. A ABC executa essa função NS vezes por iteração e as soluções com menores atrasos têm mais chances de serem exploradas diversas vezes, enquanto soluções ruins tendem a ser ignoradas pela *onlooker bee*. Nesse procedimento a solução sorteada é identificada por r' para se diferenciar da variável r utilizada em outros procedimentos da mesma meta-heurística.

iv) *Scout bee*

Gerar novas soluções que substituam soluções estagnadas é fundamental para a população da ABC garantir certa diversificação e evoluir. Neste trabalho a maneira proposta para se gerar novas soluções com a função *scout bee* é usar o mesmo procedimento de perturbação empregado pela ILS. Sempre que uma solução é considerada estagnada, ela é substituída por outra gerada a partir da perturbação da solução incumbente. Essa perturbação exercida pela *scout bee* deve ser mais intensa que as leves perturbações de *employed bee* e *onlooker bee* e é ajustada pelos parâmetros de entrada que definem as quantidades de trocas aleatórias de posição (NTP_S) e de máquinas (NTM_S). Dessa forma novas soluções são criadas, possibilitando explorar melhor uma região promissora do espaço de soluções. O critério utilizado para determinar se uma solução deve ser abandonada é um número máximo de iterações sem melhoria. O pseudocódigo da função *scout bee* é fornecido logo a seguir.

Procedimento Scout_bee (entrada: r, NTP_S, NTM_S)

```

1.  início;
2.   $Sol\_Pop[r] \leftarrow Perturbar(Sol\_Inc, NTP\_S, NTM\_S);$ 
3.   $Atr\_Tot[r] \leftarrow Avaliar(Sol\_Pop[r]);$ 
4.   $Ite\_Sem\_Mel[r] \leftarrow 0;$ 
5.  se ( $Atr\_Tot[r] < Atr\_Tot^{Sol\_Inc}$ ) faça:
6.     $Atr\_Tot^{Sol\_Inc} \leftarrow Atr\_Tot[r];$ 
7.     $Sol\_Inc \leftarrow Sol\_Pop[r];$ 
8.  fim se;
9.  fim.

```

O procedimento tem como entrada o índice r da solução estagnada, além das quantidades de trocas de posição (NTP_S) e de trocas de máquina (NTM_S). Com sua execução a solução estagnada é substituída por uma nova solução obtida pela perturbação da solução incumbente.

6.3.2 Artificial Bee Colony proposta

Com os componentes anteriormente descritos é possível fornecer uma visão geral da ABC proposta neste trabalho. O critério de parada é um dado número de iterações (NI). O número de soluções (NS) define a quantidade de *employed bees* e a quantidade de *onlooker bees*. O limite de iterações consecutivas sem melhoria (LSM) determina o número máximo de iterações consecutivas que uma solução pode permanecer sem melhoria antes de ser descartada e substituída pela função *scout bee*. O pseudocódigo da ABC implementada vem na sequência.

Algoritmo ABC (entrada: $Inst_JSF, NS, NI, LSM, NTP_EO, NTP_EO, NTP_S, NTP_S$)

```

1. início;
2. Gerar_Pop_Ini(NS);
3. para Contador  $\leftarrow 1$  a  $NI$  faça:
4.   para  $r \leftarrow 1$  a  $NS$  faça:
5.     Employed_bee( $r, NTP\_EO, NTP\_EO$ );
6.   fim para;
7.   para  $r \leftarrow 1$  a  $NS$  faça:
8.     Onlooker_bee( $NTP\_EO, NTP\_EO$ );
9.   fim para;
10.  para  $r \leftarrow 1$  a  $NS$  faça:
11.    se ( $Ite\_Sem\_Mel[r] = -1$ ) faça:
12.       $Ite\_Sem\_Mel[r] \leftarrow 0$ ;
13.    senão
14.       $Ite\_Sem\_Mel[r] \leftarrow Ite\_Sem\_Mel[r] + 1$ ;
15.    fim senão;
16.    se ( $Ite\_Sem\_Mel[r] \geq LSM$ ) faça:
17.      Scout_bee( $r$ );
18.    fim se;
19.  fim para;
20. fim para;
21. Devolva ( $Sol\_Inc$ );
22. fim.

```

Nesse algoritmo os parâmetros necessários são fornecidos juntamente com a instância. Uma população inicial é gerada pela chamada na linha 2 e o processo iterativo tem início com o laço da linha 4. As funções *employed bee* e *onlooker bee* são executadas alternadamente e, quando uma solução se mantém estagnada por LSM iterações, a função *scout bee* é utilizada para gerar uma nova solução. Ao final do processo a melhor solução obtida é devolvida.

7 EXPERIMENTOS COMPUTACIONAIS COM O JSF

Neste capítulo os resultados obtidos com os modelos matemáticos, com as heurísticas construtivas, com as Buscas Locais (BL) e com as meta-heurísticas para o *Job Shop* Flexível (JSF), sem operações de montagem e desmontagem, são apresentados e analisados. Adicionalmente são descritas as instâncias utilizadas nos experimentos. Além da análise dos resultados encontrados por cada meta-heurística, também são feitas análises que visam identificar os pontos fortes de cada uma das três categorias de meta-heurísticas. Por tal razão, houve o cuidado de que as propostas desenvolvidas não descaracterizassem as meta-heurísticas originais da literatura.

Os modelos matemáticos foram implementados em CPLEX no ILOG versão 12.2 de 32 bits, *software* que utiliza simultaneamente todos os núcleos do computador empregado nos experimentos. Os testes foram realizados sem alterações nos parâmetros do *software*, com exceção da memória disponível e do tempo de resolução que foi limitado a 1 hora. Os algoritmos propostos foram implementados em linguagem C sem explorar processamento paralelo, ou seja, utilizando somente 1 núcleo do computador empregado. Todos os experimentos, tanto com o *software* de programação matemática quanto com os algoritmos, foram conduzidos em um computador com um processador Intel Core i7 870, com 8 núcleos e 2,93GHz, com 16GB de RAM e Windows 7 de 64 bits.

7.1 GERAÇÃO DE INSTÂNCIAS

Além da literatura não conter grande volume de trabalhos que abordem a minimização do atraso total no ambiente JSF, também foi constatada certa carência de instâncias para o problema, seja para o critério de minimização do atraso total, que exige datas de entrega, seja para outro critério. Alguns trabalhos, como Brandimarte (1993), Kacem; Hammadi e Borne (2002) e Fattahi; Mehrabad e Jolai (2007), propõem instâncias para o JSF, mas os conjuntos são bastante limitados, tanto com relação ao número de problemas quanto com relação às suas dimensões. Scrich (1997) e Scrich; Armentano e Laguna (2004) geram um grande conjunto com 300 instâncias para o JSF, sendo que maiores detalhes da geração são encontrados no primeiro trabalho. Desse modo, este estudo envolve a geração aleatória de instâncias em moldes semelhantes aos propostos por Scrich (1997) para a realização de experimentos e posterior disponibilização *online*.

Algumas alterações são propostas com relação à geração de instâncias de Scrigh (1997). Em vez de todas as operações terem o mesmo número de máquinas alternativas, se preferiu trabalhar com faixas de flexibilidade. Também são propostas mudanças na maneira de calcular as datas de entrega dos *jobs*. Todos os valores aleatórios utilizados foram fornecidos pelo gerador proposto por Taillard (1993) que permite obter valores inteiros em uma distribuição uniforme $U[a,b]$. Cada instância gerada tem os seguintes componentes:

n	número de <i>jobs</i> ;
m	número de máquinas;
$ O_i $	número de operações do <i>job</i> i ;
M_j	conjunto de máquinas que podem processar a operação j ;
$ M_j $	número de máquinas alternativas para a operação j ;
t_{jk}	tempo de processamento da operação j na máquina k ;
$\bar{t}_j$	tempo médio de processamento da operação j em suas máquinas alternativas;
d_i	data de entrega do <i>job</i> i ;
δ	fator usado no cálculo das datas de entrega dos <i>jobs</i> .

Os valores dos componentes das instâncias geradas são indicados na Tabela 1.

Tabela 1 - Valores dos componentes das instâncias JSF geradas.

Componente	Valores utilizados
n	5, 10, 15, 30, 50, 100
m	5, 10, 15
$ O_i $	$U[\lceil m/2 \rceil, m]$
$ M_j $	$U[\lceil 0,1 \cdot m \rceil, \lceil 0,3 \cdot m \rceil]; U[\lceil 0,3 \cdot m \rceil, \lceil 0,7 \cdot m \rceil]; U[\lceil 0,6 \cdot m \rceil, m]$
M_j	$ M_j $ máquinas sorteadas utilizando $U[1, m]$
t_{jk}	$U[1, 99]$, para uma máquina $k' \in M_j$; e $U[t_{jk}, \min(3 \cdot t_{jk}, 99)]$, para as demais máquinas de M_j
δ	Se $n \geq m$, $\delta = (n/m)^{1/2}$, senão $\delta = 1$
d_i	$0,8 \cdot \delta \cdot \sum_{j=1}^{ O_i } \bar{t}_j$; $1,0 \cdot \delta \cdot \sum_{j=1}^{ O_i } \bar{t}_j$; $1,2 \cdot \delta \cdot \sum_{j=1}^{ O_i } \bar{t}_j$; CA

As quantidades de *jobs*, n , e de máquinas, m , são definidas previamente para cada instância. O número de operações de cada *job* i , $|O_i|$, varia de $\lceil m/2 \rceil$ a m . O tempo de processamento da operação j na sua primeira máquina alternativa k' , $t_{jk'}$, é obtido no intervalo $[1, 99]$. Os tempos de processamento das demais máquinas que podem executar a operação j são limitados a intervalos mais restritos, compreendidos em $[t_{jk'}, \min(3 \cdot t_{jk'}, 99)]$. Com tais cuidados são evitadas grandes discrepâncias de tempos de processamento.

Uma diferença em relação a Scrich (1997) se refere ao número de máquinas alternativas por operação, $|M_j|$. No presente trabalho o valor de $|M_j|$ pode ser obtido em três faixas distintas de flexibilidade. A primeira faixa fica em torno de 20% de flexibilidade, segundo $U[\lceil 0,1 \cdot m \rceil, \lceil 0,3 \cdot m \rceil]$; a segunda faixa de flexibilidade está próxima a 50%, com $U[\lceil 0,3 \cdot m \rceil, \lceil 0,7 \cdot m \rceil]$; e a terceira faixa está perto de 80%, com $U[\lceil 0,6 \cdot m \rceil, \lceil 1,0 \cdot m \rceil]$. Cada instância é gerada utilizando somente uma dessas três faixas de flexibilidade. O intuito do uso dessas faixas é permitir que as instâncias tenham um caráter mais realista, com diferentes números de máquinas habilitadas para cada operação, ao contrário de se determinar um valor fixo para a flexibilidade de todas as operações. Os $|M_j|$ elementos de cada conjunto de máquinas capazes de processar uma operação, M_j , são obtidos aleatoriamente em $U[1, m]$. Em Scrich; Armentano e Laguna (2004) todas as operações de uma dada instância têm o mesmo $|M_j|$.

Outra diferença em relação à Scrich (1997) envolve as datas de entrega que são calculadas a partir da soma dos tempos médios de processamento das operações dos *jobs*. Neste trabalho o tempo médio do *job* é multiplicado por um fator δ definido a partir de n e m , sendo que $\delta = 1$ se $n < m$, e $\delta = (n/m)^{1/2}$, caso contrário. Esse cálculo de δ tem como objetivo dar maiores chances aos métodos de resolução obterem soluções sem grande atraso. Sempre que o número de máquinas equivale ao número de *jobs* ou o excede, é razoável esperar que um fator $\delta = 1$ permita menos atrasos, visto que numa situação hipotética ideal todos os *jobs* poderiam ser alocados simultaneamente. Evidentemente essa situação hipotética desconsidera as relações de precedência e toda complexidade do problema, mas ilustra um dos melhores casos possíveis. Por outro lado, se n excede m , somente m *jobs* podem ser alocados simultaneamente, enquanto $n - m$ *jobs* aguardam a liberação de máquinas. Assim, a equação $\delta = (n/m)^{1/2}$ visa suavizar a rigidez das datas de entrega, possibilitando que uma parte maior dos *jobs* possa ser finalizada sem se afastar demasiadamente do prazo. A elevação da razão n/m à $1/2$ tem o intuito de manter as datas de entrega das instâncias próximas às observadas nas instâncias da literatura, o que não ocorreria se o fator δ fosse obtido pela simples divisão de n por m , cálculo que levaria a datas de entrega muito tolerantes.

Em cada instância o valor de δ é multiplicado por uma dentre as quatro seguintes alternativas: 0,8; 1,0; 1,2 ou uma Combinação Aleatória (CA) desses três valores. Em Scrich (1997) os tempos médios dos *jobs* são multiplicados por $[(n \cdot m)/1000] + 0,5$ ou $[(n \cdot m)/1000] + 1,0$. O gráfico da Figura 25 exibe os valores dos multiplicadores dos tempos médios usados em Scrich (1997) e neste trabalho com diferentes combinações de n e m .

Figura 25 - Multiplicadores usados para as datas de entrega.

Fonte: elaborada pelo autor.

Por esse gráfico se observa que, quando o número de *jobs* cresce e o número de máquinas se mantém, o valor de $(n/m)^{1/2}$ aumenta, concedendo mais prazo para a execução dos *jobs*. Quando o número de *jobs* é fixado e o número de máquinas aumenta, a relação diminui, antecipando as datas de entrega. Esses comportamentos se mostram próximos ao modo pelo qual as datas de entrega são definidas em ambientes reais.

Foram geradas instâncias com as dimensões expressas pela relação $n \times m$: 5x5, 10x5, 10x10, 15x10, 15x15, 30x15, 50x5, 50x10, 50x15 e 100x10. Para cada uma dessas 10 combinações $n \times m$ foi gerado um grupo de 5 instâncias, formando um subconjunto de 50 instâncias. Como ainda existem 3 faixas de flexibilidade e 4 tipos de data de entrega, foram gerados 12 subconjuntos de 50 instâncias, cada qual com uma combinação distinta de faixa de flexibilidade e opção de data de entrega. No total foram geradas 600 instâncias para o JSF com minimização do atraso total.

Essa geração de instâncias, além de oferecer um conjunto numeroso de problemas para testes, também oferece problemas com dimensões bastante superiores que as dimensões da maioria das instâncias disponibilizadas na literatura. Para exemplificar, uma dessas instâncias é apresentada e descrita no Apêndice E.

7.2 FORMULAÇÕES MATEMÁTICAS

Nesta seção são analisados os resultados obtidos através das formulações matemáticas, as quais foram implementadas e executadas em um *software* de programação matemática, o CPLEX. Primeiramente são feitas comparações entre o modelo de Özgüven; Özbakir e Yavuz (2010) e o modelo adaptado, ambos para minimização do *makespan*. Depois são apresentados os resultados do modelo proposto para minimização do atraso total.

7.2.1 Minimização do *makespan*

Foram realizados experimentos concisos envolvendo o critério de minimização do *makespan* para comparar o modelo de Özgüven; Özbakir e Yavuz (2010) com o modelo adaptado e avaliar os potenciais efeitos das alterações propostas. Para manter a coerência com o artigo do modelo de referência foram utilizadas as instâncias criadas por Fattahi; Mehrabad e Jolai (2007). São 20 instâncias para o JSF com dimensões variando de 2 *jobs* e 2 máquinas a 12 *jobs* e 8 máquinas.

Os modelos apresentados utilizam um parâmetro L , mencionado como um número suficientemente grande. Para que os modelos sejam válidos, L deve ser uma delimitação superior do *makespan* ótimo. Este trabalho utiliza como delimitação a soma dos tempos máximos de processamento de todas as operações. Essa soma foi o valor adotado para L nos três modelos testados. No artigo de Özgüven; Özbakir e Yavuz (2010), as restrições do tipo (2) aparecem sem o fator 2 multiplicando o valor L . O acréscimo do fator 2 foi feito nas restrições (3) para permitir o uso do mesmo valor de L para os três modelos. Sem essa multiplicação, a utilização de um valor L menor que duas vezes o *makespan* ótimo pode resultar em um modelo incorreto para o problema, encontrando uma solução pior que a verdadeira solução ótima do problema.

Os resultados obtidos são exibidos na Tabela 2, juntamente com as dimensões das instâncias de Fattahi; Mehrabad e Jolai (2007). Para cada um dos modelos testados e para cada instância, a tabela apresenta o *makespan*, C_{max} , ótimo ou, no caso de uma solução ótima não ter sido obtida dentro do tempo limite, os limitantes inferior e superior obtidos pelo CPLEX entre colchetes. Também é fornecido o *gap* para cada instância, sendo que o mesmo foi calculado segundo a equação:

$$gap = \frac{\text{Limitante superior} - \text{Limitante inferior}}{\text{Limitante inferior}} \cdot 100 \quad (48)$$

Na Tabela 2 os tempos são fornecidos em segundos. A resolução de cada problema foi limitada a 1 hora de processamento. Esses tempos são os tempos de relógio e não os tempos de CPU. A diferença é que o tempo de CPU soma os tempos de processamento dos diferentes núcleos utilizados paralelamente e a execução é interrompida antes do tempo limite ser alcançado em um relógio real. Utilizando a interrupção por tempo de relógio isso não ocorre. Os melhores resultados são destacados em negrito.

Tabela 2 - Resultados da minimização do *makespan*.

Instância	Dimensão (<i>n, m, N</i>)	Modelo de Özgüven; Özbakir e Yavuz (2010)			Modelo Adaptado		
		C_{max}	<i>gap</i> (%)	Tempo (s)	C_{max}	<i>gap</i> (%)	Tempo (s)
SFJS1	2, 2, 4	66	0,00	0,00	66	0,00	0,00
SFJS2	2, 2, 4	107	0,00	0,01	107	0,00	0,00
SFJS3	3, 2, 6	221	0,00	0,00	221	0,00	0,01
SFJS4	3, 2, 6	355	0,00	0,00	355	0,00	0,00
SFJS5	3, 2, 6	119	0,00	0,01	119	0,00	0,01
SFJS6	3, 2, 9	320	0,00	0,01	320	0,00	0,00
SFJS7	3, 5, 9	397	0,00	0,01	397	0,00	0,01
SFJS8	3, 4, 9	253	0,00	0,03	253	0,00	0,02
SFJS9	3, 3, 9	210	0,00	0,01	210	0,00	0,02
SFJS10	4, 5, 12	516	0,00	0,02	516	0,00	0,00
MFJS1	5, 6, 15	468	0,00	0,29	468	0,00	0,32
MFJS2	5, 7, 15	446	0,00	0,60	446	0,00	1,06
MFJS3	6, 7, 18	466	0,00	2,47	466	0,00	11,84
MFJS4	7, 7, 21	554	0,00	24,06	554	0,00	110,97
MFJS5	7, 7, 21	514	0,00	10,66	514	0,00	3,39
MFJS6	8, 7, 24	634	0,00	44,67	634	0,00	38,79
MFJS7	8, 7, 32	[796,00; 879]	10,43	3600,00	879	0,00	1358,97
MFJS8	9, 8, 36	[764,00; 884]	15,71	3600,00	[765,07;884]	15,54	3600,00
MFJS9	11, 8, 44	[808,00; 1108]	37,13	3600,00	[843,03;1086]	28,82	3600,00
MFJS10	12, 8, 48	[957,67;1258]	31,36	3600,00	[944,12;1266]	34,09	3600,00

Conforme a Tabela 2 os modelos foram capazes de encontrar a solução ótima para os problemas de SFJS1 a SFJS10 e de MFJS1 a MFJS6. O modelo adaptado ainda encontrou a solução ótima para a instância MFJS7. Para essas resoluções o *gap* sempre foi igual a zero. Nas demais resoluções, mesmo após 1 hora de processamento computacional, os modelos não conseguiram zerar o *gap*, não indicando uma solução ótima. Nesses casos, cada coluna *makespan* exibe, para cada instância, os limitantes inferior e superior encontrados por cada modelo.

Com a instância MFJS7 o modelo de Özgüven; Özbakir e Yavuz (2010) obteve *gap* igual a 10,43% enquanto o modelo adaptado chegou ao ótimo em 1358,97s. Com a instância MFJS8 os *gaps* foram 15,71% e 15,54%, respectivamente, indicando pequena vantagem do modelo adaptado. Na instância MFJS9 o *gap* do modelo adaptado foi consideravelmente inferior, 28,82% contra 37,13%. O problema MFJS10 foi o único onde o modelo de Özgüven; Özbakir e Yavuz (2010) chegou ao menor *gap*, 31,36% contra 34,09%. No geral esses *gaps* sugerem a superioridade do modelo adaptado.

Outro aspecto relevante a ser observado é o tempo de resolução gasto pelos modelos para encontrar uma solução ótima. Pela Tabela 2 se nota que, nos problemas de menor porte,

SFJS1 a SFJS10, as diferenças não são significativas. Melhores observações podem ser feitas a partir de MFJS1. A média dos tempos de execução com as instâncias até MFJS6 foi de 5,18 segundos no modelo de Özgüven; Özbakir e Yavuz (2010) e de 10,40 segundos no modelo adaptado. Em contrapartida com a instância MFJS7 o modelo adaptado gastou cerca de 2241 segundos a menos que o modelo original.

Em âmbito geral, tais resultados indicam que o modelo adaptado se mostra vantajoso em relação ao modelo de Özgüven; Özbakir e Yavuz (2010). Além de obter menores *gaps* ele é capaz de lidar com uma variedade maior de configurações do JSF, incluindo *jobs* com operações de montagem e desmontagem.

Cabe ainda mencionar que a solução ótima fornecida por Özgüven; Özbakir e Yavuz (2010) para a instância MFJS4 possui *makespan* igual a 564. Porém, a resolução do modelo de Özgüven; Özbakir e Yavuz (2010) em CPLEX realizada neste trabalho obteve uma solução ótima com *makespan* 554 para esse mesmo problema. Então, o *makespan* igual a 564 possivelmente se deve à utilização de um L insuficientemente grande nas restrições do tipo (3) sem a multiplicação pelo fator 2.

Na Figura 26 consta o escalonamento encontrado pelo CPLEX para a instância MFJS4 com *makespan* igual a 554. São 7 máquinas e 7 *jobs*, cada um representado por uma cor. Cada *job* possui 3 operações e entre parênteses está a enumeração sequencial de todas as operações de 1 a 21, como na Tabela 3. Assim, cada operação da figura é identificada por dois valores que indicam o *job* ao qual ela pertence e sua sequência na contagem geral de todas as operações. As setas indicam as precedências das operações nos *jobs*.

Figura 26 - Gráfico de Gantt da solução ótima obtida para a instância MFJS4.

Fonte: elaborada pelo autor.

A Tabela 3 traz a solução relativa à Figura 26.

Tabela 3 - Inícios e terminos das operações da instância MFJS4.

<i>Job</i>	Operação (sequência geral)	Instante de início	Instante de término	Máquina
1	1	0	247	1
1	2	263	393	2
1	3	393	543	4
2	4	145	295	3
2	5	393	459	2
2	6	459	554	6
3	7	78	140	2
3	8	140	285	7
3	9	334	434	5
4	10	0	65	2
4	11	65	238	5
4	12	238	374	6
5	13	140	263	2
5	14	287	334	5
5	15	374	459	6
6	16	0	154	4
6	17	311	434	3
6	18	434	554	5
7	19	0	145	3
7	20	208	376	4
7	21	376	554	7

Com o gráfico da Figura 26 e a Tabela 3 se confirma que a solução ótima de fato possui *makespan* igual a 554.

7.2.2 Minimização do atraso total

Para os experimentos envolvendo o modelo proposto para minimização do atraso total foram utilizadas as 240 menores instâncias geradas neste trabalho. O uso das instâncias geradas se deve ao fato de que elas possuem datas de entrega, informação que não consta nas instâncias de Fattahi; Mehrabad e Jolai (2007). Além disso, os resultados fornecidos pelo CPLEX para essas instâncias devem servir como parâmetro para a avaliação dos métodos heurísticos utilizados neste trabalho. A Tabela 4 sumariza esses resultados representando a média do atraso total por T_{total} . Para cada subconjunto de dimensão $n \times m$ e 60 instâncias são exibidas as médias aritméticas do atraso total e do *gap*, o número total de soluções ótimas obtidas e o tempo médio, em segundos. Nas resoluções em que o *software* de programação matemática não encontrou a solução ótima foi utilizado o valor do limitante superior. O *gap* médio obtido pela resolução exata está na terceira coluna, mas esse cálculo exclui *gaps* infinitos, casos nos quais o limitante inferior foi zero e o superior foi maior que zero.

Subconjuntos com *gaps* infinitos excluídos são identificados pelo símbolo ∞ . Um subconjunto para o qual o CPLEX encontrou todas as soluções ótimas é marcado com *.

Tabela 4 - Resultados obtidos com o modelo que minimiza o atraso total.

Dimensão ($n \times m$)	Instâncias	T_{total}	<i>gap</i> (%)	Soluções ótimas	Tempo (s)
5×5	60	120,3*	0,00	60	1,679
10×5	60	404,6	0,00 ∞	34	1919,324
10×10	60	550,8	603,51 ∞	15	2895,763
15×10	60	1248,7	3866,45 ∞	8	3382,768
Média geral	–	581,1	1117,49	–	2049,884

* Subconjunto no qual todas as soluções ótimas foram encontradas.

A resolução exata encontrou todas as soluções ótimas na dimensão 5×5 gastando, em média, menos de 2 segundos. Na dimensão 10×5 o *software* de programação matemática alcançou 34 soluções ótimas. Coincidentemente, nas demais 26 instâncias desse subconjunto os *gaps* foram desconsiderados por terem os limitantes inferiores iguais a zero. Por tal motivo o valor médio do *gap* resultou zero. O tempo médio de resolução subiu para mais de 30 minutos. Na dimensão 10×10 o CPLEX conseguiu 15 soluções ótimas e em 17 ocasiões o *gap* precisou ser desconsiderado. Mesmo assim o *gap* médio chegou a 603,51% com tempo médio de resolução se aproximando de 50 minutos. Com a dimensão 15×10 a complexidade do JSF dificultou a resolução exata, que conseguiu 8 soluções ótimas. Em 43 ocasiões, *gaps* infinitos foram excluídos do cálculo do *gap* médio que mesmo assim atingiu 3866,45%. O tempo médio se aproximou de 1 hora. Além dos *gaps* aumentarem com o crescimento do número de *jobs* e de máquinas, também se observou que com o aumento da flexibilidade se tem efeito similar. Basta informar que entre as 80 instâncias com 20% de flexibilidade, houve 10 ocasiões em que o CPLEX retornou *gap* infinito. Com 80% de flexibilidade isso aconteceu com 37 problemas.

O CPLEX encontrou comprovadamente a solução ótima em 117 dos 240 problemas. A tabela não exibe os resultados do restante das instâncias porque, a partir da dimensão 15×15, o método exato limitado a 1 hora não foi capaz de obter limitante superior para todas as instâncias de cada subconjunto. A partir da dimensão 30×15 o *software* sequer foi capaz de iniciar a resolução devido a limitações computacionais. Isso evidencia as limitações dos métodos de enumeração implícita. Considerando que na literatura e em ambientes reais o JSF pode envolver um número maior de *jobs* e de máquinas, fica evidente a necessidade de técnicas mais adequadas para resolução rápida desses problemas.

7.3 HEURÍSTICAS CONSTRUTIVAS

Neste estudo foram implementadas todas as regras de prioridade descritas no Capítulo 4, bem como o algoritmo construtivo exposto nesse mesmo capítulo. Dentre as regras, *Shortest Processing Time* (SPT) e *Earliest Due Date* (EDD) não precisam de adaptações para lidar com a flexibilidade de máquinas do JSF. As regras *Minimum Slack Time* (MST), *Smallest Critical Ratio* (SCR), *Modified Due Date* (MDD), *Modified Operation Due date* (MOD) e *Priority Rule for Total Tardiness* adaptada (PRTTa) utilizaram as adaptações descritas na Seção 4.1. Primeiramente é explicado o processo de calibragem das regras com antevisão. Depois são apresentados os resultados das heurísticas construtivas básicas, aquelas que não incorporam as estratégias de afinidade e de flexibilidade. Por fim são fornecidos os resultados das regras que utilizam essas estratégias.

Calibragem do parâmetro α

Inicialmente foram realizados testes preliminares com o intuito de calibrar as regras com antevisão, definindo o valor da constante de ponderação α . Após diversos valores terem sido avaliados, se observou que adotar um valor relacionado à dimensão das instâncias fornecia bons resultados. A partir disso, a constante α passou a ser composta por um valor relacionado à dimensão da instância multiplicado por outro componente denominado α' . O valor selecionado para relacionar α às dimensões das instâncias foi o número de máquinas, m . Além disso, esse valor se mostrou interessante pelo fato de que, independente do nível médio de flexibilidade, quanto mais máquinas o problema envolve, mais máquinas alternativas cada operação possui. Com isso, em instâncias com maior número de máquinas é pouco interessante inserir tempo ocioso, uma vez que as operações têm outras alternativas para seu processamento. Ao contrário, quando o número de máquinas da instância é reduzido, pode ser vantajoso inserir tempo ocioso para que uma operação seja alocada à sua melhor máquina. Desse modo, a constante ficou definida como $\alpha = m \cdot \alpha'$ e o valor de α variou somente pela variação de α' . O gráfico da Figura 27 descreve o valor do atraso total médio das 600 instâncias obtido pela regra Data de entrega Modificada com Antevisão (DMA) à medida que α' variou de 0 a 100 com passo 0,1.

Figura 27 - Calibragem da regra DMA.

Fonte: elaborada pelo autor.

O gráfico permite constatar que, para a DMA, o melhor valor para o parâmetro foi $\alpha = m \cdot 16,8$, com o qual a média do atraso total ficou em 14681,3. Com $\alpha = \alpha' = 0$ o valor médio do atraso total foi 19766,7, deixando claro que a inserção de tempos ociosos pode ser prejudicial se não for feita com moderação. É importante notar que, a partir de um determinado valor de penalização, o atraso total médio oscila pouco, o que torna o processo de calibragem menos crítico. Calibrações semelhantes foram realizadas com as demais regras de prioridade que empregam antevisão. No caso da PRTTa, o melhor valor encontrado foi $\alpha = m \cdot 5,1$.

Regras básicas

Na sequência são apresentados os resultados das heurísticas construtivas básicas, sem inclusão de afinidade e de flexibilidade. Na Tabela 5 cada linha equivale a 60 instâncias. Para cada regra básica e para cada dimensão é exibida a média do atraso total, com a melhor média por dimensão destacada em negrito. Imediatamente abaixo de cada média dessas é fornecido, entre parênteses, o número de vezes em que a respectiva regra obteve o menor atraso total no subconjunto, mesmo que esse menor atraso tenha gerado um empate com uma ou mais regras. Na última linha é fornecido o tempo total de CPU gasto por cada regra para resolver todas as 600 instâncias.

Tabela 5 - Resultados das regras de prioridade básicas por dimensão.

Dimensão ($n \times m$)	T_{total} (Melhores soluções)							
	SPT	MST	SCR	EDD	MDD	MOD	PRTTa	DMA
5×5	190,0 (36)	231,6 (17)	229,1 (15)	208,5 (25)	205,0 (24)	190,0 (37)	190,7 (33)	205,0 (24)
10×5	715,5 (9)	842,5 (6)	722,8 (5)	645,2 (13)	639,7 (11)	708,2 (6)	629,8 (22)	640,4 (10)
10×10	593,4 (28)	662,4 (9)	655,4 (12)	668,3 (10)	663,4 (8)	593,7 (22)	595,9 (26)	669,3 (10)
15×10	1108,9 (8)	1222,3 (14)	1123,6 (8)	1099,4 (3)	1099,5 (8)	1086,0 (10)	966,6 (26)	1118,3 (3)
15×15	1201,1 (24)	1346,3 (17)	1314,9 (13)	1277,3 (17)	1282,7 (18)	1227,3 (16)	1192,5 (25)	1284,9 (12)
30×15	5383,2 (0)	5786,2 (5)	4645,9 (4)	4056,5 (14)	4082,7 (7)	4948,3 (2)	4324,4 (19)	4085,7 (9)
50×5	25883,2 (3)	29293,6 (0)	22506,9 (4)	21915,2 (9)	21526,4 (26)	27471,7 (0)	25230,1 (0)	21468,1 (26)
50×10	22187,1 (0)	25302,1 (0)	17806,6 (2)	16598,1 (15)	16390,9 (17)	22976,4 (0)	20386,8 (1)	16312,5 (25)
50×15	19077,3 (1)	23936,4 (0)	16209,0 (1)	14331,0 (13)	14284,3 (16)	19647,6 (0)	17698,6 (2)	14075,0 (27)
100×10	118898,5 (0)	123230,3 (0)	89425,6 (5)	87544,1 (15)	87643,4 (14)	123985,4 (0)	111557,4 (0)	86953,4 (26)
Média (Total)	19523,8 (109)	21185,4 (68)	15464,0 (69)	14834,3 (134)	14781,8 (149)	20283,5 (93)	18277,3 (154)	14681,3 (172)
Tempo total (s)	3,5	3,6	3,9	3,8	3,8	3,8	4,0	4,5

Com os resultados da Tabela 5 se nota que, em geral, a DMA foi a melhor regra com atraso total médio geral igual a 14681,3. Em seguida veio a MDD, melhor regra conforme Scrich; Armentano e Laguna (2004), com média geral igual a 14781,8. A EDD apareceu em terceiro lugar com atraso médio 14834,3. A DMA também foi a regra que mais vezes obteve a melhor solução, 172 ocasiões, seguida pela PRTTa, com 154 melhores resultados, e pela DMA, com 149. Apesar da grande quantidade de melhores soluções, a PRTTa obteve atraso médio geral elevado. Isso pode ser compreendido ao se notar que, em instâncias com até 30 *jobs*, a PRTTa teve bom desempenho, tanto em termos de atraso total, quanto em termos de melhores resultados. Por outro lado, nas demais instâncias a PRTTa não foi tão bem e ficou consideravelmente aquém das demais regras. Uma justificativa para esse comportamento pode ser a falta de precisão das estimativas das datas de entrega por operação que se torna mais crítica à medida que os problemas crescem. Comportamento similar foi observado com a MOD, regra que também estima datas de entrega para as operações.

Vale ressaltar o desempenho surpreendente da SPT, a única regra implementada que não considera as datas de entrega. Apesar de ter ficado com média geral elevada, essa regra conseguiu 3 menores médias de atraso total, todas entre as menores dimensões. Com relação às maiores instâncias, aquelas com 50 *jobs* ou mais, a DMA apresentou o melhor desempenho, obtendo média de atraso total menor em todas essas dimensões. Ao mesmo tempo conseguiu o maior número de melhores soluções em todas essas dimensões. As heurísticas com antevisão, DMA e PRTTa, conseguiram menores médias em 7 dos 10 subconjuntos, indicando que essa característica pode promover melhores escalonamentos. O tempo computacional demandado pelas heurísticas pode ser considerado satisfatório, pois cada uma delas consumiu menos de 5 segundos para resolver todas as 600 instâncias. Na Tabela 6 esses resultados estão estratificados por tipo de data de entrega.

Tabela 6 - Resultados das regras básicas por tipo de data de entrega.

Multiplicador para data de entrega	T_{total} (Melhores soluções)							
	SPT	MST	SCR	EDD	MDD	MOD	PRTTa	DMA
0,8	25469,1 (32)	32298,6 (5)	21691,3 (11)	20848,1 (41)	20860,4 (35)	26274,9 (25)	25666,5 (27)	20681,7 (45)
1,0	19373,7 (26)	22215,9 (10)	15897,4 (13)	15118,4 (30)	15126,7 (34)	20271,6 (19)	18842,6 (32)	14988,3 (39)
1,2	14363,5 (36)	14376,4 (39)	11024,6 (28)	10362,8 (39)	10282,9 (45)	15350,2 (35)	12648,0 (59)	10184,6 (50)
CA	18889,0 (15)	15850,6 (14)	13242,7 (17)	13008,0 (24)	12857,2 (35)	19237,1 (14)	15952,1 (36)	12870,4 (38)

A DMA conseguiu melhor desempenho nos 3 primeiros tipos de data de entrega, desde o mais apertado, com multiplicador 0,8, até o mais folgado, com 1,2. Somente no último tipo, uma combinação aleatória dos três primeiros, a MDD foi melhor. Quanto ao número de melhores soluções, a DMA superou as demais regras em 3 dos subconjuntos compostos por 150 instâncias cada. Na Tabela 7 os resultados estão separados por nível médio de flexibilidade.

Tabela 7 - Resultados das regras básicas por nível de flexibilidade.

Nível de flexibilidade	T_{total} (Melhores soluções)							
	SPT	MST	SCR	EDD	MDD	MOD	PRTTa	DMA
20	20217,7 (37)	19812,8 (7)	16236,2 (17)	15576,6 (53)	15390,6 (67)	20482,7 (36)	18206,8 (47)	15280,2 (66)
50	18768,3 (38)	20919,5 (28)	14916,5 (28)	14184,6 (35)	14230,0 (38)	19685,4 (22)	17622,0 (47)	14155,2 (47)
80	19585,3 (34)	22823,9 (33)	15239,3 (24)	14741,9 (46)	14724,9 (44)	20682,3 (35)	19002,9 (60)	14608,3 (59)

Nos três níveis médios de flexibilidade a DMA apresentou melhor desempenho. Conforme Baykasoğlu e Ozbakir (2010), existe a tendência de que as distâncias entre os resultados de diferentes regras diminuam com o aumento da flexibilidade, mas isso não fica evidente com os resultados da tabela anterior.

A partir dos resultados expostos foi possível identificar DMA, MDD e EDD como as regras básicas com melhor desempenho. Para verificar a existência de diferença estatisticamente significativa entre essas três regras, se utilizou o teste proposto por Friedman (1937), conhecido como *two-way analysis of variance by ranks*. Tal teste foi conduzido no *software* Minitab e utilizou os valores dos atrasos. O teste foi ajustado para empates e o nível de significância considerado foi igual a 0,05. Considerando DMA e MDD, o teste indicou valor- $p=0,118$. Desse modo não foi possível rejeitar a hipótese nula de que os métodos não diferem entre si. Envolvendo DMA e EDD o teste encontrado valor- $p=0,008$, desta vez permitindo rejeitar a hipótese nula. Com MDD e EDD o teste também não encontrou evidências que indicassem diferença estatisticamente significativa entre as regras, resultando em valor- $p=0,367$.

Regras com afinidade e flexibilidade

As três melhores regras básicas foram selecionadas para compor regras mais elaboradas que utilizam as estratégias de afinidade e de flexibilidade. Primeiramente as regras EDD e MDD foram associadas às estratégias de afinidade e flexibilidade. Em seguida se constatou que a inclusão simultânea da antevisão e das estratégias permitiria melhorias adicionais. Essas melhorias adicionais obtidas por incluir também a antevisão, ainda que modestas, não puderam ser desprezadas. O princípio utilizado nessas novas regras baseadas na EDD e na MDD foi o mesmo da elaboração da DMA com Afinidade e Flexibilidade (DMA-AF) com a diferença de que A_{jk} e F_j multiplicam, respectivamente, os valores fornecidos pelas regras básicas. As novas regras propostas, chamadas EDD com Antevisão, Afinidade e Flexibilidade (EDD-AAF) e MDD com Antevisão, Afinidade e Flexibilidade (MDD-AAF), seguem:

$$EDD-AAF(j,k) = \alpha \cdot \max(r_j, t) + d_i \cdot A_{jk} \cdot F_j \quad (49)$$

$$MDD-AAF(j,k) = \alpha \cdot \max(r_j, t) + d_i' \cdot A_{jk} \cdot F_j \quad (50)$$

Como nas demais regras com antevisão, o primeiro termo da EDD-AAF e da MDD-AAF penaliza a escolha de uma operação ainda não liberada. Na EDD-AAF os valores A_{jk} e F_j multiplicam a data de entrega do *job* i , ao qual a operação j pertence. Na MDD-AAF

os valores relativos à afinidade e à flexibilidade multiplicam a data de entrega modificada do *job i* que é o valor fornecido pela regra básica MDD. Desse modo, a antevisão e as estratégias de exploração puderam ser associadas de forma simples às regras da literatura. Assim como ocorreu com a DMA, as regras propostas EDD-AAF, MDD-AAF e DMA-AF foram calibradas. Na EDD-AAF a constante de penalização foi fixada em $\alpha=m\cdot0,5$, enquanto na MDD-AAF e na DMA-AF foi fixada em $\alpha=m\cdot0,6$. Também foram realizados alguns experimentos com as regras utilizando somente a afinidade ou somente a flexibilidade, mas isoladamente tais estratégias não obtiveram bons resultados. Isso pode indicar que, atuando juntas, afinidade e flexibilidade se contrabalançam. Os resultados das regras propostas que exploram as características do JSF utilizando afinidade e flexibilidade são exibidos na Tabela 8. Adicionalmente a tabela informa a média do número de operações de cada instância.

Tabela 8 - Resultados das regras com estratégias por dimensão.

Dimensão ($n \times m$)	N	T_{total} (Melhores soluções)		
		EDD-AAF	MDD-AAF	DMA-AF
5×5	19,67	187,7 (54)	183,3 (51)	187,6 (48)
10×5	41,27	554,7 (43)	543,6 (33)	546,8 (32)
10×10	73,80	587,2 (43)	585,1 (39)	585,7 (38)
15×10	113,67	866,1 (37)	876,1 (29)	881,4 (27)
15×15	173,93	1130,1 (40)	1101,9 (36)	1103,9 (36)
30×15	349,93	2750,6 (36)	2816,4 (26)	2825,8 (24)
50×5	197,73	17355,3 (32)	17266,9 (21)	17248,7 (22)
50×10	372,40	11242,6 (25)	11168,6 (27)	11142,6 (30)
50×15	576,13	8746,2 (36)	8806,9 (24)	8839,0 (17)
100×10	754,60	64681,6 (43)	65638,3 (14)	65642,4 (11)
Média geral (Total)	–	10810,2 (389)	10898,7 (300)	10900,4 (285)
Tempo total (s)	–	4,2	4,6	4,4

Os resultados da Tabela 8 mostram que as estratégias de exploração permitiram que os valores dos atrasos totais melhorassem substancialmente. No caso da EDD-AAF e da MDD-AAF, a antevisão também teve sua parcela de contribuição, ainda que menor. Tanto

EDD-AAF quanto MDD-AAF conseguiram melhor média de atraso total em 4 dimensões, enquanto a DMA-AF gerou menor média em 2 dimensões. O melhor desempenho foi da EDD-AAF, cuja média geral de atraso total foi 10810,2. Esse resultado guarda certa coerência com Tay e Ho (2008) que afirmam que em seus experimentos com múltiplos objetivos ponderados a EDD foi melhor que a MDD em relação ao critério atraso. Em segundo lugar ficou a MDD-AAF, com média 10898,7, ao passo que a DMA-AF teve média 10900,4. A EDD-AAF também conseguiu indicar maiores quantidades de melhores resultados, obtendo melhor solução em 389 ocasiões. Depois veio a MDD-AAF com 300 melhores soluções e a DMA-AF com 285. Assim como nas regras básicas, o tempo computacional se manteve reduzido com nenhum método ultrapassando 5 segundos de tempo de CPU para resolver todas as instâncias. Isso mostra que a inclusão da antevisão, da afinidade e da flexibilidade, apesar de possibilitar grandes melhorias, não provocou aumento significativo do tempo computacional. Além das comparações permitidas pela tabela anterior, cabe ressaltar que a EDD-AAF conseguiu menor atraso total que a melhor regra da literatura, a MDD, em 81% das instâncias. Na Tabela 9 os resultados das novas regras são organizados por tipo de data de entrega.

Tabela 9 - Resultados das regras com estratégias por tipo de data de entrega.

Multiplicador para data de entrega	T_{total} (Melhores soluções)		
	EDD-AAF	MDD-AAF	DMA-AF
0,8	16452,3 (99)	16648,7 (59)	16656,4 (60)
1,0	11037,6 (103)	11104,1 (68)	11090,0 (64)
1,2	6797,5 (94)	6750,4 (101)	6769,5 (98)
CA	8953,4 (93)	9091,6 (72)	9085,7 (63)

Pela Tabela 9 se constata que, apesar da EDD-AAF ter vencido em 3 dos 4 tipos de data de entrega, a MDD-AAF e a DMA-AF se aproximam dela à medida que a data de entrega ficou mais folgada, tanto com relação ao atraso quanto ao número de melhores soluções. Basta observar que as diferenças de atraso total médio caíram do tipo de data de entrega 0,8 até o tipo 1,2, inclusive com MDD-AAF e DMA-AF superando a EDD-AAF nas datas de entrega mais folgadas. Uma possibilidade para isso é o potencial deterioramento das datas de entrega modificadas, tanto da MDD-AAF quanto da DMA-AF, quando essas estimativas se afastam demais da data de entrega original. Já no tipo CA a EDD-AAF voltou a

ter melhor desempenho. Na Tabela 10 é possível analisar o desempenho dessas regras em cada nível de flexibilidade.

Tabela 10 - Resultados das regras com estratégias por nível de flexibilidade.

Nível de flexibilidade	T_{total} (Melhores soluções)		
	EDD-AAF	MDD-AAF	DMA-AF
20	13485,6	13383,1	13380,2
	(106)	(113)	(107)
50	9310,9	9475,4	9466,4
	(135)	(109)	(101)
80	9634,1	9837,6	9854,6
	(148)	(78)	(77)

Nota-se aqui a tendência de que, à medida que a flexibilidade aumenta, o desempenho da EDD-AAF melhora e se distancia do desempenho das demais. O número de melhores resultados da EDD-AAF saiu de 106, com 20% de flexibilidade, passou para 135, com 50%, e chegou a 148, com 80% de flexibilidade. Enquanto isso as outras regras apresentaram queda no número de melhores soluções. É possível que com mais máquinas alternativas não seja tão vantajoso modificar as datas de entregas dos *jobs*. Em vez disso pode ser melhor manter as datas de entrega originais, insistindo em alocar operações de *jobs* com data de entrega próxima, já que as possibilidades de alocação das operações são mais amplas.

Para evidenciar a contribuição das estratégias de afinidade e de flexibilidade, é preciso mencionar que as versões sem antevisão da EDD-AAF e da MDD-AAF obtiveram, respectivamente, médias de atraso total iguais a 11003,6 e 11106,9. Esses valores são bastante inferiores aos gerados pelas regras EDD e MDD originais e salientam que a contribuição das estratégias de exploração foi mais decisiva que a da antevisão. As diferenças relativas entre as médias dos resultados de cada regra com afinidade e flexibilidade e as médias dos resultados da cada regra básica correspondente podem ser consultadas na Tabela 11. O cálculo foi feito conforme segue:

$$\%Dif = \left[\frac{Médiade T_{total}^{método1} - Médiade T_{total}^{método2}}{Médiade T_{total}^{método2}} \right] \cdot 100 \quad (51)$$

onde *método 2* representa uma regra básica e *método 1* representa sua respectiva regra com afinidade e flexibilidade. Valores negativos indicam que as regras com afinidade e flexibilidade conseguiram melhoria.

Tabela 11 - Melhorias obtidas ao explorar afinidade e flexibilidade.

Dimensão ($n \times m$)	%Dif		
	EDD-AAF	MDD-AAF	DMA-AF
5×5	-9,98	-10,60	-8,50
10×5	-14,02	-15,02	-14,61
10×10	-12,14	-11,81	-12,50
15×10	-21,22	-20,32	-21,18
15×15	-11,53	-14,10	-14,09
30×15	-32,19	-31,02	-30,84
50×5	-20,81	-19,79	-19,65
50×10	-32,27	-31,86	-31,69
50×15	-38,97	-38,35	-37,20
100×10	-26,12	-25,11	-24,51
Média geral	-21,92	-21,80	-21,48

Explorar o conhecimento a respeito da afinidade e da flexibilidade das operações fez com que os resultados da DMA-AF fossem, no geral, 21,48% melhores que os da DMA. No caso da MDD-AAF o atraso médio geral foi reduzido em 21,80%. Enquanto isso, com a EDD-AAF explorar as características do problema possibilitou as maiores melhorias, chegando a 21,92%, no geral. As melhorias foram significativas em todas as dimensões e, no pior caso, alcançaram 8,50%. A melhoria média máxima ocorreu na dimensão 50×15 com a EDD-AAF, 38,97%. Nas três regras com afinidade, flexibilidade e antevisão as maiores melhorias aconteceram nas maiores dimensões. Essa constatação é importante porque a maior dificuldade dos métodos de resolução ocorre exatamente com os maiores problemas. Isso acontece especialmente com os métodos de melhoria que passam a consumir demasiado tempo de processamento quando muitos *jobs* ou máquinas são envolvidos. Na Tabela 12 estão as melhorias alcançadas por tais regras em diferentes níveis médios de flexibilidade. Essas melhorias foram obtidas sobre as médias de cada nível de flexibilidade.

Tabela 12 - Melhorias obtidas pelas estratégias por nível de flexibilidade.

Nível médio de flexibilidade	%Dif		
	EDD-AAF	MDD-AAF	DMA-AF
20	-13,42	-13,04	-12,43
50	-34,36	-33,41	-33,12
80	-34,65	-33,19	-32,54

Claramente se nota que, com flexibilidades mais altas, explorar os conceitos de afinidade e de flexibilidade permitiu que melhorias ainda mais significativas pudessem ser alcançadas. Quanto maior o nível de flexibilidade, mais máquinas alternativas as operações têm, maior é o número de soluções possíveis do problema e mais complexa sua resolução. Então as estratégias propostas tiveram desempenho superior justamente nos problemas mais

complexos. Com isso a exploração da afinidade e da flexibilidade foi capaz de inverter o comportamento observado por Baykasoğlu e Ozbakir (2010). Tais autores afirmam que, em seus experimentos, as diferenças entre os resultados de regras distintas diminuem à medida que a flexibilidade aumenta. Na Tabela 13 as regras com afinidade e flexibilidade e suas regras básicas correspondentes são comparadas através do *Relative Deviation Index* (RDI). O RDI da solução de um método em relação aos demais é dado por:

$$RDI = \frac{f(s) - f(s_{\text{melhor}})}{f(s_{\text{pior}}) - f(s_{\text{melhor}})} \quad (52)$$

onde, numa dada instância, $f(s)$ é o valor do atraso total obtido pelo método avaliado, $f(s_{\text{melhor}})$ é o valor encontrado pelo método com melhor desempenho e $f(s_{\text{pior}})$ é o valor alcançado pelo método de pior desempenho. Nos casos em que $f(s_{\text{melhor}})$ e $f(s_{\text{pior}})$ foram iguais, se adotou RDI igual a zero.

Tabela 13 - RDIs médios das principais regras implementadas.

Dimensão ($n \times m$)	EDD	MDD	DMA	EDD-AAF	MDD-AAF	DMA-AF
5×5	0,532	0,506	0,506	0,147	0,186	0,242
10×5	0,697	0,758	0,776	0,184	0,248	0,260
10×10	0,623	0,639	0,613	0,189	0,213	0,215
15×10	0,696	0,660	0,708	0,240	0,275	0,291
15×15	0,579	0,598	0,598	0,241	0,222	0,227
30×15	0,873	0,880	0,878	0,038	0,072	0,075
50×5	0,932	0,693	0,667	0,260	0,088	0,089
50×10	0,943	0,899	0,891	0,059	0,035	0,027
50×15	0,954	0,943	0,906	0,024	0,031	0,037
100×10	0,950	0,952	0,916	0,018	0,054	0,053
Média geral	0,778	0,753	0,746	0,140	0,142	0,152

Os valores da Tabela 13 confirmam a superioridade dos métodos que utilizam as estratégias de afinidade, flexibilidade e que incorporam antevisão. EDD-AAF, MDD-AAF e DMA-AF obtiveram RDI médios inferiores a 0,160 enquanto EDD, MDD e DMA tiveram valores superiores a 0,740. Em 7 das 10 dimensões a EDD-AAF gerou o menor RDI médio, confirmando sua superioridade de forma ainda mais incisiva.

Comparações adicionais entre as regras com e sem as estratégias de exploração podem ser realizadas utilizando uma técnica proposta por Dolan e Moré (2002). A técnica denominada perfis de desempenho usa uma representação compacta para comparação rápida entre diferentes métodos de resolução. Nessa técnica, o valor obtido por um método a no problema p é comparado ao melhor resultado para esse mesmo problema dentre os obtidos por todos os métodos em análise. É utilizada a relação de desempenho:

$$\sigma(p,a) = \frac{f(p,a)}{\min\{f(p,a'): a' \in U\}} \quad (53)$$

onde $f(p,a)$ é o valor da função-objetivo alcançado pelo método a no problema p , e U é o conjunto de métodos comparados. Assim pode ser calculada a probabilidade $Probabilidade(a,\tau)$ de que a relação de desempenho $\sigma(p,a)$ do método a esteja limitada a um fator $\tau \in \Re$, como segue:

$$Probabilidade(a,\tau) = \frac{|\{p \in P : \sigma(p,a) \leq \tau\}|}{|P|} \quad (54)$$

onde $|P|$ indica o número de elementos do conjunto de problemas P . Assim, $Probabilidade(a,\tau): \Re \rightarrow [0,1]$ é uma função de distribuição acumulada que expressa o desempenho relativo de a frente aos demais métodos na resolução dos problemas do conjunto P . Então, dado um valor τ , $Probabilidade(a,\tau)$ permite identificar a porcentagem de problemas de P em que o método a consegue obter uma solução até τ vezes pior que a melhor solução alcançada. A curva inicia em $\tau=1$ onde o valor de $Probabilidade(a,\tau)$ indica a porcentagem de melhores soluções que o método s conseguiu. Quando para um determinado τ' se tem $Probabilidade(a,\tau')=1$, significa que em 100% dos problemas o método a obteve resultados, no máximo, τ' vezes piores que os melhores resultados de cada problema. Visando evitar a divisão por zero, neste trabalho é adicionada uma unidade ao valor da função-objetivo obtido por cada método em cada instância. A Figura 28 apresenta perfis de desempenho que permitem avaliar o uso das estratégias de afinidade e de flexibilidade. Para permitir melhor visualização o valor de τ foi limitado a 5.

Figura 28 - Perfis de desempenho das melhores regras de prioridade.

Fonte: elaborada pelo autor.

Pelos perfis da Figura 28 é possível observar a dominância das regras que utilizam as estratégias de afinidade e de flexibilidade sobre as demais. Considerando apenas as seis regras, a EDD-AAF obtém melhor solução em mais de 60% das instâncias, como indica o extremo esquerdo de sua curva. Juntamente com as regras MDD-AAF e DMA-AF, a EDD-AAF exerce dominância sobre as regras básicas até a extremidade direita do gráfico. Os perfis das regras MDD e DMA quase se sobrepõem, indicando que elas têm comportamento muito similar. Já a curva da EDD inicia ligeiramente abaixo das demais, mas evoluiu acompanhando as outras regras básicas. De fato o teste de Friedman indica haver diferenças significativas entre as regras sem e com afinidade e flexibilidade. Envolvendo DMA e DMA-AF, situação na qual a afinidade e a flexibilidade geraram os menores ganhos, o teste resulta em valor- $p=0,000$. Considerando somente as regras que empregam as estratégias propostas, entre MDD-AAF e DMA-AF o teste resulta em valor- $p=0,128$, um resultado bastante coerente visto que as regras são bastante semelhantes. Já a EDD-AAF parece se diferenciar das demais. Tanto na comparação com a MDD-AAF quanto na comparação com a DMA-AF o teste de Friedman indicou valor- $p=0,000$. Para que se possa avaliar melhor a qualidade dos resultados das heurísticas construtivas a Tabela 14 compara seus resultados com as soluções ótimas e limitantes fornecidos pelo CPLEX.

Tabela 14 - Resultados do CPLEX e das heurísticas construtivas.

Dimensão ($n \times m$)	CPLEX			EDD-AAF		MDD-AAF		DMA-AF	
	T_{total}	gap (%)	Tempo (s)	T_{total}	Tempo (s)	T_{total}	Tempo (s)	T_{total}	Tempo (s)
5×5	120,3*	0,00	1,68	187,7	0,000	183,3	0,000	187,6	0,000
10×5	404,6	0,00 [∞]	1984,00	554,7	0,000	543,6	0,000	546,8	0,000
10×10	550,8	603,51 [∞]	3053,41	587,2	0,001	585,1	0,000	585,7	0,001
15×10	1248,7	4803,34 [∞]	3382,768	866,1	0,001	876,1	0,001	881,4	0,001
15×15	—	—	—	1130,1	0,004	1101,9	0,005	1103,9	0,003
30×15	—	—	—	2750,6	0,009	2816,4	0,009	2825,8	0,010
50×5	—	—	—	17355,3	0,002	17266,9	0,002	17248,7	0,001
50×10	—	—	—	11242,6	0,007	11168,6	0,008	11142,6	0,007
50×15	—	—	—	8746,2	0,021	8806,9	0,022	8839,0	0,022
100×10	—	—	—	64681,6	0,025	65638,3	0,029	65642,4	0,028
Média parcial (5×5 a 15×10)	581,1	1351,71	2049,88	548,9	0,001	547,0	0,000	550,4	0,000

* Subconjunto no qual todas as soluções ótimas foram encontradas.

Verifica-se pela Tabela 14 que nas 3 menores dimensões as heurísticas construtivas, como esperado, obtêm resultados piores que os do método exato. Por outro lado, com a dimensão 15×10 as três regras geraram média de atraso total inferior à do CPLEX, mesmo consumindo, em média, somente 1 milissegundo. A média de atraso total das 4 menores

dimensões da EDD-AAF foi 548,9, da MDD-AAF chegou a 547,0 e da DMA-AF ficou em 550,4, resultados melhores que a média igual a 581,1 do CPLEX. Esses resultados indicam que as regras com afinidade e flexibilidade cumprem seu papel, pois são capazes de fornecer resultados de qualidade razoável em pequeno tempo computacional.

7.4 BUSCAS LOCAIS

Esta seção traz os resultados das BLs propostas com o intuito de avaliar seus desempenhos isolados, tanto em termos de qualidade de soluções quanto em termos de tempo de processamento. Essa avaliação é importante porque as meta-heurísticas propostas, cada uma de um modo, utilizam buscas em vizinhança em suas estruturas básicas. Experimentos iniciais foram realizados para verificar o desempenho das BLs baseadas em inserção com um procedimento de verificação de factibilidade alternativo, proposto por Dautère-Pérès e Paulli (1997). Esses experimentos indicaram que o procedimento de Dell’Amico e Trubian (1993), embora menos simples que o procedimento alternativo de Dautère-Pérès e Paulli (1997), permitiu obter melhores soluções com igual tempo de execução. Isso se deve, provavelmente, ao fato do procedimento de Dell’Amico e Trubian (1993) evitar alguns movimentos que, embora factíveis, não geram melhoria. Assim, a BL baseada em Inserção (BL-INS) e a BL-INS com Atualizações Internas (BL-INS-AI) utilizaram o procedimento de Dell’Amico e Trubian (1993) descrito no Capítulo 5.

As buscas propostas, denominadas BL baseada em inversão de Arcos (BL-ARC), BL-ARC com Atualizações Internas (BL-ARC-AI), BL-INS e BL-INS-AI, foram executadas tendo como ponto de partida a solução gerada pela EDD-AAF, a melhor heurística construtiva implementada. A Tabela 15 mostra as melhorias das médias alcançadas por tais métodos e seus tempos médios de execução. Para cada subconjunto a diferença relativa do atraso total médio de uma BL em relação à média da solução inicial foi calculada como na Equação (51), com cada BL sendo o *método* 1 e a EDD-AAF sendo o *método* 2. Os sinais negativos indicam a redução do atraso total médio em relação à solução inicial. Caso a solução inicial tenha atraso total igual a zero, a melhoria é também igual a zero.

Tabela 15 - Resultados das buscas locais propostas.

Dimensão ($n \times m$)	BL-ARC			BL-ARC-AI			BL-INS			BL-INS-AI		
	T_{total}	%Dif	Tempo (s)	T_{total}	%Dif	Tempo (s)	T_{total}	%Dif	Tempo (s)	T_{total}	%Dif	Tempo (s)
5×5	181,8	-3,16	0,000	183,0	-2,50	0,000	163,4	-12,96	0,000	164,3	-12,47	0,001
10×5	512,6	-7,59	0,001	507,2	-8,56	0,001	472,7	-14,79	0,003	472,0	-14,91	0,003
10×10	576,5	-1,82	0,002	575,2	-2,04	0,001	450,2	-23,33	0,030	441,8	-24,77	0,014
15×10	841,5	-2,84	0,004	840,0	-3,01	0,002	658,6	-23,96	0,103	653,1	-24,59	0,039
15×15	1119,8	-0,91	0,006	1119,8	-0,91	0,004	857,6	-24,11	0,453	880,2	-22,12	0,119
30×15	2734,8	-0,57	0,034	2734,3	-0,59	0,025	2348,5	-14,62	4,566	2353,9	-14,42	0,784
50×5	17063,2	-1,68	0,206	17066,1	-1,67	0,055	16612,5	-4,28	1,604	16633,9	-4,16	0,399
50×10	11134,7	-0,96	0,230	11131,7	-0,99	0,090	10698,3	-4,84	5,263	10707,7	-4,76	1,067
50×15	8706,1	-0,46	0,246	8706,7	-0,45	0,108	8207,1	-6,16	23,152	8221,4	-6,00	3,718
100×10	64378,0	-0,47	4,719	64382,0	-0,46	0,889	63698,4	-1,52	63,809	63682,3	-1,54	9,608
Média geral	10724,9	-2,05	0,545	10724,6	-2,12	0,118	10416,7	-13,06	9,898	10421,0	-12,97	1,575

Pelos valores da Tabela 15 se constata que a BL-INS e a BL-INS-AI tiveram resultados semelhantes, todos bem superiores aos da BL-ARC e da BL-ARC-AI. As melhorias da BL-INS foram as mais altas em 6 dimensões, enquanto as melhorias da BL-INS-AI foram superiores nos outros 4 subconjuntos. No geral, a BL-INS foi ligeiramente melhor que a BL-INS-AI e isso se refletiu no teste de Friedman que, com valor- $p=0,001$, indica existir diferença entre essas duas BLs. Na média geral a BL-ARC-AI foi levemente superior à BL-ARC, mas isso não foi suficiente para o teste de Friedman detectar diferenças significativas, indicando valor- $p=0,285$. Mais relevante que as diferenças entre os atrasos, a utilização de atualizações internas permitiu que BL-ARC-AI e BL-INS-AI fossem bem mais rápidas que BL-ARC e BL-INS, respectivamente. Enquanto a BL-INS teve tempo médio de 9,898 segundos, a BL-INS-AI gastou, em média, 1,575 segundos, uma redução de 84%. Os tempos das buscas baseadas em inversão com e sem atualizações internas tiveram diferença proporcional. Por essas razões a análise se concentra mais sobre os métodos com atualizações internas.

A maior melhoria de média da BL-INS-AI ocorreu no subconjunto 10×10, sendo igual a 24,77%. Em todas as dimensões com até 30 *jobs* as melhorias chegaram a dois dígitos. Já com as maiores dimensões as melhorias foram mais modestas, com a menor melhoria ocorrendo na maior dimensão. Vale ressaltar que, apesar de as melhorias relativas não terem sido tão expressivas nas maiores dimensões, as melhorias foram bastante evidentes. Na dimensão 100×10, por exemplo, a média do atraso total da EDD-AAF foi igual a 64681,6 e da BL-INS-AI foi igual a 63682,3, indicando uma diferença média de 999,2 unidades de tempo de atraso por instância. Além da grandeza dos atrasos, outra razão para as melhorias alcançadas com as maiores instâncias serem percentualmente menos expressivas é o grande

número de soluções factíveis envolvidas. Como as regiões factíveis dessas instâncias são muito maiores que as regiões factíveis das instâncias menores, cada vizinhança explorada representa uma porção proporcionalmente menor do total de possíveis soluções.

O desempenho da BL-ARC-AI comparado ao da BL-INS-AI foi bastante modesto. Sua maior melhoria de média ocorreu com 10 *jobs* e 5 máquinas, chegando a 8,56%, e sua média geral de melhoria ficou em 2,12%. As razões para esse desempenho inferior seriam a não realização de trocas de operações de máquina e a limitação do movimento de inversão de arcos disjuntivos. Tal desempenho reforça a importância da proposta deste trabalho de considerar vizinhanças geradas por movimentos mais abrangentes, especificamente pela inserção. Apesar desses resultados, a BL-ARC-AI não deve ser necessariamente descartada por causa de sua rapidez. Se por um lado a BL-ARC-AI conseguiu melhorias que ficaram em torno de 1/6 das melhorias da BL-INS-AI, por outro lado consumiu somente 1/13 do tempo computacional da BL-INS-AI, tornando seu custo-benefício interessante.

7.5 ITERATED LOCAL SEARCH

Nesta seção são apresentados e discutidos os resultados obtidos pela *Iterated Local Search* (ILS) proposta. Partindo de uma solução inicial fornecida por um método construtivo, essa meta-heurística alterna uma BL e um procedimento de perturbação para encontrar melhores soluções. Inicialmente são descritos os experimentos preliminares realizados para selecionar e ajustar os componentes da ILS. Em seguida são analisados os resultados encontrados pela ILS proposta.

7.5.1 Seleção e ajuste de componentes

O desenvolvimento da ILS proposta envolveu a seleção da heurística construtiva e da BL a serem utilizadas, além da calibragem de seus parâmetros. Também foram investigadas diversas estratégias de perturbação e diferentes variações da meta-heurística. Cada etapa desse processo de desenvolvimento é descrito com maiores detalhes na sequência.

Seleção da heurística construtiva e da Busca Local

Para a escolha da heurística construtiva a ser utilizada pela ILS, foram realizados experimentos com as melhores 3 regras apresentadas na Seção 7.3, sendo elas EDD-AAF, MDD-AAF e DMA-AF. Os resultados indicaram que a EDD-AAF é o melhor ponto de partida para a meta-heurística. Tal fato era esperado dado que a EDD-AAF apresentou desempenho significativamente superior às demais regras, especialmente quando foram contabilizadas as quantidades de melhores soluções.

Para a definição da BL a ser utilizada pela ILS foram realizados experimentos envolvendo BL-ARC-AI e BL-INS-AI, uma vez que essas buscas tiveram desempenho similar às suas correspondentes sem atualizações internas, mas com tempos bem inferiores. Além disso, apesar de as melhorias da BL-INS-AI terem sido amplamente superiores, a relação entre melhoria e tempo de resolução da BL-ARC-AI foi tão ou mais favorável que a da BL-INS-AI. Cabe observar que no caso da ILS com BL-ARC-AI as trocas de máquina são realizadas exclusivamente pelo procedimento de perturbação. Para ambas as versões da ILS, uma utilizando a BL-ARC-AI e outra utilizando a BL-INS-AI, foi concedido o mesmo tempo computacional de resolução e, no restante, os mesmos ajustes de parâmetros. Os resultados indicam que a ILS com a BL-INS-AI obteve desempenho amplamente superior. Seus atrasos foram inferiores em todas as dimensões, com diferenças variando de -1,70%, na dimensão 100×10 , a -32,97%, na dimensão 15×15 , em relação à ILS com a BL-ARC-AI. A ILS com a BL-INS-AI obteve 592 melhores soluções, enquanto que a ILS com a BL-ARC-AI obteve 84 melhores resultados. De fato o teste de Friedman indica haver diferenças estatisticamente significativas entre tais versões ao resultar em valor- p igual a 0,000. Esses resultados podem indicar que, mesmo utilizando o procedimento de perturbação que efetua trocas aleatórias de máquina, a ILS com BL-ARC-AI foi prejudicada pelo fato dessa BL não investigar trocas de máquinas. Por tais razões a BL-INS-AI foi selecionada para compor a ILS. Os atrasos médios da ILS com BL-ARC-AI podem ser consultados no Apêndice B.

Calibragem dos parâmetros

O critério de parada da ILS foi definido como um determinado número de iterações (NI). Foram realizados experimentos com elevados valores de NI , precisamente $100000/n$, para se observar a evolução do atraso total no decorrer da resolução. O objetivo foi analisar a relação entre a melhoria obtida e o número de iterações, identificando o número de iterações necessárias para que a meta-heurística se aproximasse da estabilização do valor do atraso total. Esses experimentos envolveram 60 instâncias, equivalendo a 10% do total, e contemplaram igualmente as 10 dimensões e os 3 níveis de flexibilidade. Foram usadas instâncias com datas de entrega mais apertadas a fim de que o atraso total não atingisse valor zero, comprometendo as análises. Verificou-se que, usualmente, nas primeiras iterações as reduções do atraso são bastante significativas e, à medida que as iterações avançam, essas reduções se tornam menos expressivas. Para exemplificar, a Figura 29 mostra a redução do atraso total de 6 instâncias de dimensão 10×10 e a média desses atrasos no decorrer das

iterações. Para melhor visualização o eixo horizontal foi restrito às primeiras 1000 iterações e o eixo vertical foi restrito para valores de atraso de 300 a 1300.

Figura 29 - Atraso total em função do número de iterações da ILS.

Fonte: elaborada pelo autor.

A Figura 29 evidencia como as melhorias mais significativas com as instâncias 10×10 ocorrem nas primeiras 200 iterações, sendo que depois disso o atraso se aproxima da estabilização. Nos experimentos se identificou que o atraso total médio fornecido pela EDD-AAF foi igual a 1214,3. Esse atraso decai rapidamente encerrando a décima quinta iteração com valor 800,5. Em seguida vem uma série de melhorias constantes, mas não tão expressivas quanto as anteriores, as quais permitem que o atraso médio chegue a 699,0 após 130 iterações. Então se seguem períodos de estagnação e de leves melhorias, com a tendência de que os períodos de estagnação se tornem cada vez mais longos. Nestes experimentos a última melhoria foi constatada na iteração 4533, quando o atraso total médio alcançou valor 606,5, não havendo qualquer melhoria posterior até o término da iteração de número 10000. Destaca-se que na iteração 300 a ILS já havia alcançado atraso médio igual a 654,5. Com 600 iterações o método já havia reduzido o atraso inicial em 46,46%, consumindo cerca de 14s de processamento por instância. Nota-se que as melhorias após este ponto são relativamente pequenas. Considerando o desempenho da ILS com 400 iterações adicionais, correspondendo às 1000 iterações representadas no gráfico, a melhoria adicional em relação à média inicial seria de somente 0,19%. Vale observar que para alcançar atraso médio igual a 606,5 é necessário que o tempo de resolução seja aproximadamente 7,5 vezes maior que o gasto com 600 iterações, totalizando 110 segundos. Nessa situação a ILS conseguiria 3,31% de melhoria por segundo de resolução nos 14s iniciais, enquanto nos restantes 96 segundos a meta-heurística obteria somente 0,04% de melhoria por segundo. Então, com base nesse

estudo, para a dimensão 10×10 se estabeleceu o limite de 600 iterações, indicada por um traço vertical na Figura 29. Essa quantidade de iterações permite uma maior segurança em relação à obtenção de melhorias sem que o tempo de execução seja estendido demasiadamente.

Para as demais dimensões foram gerados gráficos similares para identificar o comportamento do atraso total em função das iterações. Contudo, à medida que as instâncias se tornam maiores, mais iterações são necessárias para se atingir a estabilidade, descrevendo curvas descendentes bastante suaves. Ao mesmo tempo, quanto maior a relação $n \times m$, maior o tempo computacional exigido por iteração, similarmente ao mostrado anteriormente com a BL-INS-AI. Então se decidiu que, não sendo viável aguardar a estabilização, se trabalharia com um número de iterações que permitisse aproveitar as melhorias mais significativas, ou seja, as partes mais inclinadas das curvas. Assim se chegou a uma expressão que determina o número de iterações em função do número de *jobs* e do número de máquinas do seguinte modo: $NI = 60000 / (n \cdot m)$. Essa limitação aproveita as grandes melhorias iniciais sem estender demasiadamente o tempo computacional, o que é especialmente crítico com as maiores dimensões. Basta informar que, para a elaboração de tais gráficos, cada instância 100×10 explorada por 1000 iterações teve tempo médio de resolução igual a 4 horas e 15 minutos. Ao mesmo tempo, os problemas de menor porte puderam ser explorados por um número maior de iterações, uma vez que investigar suas vizinhanças tem um custo computacional razoavelmente pequeno.

Também foram realizados testes para definir o nível de perturbação utilizado pela ILS através do procedimento *Perturbar*. Esse nível de perturbação é definido pelo número de trocas aleatórias de posição (*NTP*) e pelo número de trocas aleatórias de máquina (*NTM*). Foram investigados valores relacionados às dimensões das instâncias, desde pequenos percentuais do número de operações até frações do número de *jobs* ou de máquinas. Com valores fixos foram avaliadas todas as combinações do parâmetro *NTP* variando de 1 a 5 e com o parâmetro *NTM* variando de 0 a 4. Também foram feitos experimentos com valores determinados aleatoriamente dentro de faixas previamente estabelecidas para os parâmetros *NTP* e *NTM*. Os resultados indicaram que, independentemente dos demais parâmetros, utilizar valores fixos permitiu alcançar melhores soluções, sendo que a melhor configuração utilizou $NTP=3$ e $NTM=1$. Assim ficou definido que, sendo possível, a cada vez que o procedimento fosse executado, três operações deveriam ser trocadas aleatoriamente de posição na mesma máquina e uma operação deveria ser trocada aleatoriamente de máquina.

Diferentes estratégias de seleção do procedimento de perturbação

Foram realizadas investigações sobre os efeitos de diferentes estratégias de seleção das operações a serem movimentadas pelo procedimento de perturbação. A primeira estratégia de seleção, denominada *Neutra*, é completamente aleatória. Nessa estratégia todas as operações têm igual probabilidade de sorteio para movimentação. Para as outras duas estratégias foi utilizada uma matriz de frequência tridimensional que acumula, a cada iteração, a posição de cada operação em cada máquina no decorrer das iterações. Tais valores históricos são usados para permitir perturbações direcionadas. A Figura 30 exemplifica tal matriz tridimensional.

Figura 30 - Exemplo de matriz de frequência.

Fonte: elaborada pelo autor.

A matriz de frequência tridimensional da Figura 30 acumula as posições que 4 operações ocuparam em 3 máquinas no decorrer de algumas iterações. Verifica-se que, por exemplo, a operação 1 esteve em 4 ocasiões na primeira posição da máquina 3 e 3 vezes na segunda posição dessa máquina. Essa mesma operação já ocupou 2 vezes a primeira posição da máquina 2 e nunca foi alocada na máquina 1.

Na primeira estratégia de perturbação direcionada investigada, denominada *Direcionada A*, a probabilidade de sorteio de cada operação é proporcional ao número de vezes que ela já esteve nessa mesma posição da máquina atual. Com isso as soluções perturbadas tendem a se diferenciar mais das soluções históricas. Essa matriz de frequência também foi utilizada no sentido contrário, ao tornar a probabilidade de sorteio de cada operação inversamente proporcional ao número de vezes que a mesma ocupou a posição atual. Com isso existe a tendência de que operações que ocupam posições pouco frequentes sejam sorteadas, uma vez que tais posições não devem ser suas melhores colocações. Essa estratégia foi chamada de *Direcionada B*. Foram testadas adicionalmente estratégias mistas

que alternam as estratégias mencionadas a cada iteração, objetivando visitar locais da região factível mais distantes ou mais próximos da solução incumbente. Primeiro se testou alternar as estratégias *Direcionada A* e *Direcionada B* e também alternar *Neutra* e *Direcionada A*. Depois se testou alternar ciclicamente as estratégias na ordem *Neutra*, *Direcionada A*, *Neutra* e *Direcionada B*. Também foi testada a troca de estratégia condicionada a certo número de iterações sem melhoria, aumentando a diversidade das soluções perturbadas caso o método sofresse estagnação. Iniciava-se com a estratégia *Neutra* ou com a estratégia *Direcionada B* e, caso fosse alcançado certo número de iterações sem melhoria, era acionada a estratégia *Direcionada A*. Foram testadas diferentes tolerâncias do número de iterações sem melhoria.

Contudo as diferenças dos resultados das estratégias, sejam isoladas ou combinadas, foram insignificantes, com leve vantagem para a estratégia *Neutra* aplicada isoladamente. Uma possível explicação pode ser o excesso de perturbação promovida pela estratégia *Direcionada A*, bem como a insuficiência de perturbação exercida pela estratégia *Direcionada B*. Em contrapartida, a estratégia *Neutra* promove perturbações intermediárias e verdadeiramente aleatórias. É interessante observar que, mesmo nas dimensões com menor espaço de soluções e elevados números de iterações, a estratégia *Direcionada A* não se mostrou melhor que a aleatoriedade pura. Com tudo isso se optou pela ILS utilizar a estratégia *Neutra*, inclusive porque a mesma preserva a simplicidade do método, uma qualidade importante para métodos heurísticos, segundo Cordeau *et al.* (2002).

Diferentes variações da ILS

Além da ILS anteriormente apresentada, foi investigado o efeito da incorporação de elementos adicionais sobre os resultados. Para tanto duas novas versões derivadas da ILS proposta foram implementadas. A primeira investigação buscou explorar simultaneamente diferentes pontos do espaço de soluções ao utilizar um conjunto de soluções de elite. A versão denominada ILS-Elite armazena certo número de soluções de elite (*NSE*) durante a resolução, sendo que tais soluções são as melhores obtidas no decorrer do processo. Inicialmente uma solução é gerada pela heurística construtiva e sobre ela é aplicada a BL. O ótimo local se torna solução incumbente e a primeira solução de elite. Nas *NSE*-1 primeiras iterações, todos os ótimos locais entram no conjunto de elite. A partir da *NSE*-ésima iteração cada novo ótimo local somente entra no conjunto elite se for melhor que a pior das soluções de elite do momento. Nesse caso essa nova solução substitui a pior das soluções de elite. A cada iteração, a solução a ser perturbada é sorteada dentre as soluções de elite com probabilidade proporcional à sua qualidade, medida em termos do atraso total, segundo o método *roulette*

wheel. Para que a ILS-Elite não se descaracterizasse como método de trajetória e não tomasse características de método populacional, se definiu *NSE* como um pequeno valor, igual a 4.

A outra versão derivada da ILS proposta buscou ajustar o nível de perturbação à medida que melhorias da solução incumbente eram ou não observadas. Essa versão de ILS com perturbação variável foi chamada de ILS-PV. Seu funcionamento é similar à ILS original, mas a cada vez que ocorre certo número de iterações consecutivas sem melhoria, as quantidades de trocas aleatórias de posição e de máquina são incrementadas. Quando uma melhoria da solução incumbente é observada, esses parâmetros são restaurados aos valores iniciais. Os valores iniciais utilizados foram os mesmos da ILS proposta, $NTP=3$ e $NTM=1$. Na versão implementada da ILS-PV os incrementos de NTP e NTM se baseiam em frações do número total de iterações. Foram realizados experimentos iniciais para ajustar esses incrementos e se chegou à conclusão de que NTP deveria sofrer mais incrementos que NTM . Além disso, após cada incremento, é necessário conceder tempo para que o ajuste da perturbação possa surtir algum efeito. Então ficou definido que o valor de NTP é incrementado a cada vez que o número de iterações sem melhoria alcança múltiplos de $NI/10$, enquanto NTM é incrementado quando as iterações sem melhoria se igualam a múltiplos de $NI/4$.

Foram realizados experimentos envolvendo a ILS proposta original, a ILS-Elite e a ILS-PV, todas executadas pelo mesmo número de iterações, o que gerou tempos de resolução similares. Os resultados indicaram que nem incorporar soluções de elite nem variar a perturbação leva a maiores reduções de atraso. A ILS original obteve menor atraso total em 8 das 10 dimensões enquanto a ILS-Elite gerou a menor média nas dimensões 5×5 e 10×5 . Além disso, a ILS original foi a que alcançou o maior número de melhores soluções. Esses resultados guardam coerência com a pesquisa de Pan e Ruiz (2012), na qual uma ILS tradicional superou uma ILS populacional. Apesar da ILS original ter obtido melhores resultados, o teste de Friedman não detectou a existência de diferenças estatisticamente significativas entre ela e a ILS-Elite, pois resultou em valor- $p=0,105$. Já entre a ILS original e a ILS-PV o teste indicou haver diferenças significativas ao encontrar valor- $p=0,000$. Adicionalmente foram realizados testes com a ILS-Elite utilizando a estratégia de seleção do procedimento de perturbação *Direcionada A*, mas seus resultados foram inferiores aos da ILS-Elite anteriormente testada que utiliza seleção aleatória. Para verificar a consistência das diferenças de desempenhos entre as versões da ILS, também foram realizados experimentos com outras quantidades de iterações. Em todas as situações os melhores resultados foram da ILS proposta inicialmente. Pelo desempenho observado a proposta inicial da ILS foi mantida

e seus resultados são apresentados na sequência. Os resultados da ILS-Elite original e da ILS-PV estão disponíveis no Apêndice B.

7.5.2 Resultados da ILS proposta

Com a escolha dos componentes e a calibragem dos parâmetros, os resultados da ILS proposta podem ser apresentados e analisados. Vale reiterar que essa ILS utiliza a solução inicial da EDD-AAF, aplica a BL-INS-AI e trabalha com $NTP=3$, $NTM=1$, $NI=60000/(n \cdot m)$, além de selecionar aleatoriamente as operações a serem perturbadas. O desempenho da ILS proposta pode ser avaliado com o auxílio da Tabela 16. As melhorias por dimensão, %*Dif*, da ILS em relação à EDD-AAF foram obtidos para cada subconjunto de modo análogo ao da Equação (51). Os resultados estão divididos de acordo com as dimensões das instâncias e com os níveis médios de flexibilidade. São apresentados os tempos médios de execução da ILS em cada subconjunto composto por 20 instâncias. Na penúltima coluna constam as melhorias das médias das 60 instâncias de cada dimensão e na última coluna, seus tempos de CPU médios. Na última linha estão as médias de %*Dif* e do tempo utilizado.

Tabela 16 - Melhorias da ILS em relação à EDD-AAF.

Dimensão ($n \times m$)	Flexibilidade 20%		Flexibilidade 50%		Flexibilidade 80%		Geral	
	% <i>Dif</i>	Tempo (s)	% <i>Dif</i>	Tempo (s)	% <i>Dif</i>	Tempo (s)	% <i>Dif</i>	Tempo (s)
5×5	-11,61	0,584	-58,64	1,154	-61,23	1,018	-32,71	0,919
10×5	-33,00	2,036	-54,36	3,317	-59,94	2,860	-39,86	2,738
10×10	-56,39	3,508	-59,80	6,503	-57,25	8,341	-57,41	6,117
15×10	-49,42	7,939	-56,41	9,275	-58,27	11,205	-52,40	9,473
15×15	-54,95	15,634	-58,35	23,945	-47,13	37,137	-54,32	25,572
30×15	-34,63	52,247	-29,37	100,218	-34,51	129,654	-33,28	94,040
50×5	-10,54	26,595	-6,87	110,128	-10,57	159,900	-9,50	98,874
50×10	-16,85	82,386	-4,03	195,442	-4,38	306,109	-10,37	194,646
50×15	-16,42	150,383	-6,72	351,542	-7,66	521,857	-11,62	341,261
100×10	-4,30	322,424	-1,25	824,579	-1,08	1283,880	-2,36	810,294
Média geral	-28,81	66,374	-33,58	162,610	-34,20	246,196	-30,38	158,393

Os resultados da Tabela 16 mostram que a ILS foi capaz de obter melhorias em todos os subconjuntos, com reduções de atraso mais significativas nos problemas com até 30 *jobs*. Analisando os níveis de flexibilidade separadamente, se constata que em problemas menores houve uma tendência de que as melhorias fossem mais amplas em níveis de flexibilidade mais altos. Com as instâncias de maior porte a ILS se saiu melhor com níveis de flexibilidade inferiores. Uma possível razão para isso é o crescimento do número de soluções do problema. É provável que, por exemplo, na dimensão 50×10 o número de iterações necessárias para alcançar determinado nível de melhoria seja maior que na dimensão 50×5, pois o número de

soluções possíveis é bastante maior e deve haver mais ótimos locais a ultrapassar. Apesar disso, mesmo nos maiores problemas, a ILS conseguiu melhorias significativas. Nas instâncias com 50 *jobs* a melhoria média máxima foi de 16,85% e a mínima de 4,03%. Já com 100 *jobs* os melhores resultados foram obtidos com o nível médio de flexibilidade mais baixo, 4,30% de melhoria. Nos demais níveis as melhorias dos problemas com 100 *jobs* não foram tão expressivas e isso se deve, em grande parte, à limitação imposta ao número de iterações da ILS e ao maior número de soluções do problema.

Considerando agora todos os níveis de flexibilidade, observando os valores da última coluna, se verifica que, nos subconjuntos com 15 *jobs*, as melhorias ultrapassaram 50% e no subconjunto 10×10 ela chegou a 57,41%. Nas dimensões 5×5 e 10×5 as melhorias foram de 32,71% e 39,86%, respectivamente. Provavelmente esses percentuais são reflexo da menor distância entre as soluções iniciais e as soluções ótimas, o que impediu melhorias maiores que nas dimensões com 15 *jobs*. Nas dimensões com 50 *jobs* ou mais as melhorias relativas foram mais modestas. Isso se deve, em grande parte, ao menor número de iterações às quais esses problemas maiores foram submetidos. Nos testes preliminares usados para definir o número de iterações se observou que, mesmo com um valor de *NI* igual para todas as dimensões, há discrepâncias entre as melhorias de diferentes dimensões. Isso ocorre porque em problemas de menor porte são necessárias menos iterações para que boa parte da região factível seja explorada. Em problemas de grande porte, com muito mais soluções possíveis, esse mesmo número de iterações é insuficiente para se explorar uma área proporcional da região factível. Então, a forma de conseguir melhorias mais expressivas nas maiores dimensões seria permitir mais iterações com as maiores instâncias, mas isso aumentaria demasiadamente o tempo de resolução.

Com relação ao tempo computacional da ILS, o mesmo pode ser considerado aceitável. Evidentemente, do mesmo modo que as iterações da ILS promoveram melhorias mais amplas em relação à BL-INS-AI, também custaram o aumento do tempo computacional. Nas instâncias mais demoradas, 100×10, o tempo de processamento médio foi de 810,294 segundos. Considerando todas as 600 instâncias o tempo médio de resolução da ILS foi de 158,393 segundos, o que seria aceitável até mesmo em aplicações práticas.

Também foram identificadas as melhorias obtidas pela ILS com cada tipo de data de entrega. Foi possível observar que, à medida que as datas de entrega se tornam menos restritivas, as melhorias aumentam. Com as datas de entrega mais apertadas a melhoria média geral foi igual a 21,30%. Com as datas de entrega intermediárias essa melhoria chegou a 38,23%. Com as datas mais tolerantes a melhoria alcançou 53,26%. Com as datas combinadas

aleatoriamente, CA, a melhoria geral foi de 41,11%. É preciso mencionar que com datas mais folgadas os valores absolutos dos atrasos são menores e, assim, uma mesma redução absoluta do atraso resulta em melhorias percentuais mais expressivas.

Através da Tabela 17 se pode verificar como os artifícios utilizados pela meta-heurística ILS permitem que melhorias sejam obtidas em relação à aplicação isolada da BL-INS-AI.

Tabela 17 - Melhorias da ILS em relação à BL.

Dimensão ($n \times m$)	Flexibilidade 20%		Flexibilidade 50%		Flexibilidade 80%		Geral	
	%Dif	Melhorias	%Dif	Melhorias	%Dif	Melhorias	%Dif	Melhorias
5×5	-6,54	5	-49,39	17	-47,57	13	-23,12	35
10×5	-22,87	20	-41,42	15	-53,23	12	-29,32	47
10×10	-41,85	20	-43,99	15	-47,43	15	-43,39	50
15×10	-32,44	20	-44,89	16	-42,79	17	-36,87	53
15×15	-42,18	20	-43,82	15	-35,98	14	-41,35	49
30×15	-24,02	20	-17,77	17	-22,03	17	-22,04	54
50×5	-7,87	20	-2,90	19	-4,94	20	-5,58	59
50×10	-10,50	20	-1,77	19	-1,63	18	-5,89	57
50×15	-11,16	20	-1,31	14	-1,02	9	-5,98	43
100×10	-1,84	19	-0,21	9	-0,22	8	-0,83	36
Média geral	-20,13	–	-24,75	–	-25,69	–	-21,44	–
Total	–	184	–	156	–	143	–	483

Como esperado, as melhorias alcançadas pela ILS em relação à execução isolada da BL-INS-AI foram significativas e foram mais altas com flexibilidades mais elevadas. Com a flexibilidade mais baixa a melhoria das médias foi de 20,13%. Com flexibilidade 50% essa melhoria aumentou para 24,75% e chegou a 25,69% na flexibilidade mais alta. De fato, nas tabelas anteriores é possível notar que a melhoria da BL-INS-AI sobre a solução inicial foi próxima a 13%, enquanto a ILS conseguiu, sobre as mesmas soluções iniciais, 30,38% de redução média do atraso. Em 483 ocasiões a ILS encontrou melhor solução que a BL-INS-AI.

As melhorias da ILS também se mostram relevantes quando comparadas às Buscas Tabu (BT) implementadas em Scrich (1997), as quais utilizam somente os movimentos de inversão de arcos e trocas de máquina. Os métodos são os mesmos de Scrich; Armentano e Laguna (2004), mas a comparação é feita com Scrich (1997), pois em Scrich; Armentano e Laguna (2004) a dimensão 15×15 não é apresentada. Scrich (1997) fornece as médias das melhorias por instância, o que não é necessariamente igual às melhorias das médias por subconjunto presentes nas tabelas anteriores. Para obter as médias das melhorias por instância primeiramente é preciso calcular a melhoria percentual obtida sobre a solução inicial para cada instância. Então é calculada a média dessas melhorias percentuais. Nesse cálculo,

Scrich (1997) ainda descarta melhorias iguais a 100%, casos em que a BT zerou o atraso presente na solução inicial. Considerando isso, a melhor BT com 250 iterações conseguiu 12,16% de média de melhoria sobre a MDD no nível de 20% de flexibilidade; 13,19% no nível de 50% de flexibilidade; e 7,13% no nível de 80%. Com esse mesmo cálculo a ILS conseguiu sobre a EDD-AAF, respectivamente, 30,02%, 27,62% e 23,79%. Desse modo, apesar de não serem as mesmas instâncias, existem indícios de que a ILS tem desempenho melhor que o da BT. É preciso notar que a média de atraso da EDD-AAF considerando todas as instâncias é 26,87% inferior à média da MDD. Se os mesmos cálculos forem feitos para comparar a ILS iniciando da EDD-AAF com a solução da MDD, as melhorias no nível mais baixo de flexibilidade ficam em 36,10%, no nível médio chegam a 53,26% e no nível mais alto são de 49,29%.

Com o intuito de melhor avaliar se o tempo computacional da ILS é razoável e se suas soluções são de qualidade, a Tabela 18 compara os resultados da meta-heurística aos do modelo do Capítulo 3 implementado em CPLEX 12.2. Na última coluna a tabela traz, para cada dimensão, a diferença relativa entre as médias de atraso total da ILS e do CPLEX limitado a 1 hora de processamento. Na última linha é exibida uma média parcial que somente considera as 4 menores dimensões.

Tabela 18 - Resultados do método exato e da ILS.

Dimensão ($n \times m$)	CPLEX			ILS			%Dif
	T_{total}	Tempo (s)	Melhores soluções	T_{total}	Tempo (s)	Melhores soluções	
5×5	120,3*	1,679	60	126,3	0,919	42	4,97
10×5	404,6	1919,324	34	333,6	2,738	48	-17,55
10×10	550,8	2895,763	22	250,1	6,117	49	-54,59
15×10	1248,7	3382,768	8	412,3	9,473	60	-66,98
15×15	—	—	—	516,2	25,572	—	—
30×15	—	—	—	1835,1	94,040	—	—
50×5	—	—	—	15706,4	98,874	—	—
50×10	—	—	—	10076,6	194,646	—	—
50×15	—	—	—	7730,1	341,261	—	—
100×10	—	—	—	63156,5	810,294	—	—
Média parcial (5×5 a 15×10)	581,1	2049,884	—	280,6	4,812	—	-33,54

* Subconjunto no qual todas as soluções ótimas foram encontradas.

Além dos valores de atraso total médio, a Tabela 18 oferece o número de vitórias que CPLEX e ILS tiveram em cada subconjunto, incluindo empates. No subconjunto 5×5 o CPLEX encontrou a solução ótima em todas as 60 instâncias. Nesse mesmo subconjunto a ILS conseguiu obter 42 soluções ótimas e média de atraso somente 4,97% superior à média do CPLEX. Nas 18 soluções em que a ILS não obteve o ótimo, seu atraso total médio foi igual a

79,3, ao passo que o atraso médio dos respectivos ótimos foi 59,3. Na dimensão 10×5 o CPLEX chegou à média de atraso total 404,6 e teve 34 melhores soluções, enquanto a ILS teve média 17,55% inferior, igual a 333,6, com 48 melhores soluções. No subconjunto seguinte os resultados da ILS foram 54,59% melhores que os da resolução exata, com 250,1 de atraso total médio da ILS contra 550,8 do CPLEX. Na dimensão 15×10 a resolução exata gerou atraso médio igual a 1248,7 e a ILS teve média igual a 412,3 conseguindo 66,98% de redução de atraso. Além disso, em todas as 60 instâncias desse subconjunto a ILS conseguiu resultado melhor ou igual à resolução exata limitada a 1 hora. Os resultados do CPLEX são apresentados até a dimensão 15×10, pois esta foi a maior dimensão em que o *software* conseguiu limitante superior para todas as instâncias após 1 hora de processamento.

Com tais resultados se pode afirmar que o desempenho da ILS foi satisfatório. A meta-heurística proposta conseguiu chegar a 83 das 117 soluções ótimas encontradas pela resolução exata, sendo: 42 em instâncias com 5 *jobs*, 22 na dimensão 10×5, 11 na dimensão 10×10 e 8 na dimensão 15×10. Sua média de atraso total nessas 117 instâncias ficou somente 3,53% acima dos valores ótimos. Contabilizando somente as instâncias, dentre essas 117, para as quais a ILS não encontrou solução ótima, seu atraso ficou a 7,34% dos ótimos. Considerando as 240 instâncias usadas na comparação o atraso total da ILS foi 33,54% inferior ao do CPLEX. Esses números evidenciam a qualidade das soluções da ILS. Na Tabela 19 os resultados das instâncias de dimensão até 15×10 são estratificados por nível médio de flexibilidade, sendo que cada linha corresponde a 20 instâncias. Na última coluna as diferenças relativas são calculadas sobre as médias de cada subconjunto.

Tabela 19 - Resultados do CPLEX e da ILS por nível de flexibilidade.

Nível de flexibilidade (%)	Dimensão ($n \times m$)	Instâncias	CPLEX		ILS		%Dif
			T_{total}	Melhores soluções	T_{total}	Melhores soluções	
20	5×5	20	279,4*	20	279,4	20	0,00
	10×5	20	769,3*	20	788,5	9	2,50
	10×10	20	491,6	11	441,6	14	-10,18
	15×10	20	1279,5	2	814,7	20	-36,32
50	5×5	20	54,3*	20	60,1	12	10,59
	10×5	20	245,6	7	139,9	19	-43,03
	10×10	20	645,0	4	181,9	19	-71,80
	15×10	20	1269,3	3	233,6	20	-81,60
80	5×5	20	27,3*	20	39,5	10	44,77
	10×5	20	199,1	7	72,4	20	-63,64
	10×10	20	515,8	7	126,9	16	-75,41
	15×10	20	1197,4	3	188,6	20	-84,25

* Subconjunto no qual todas as soluções ótimas foram encontradas.

À medida que a dimensão e o nível médio de flexibilidade aumentam, as médias de atraso total e os números de melhores resultados da resolução exata e da ILS evoluem conforme o esperado. O melhor desempenho do CPLEX ocorre com a menor dimensão, 5×5 , e com o nível de flexibilidade mais baixo, onde também é melhor na dimensão 10×5 . À medida que a dimensão e a flexibilidade crescem, a ILS melhora seu desempenho relativo. Isso fica claro ao se observar as expressivas melhorias obtidas com o nível de 80% de flexibilidade, cuja média é de 44,63%.

Os desempenhos da ILS e do CPLEX podem ser observados à medida que as instâncias crescem pelo gráfico da Figura 31. Esse gráfico de barras de desempenho representa em seu eixo horizontal cada uma das 240 instâncias usadas nessas comparações. Tais instâncias estão dispostas em ordem crescente, primeiramente de n e depois de m , da esquerda para a direita. Dentro de cada dimensão as instâncias estão agrupadas do menor ao maior nível de flexibilidade. Assim se pode considerar que os problemas estão ordenados por complexidade crescente. Para cada instância é plotada, verticalmente, uma barra de comprimento proporcional ao valor da diferença relativa entre os resultados do melhor e do pior método. A cor dessa barra indica o método que obteve a melhor solução com a respectiva instância. Com isso é possível observar, para cada instância, o método que conseguiu o melhor resultado e o quanto esse resultado foi melhor que o do método ao qual ele é comparado. A predominância de uma cor indica a superioridade do método correspondente.

Essa representação permite acompanhar o desempenho dos métodos através das instâncias e visualizar detalhes ocultos por valores médios. Nesse gráfico cada barra em vermelho indica um melhor resultado do CPLEX, enquanto cada barra em azul indica um melhor resultado da ILS. Nas instâncias em que há empate nenhuma barra é plotada.

Figura 31 - Melhoria percentual entre CPLEX e ILS por instância.

Fonte: elaborada pelo autor.

À extrema esquerda do eixo horizontal do gráfico estão identificadas as instâncias 5×5 com 20% de flexibilidade. Esse espaço está em branco e não possui qualquer barra porque ambos os métodos empataram em todas essas 20 instâncias. De fato isso pode ser confirmado na Tabela 19 ao se notar que, nesse subconjunto, CPLEX e ILS encontraram todas as soluções ótimas. Na sequência vêm as instâncias 5×5 com 50% de flexibilidade, onde as barras vermelhas mostram que, na maioria das vezes, o CPLEX conseguiu melhor resultado. As barras que chegam ao valor máximo da escala vertical indicam que, na respectiva instância, somente um dos métodos conseguiu zerar o atraso e a ausência de barras indica o empate. O CPLEX predominou, como indica a cor vermelha, até à dimensão 10×5 com 20% de flexibilidade. Com as instâncias 10×5 com 50% de flexibilidade a ILS começa a apresentar melhores resultados e a cor azul passa a predominar. No subconjunto 10×10 e 20% de flexibilidade o CPLEX volta a ter alguns melhores resultados, mas depois disso a ILS passar a predominar, o que é indicado pela maior presença da cor azul. Assim se nota que em cada nível de flexibilidade, à medida que a relação $n \times m$ cresce, a ILS melhora seu desempenho frente ao CPLEX. Do mesmo modo, considerando uma dimensão isoladamente, o desempenho relativo da ILS também melhora à medida que o nível de flexibilidade aumenta. Isso fica bastante claro com as instâncias 15×10 .

Todos esses resultados mostram como a resolução do JSF se torna mais difícil com o aumento da relação $n \times m$ ou do nível médio de flexibilidade. Eles também evidenciam que a ILS implementada é capaz de resolver instâncias do JSF, desde as menos até as mais difíceis, fornecendo soluções de qualidade em tempo computacional aceitável.

7.6 GRASP

A presente seção contém os resultados obtidos pela *Greedy Randomized Adaptive Search Procedure* (GRASP) proposta. Essa meta-heurística, baseada em construção de soluções, introduz determinado grau de aleatoriedade a um método construtivo e, iteração após iteração, gera uma nova solução e aplica sobre ela uma BL. Primeiramente são descritos os experimentos iniciais realizados para selecionar e ajustar os componentes da meta-heurística, incluindo a proposta de uma versão reativa da GRASP. Depois são apresentados e analisados os resultados da GRASP proposta.

7.6.1 Seleção e ajuste de componentes

No processo de implementação da GRASP proposta foi necessário tomar decisões

sobre a regra de prioridade a ser utilizada pela heurística construtiva com aleatoriedade, sobre a BL a ser empregada, sobre o número de iterações do método e sobre a forma de elaboração da Lista Restrita de Candidatas (LRC). Além disso, se investigou a utilização de conhecimento histórico obtido durante a execução do método em favor de sua própria calibragem. Em seguida as etapas do desenvolvimento da meta-heurística são apresentadas com maiores detalhes.

Seleção da heurística construtiva e da Busca Local

O método construtivo utilizado foi o algoritmo *Heur_Constr_Aleat*, descrito na Seção 6.2, associado a uma regra de prioridade. Primeiramente testes foram conduzidos para se determinar qual regra de prioridade deveria ser utilizada pela GRASP. Esses testes envolveram as 3 melhores regras implementadas neste trabalho. Os resultados mostraram a EDD-AAF como melhor escolha para a geração de soluções pela GRASP, assim como havia sido observado com a ILS.

A escolha da BL encarregada das melhorias da GRASP envolveu BL-ARC-AI e BL-INS-AI por essas buscas terem as melhores relações entre melhoria e tempo de execução. Enquanto a BL-INS-AI obteve melhorias mais significativas, a BL-ARC-AI se mostrou bastante rápida e, por tal razão, também foi incluída nos experimentos. Isso se justifica por ser possível que, concedendo tempos de execução equivalentes, a BL-ARC-AI, sendo mais rápida, permita à GRASP melhorias mais expressivas por mais soluções iniciais com aleatoriedade serem exploradas. Para ambas as versões da GRASP, uma com cada BL, foi concedido o mesmo tempo de execução. Os resultados indicam que utilizar a BL-INS-AI permitiu obter melhor desempenho, com atrasos médios inferiores nas 10 dimensões de instâncias. Na comparação direta, a GRASP com BL-INS-AI alcançou melhorias consideravelmente mais expressivas e, dentre as 600 instâncias, obteve 516 melhores soluções contra 196 melhores resultados da GRASP com BL-ARC-AI. Apesar disso, a diferença de desempenho observada entre a GRASP com BL-INS-AI e a GRASP com BL-ARC-AI foi menor que a diferença observada entre as versões da ILS que utilizaram essas mesmas BLs. A maior diferença de desempenho ocorreu na dimensão 10×10, na qual a GRASP com BL-INS-AI obteve atraso médio 26,63% menor que a GRASP com BL-ARC-AI. Isso pode ser um indício de que a geração com aleatoriedade da GRASP é capaz de promover diversidade suficiente para compensar parte da limitação da BL-ARC-AI, a qual não efetua troca de máquinas. Também é preciso considerar que a BL-ARC-AI é mais rápida que a BL-INS-AI e, assim, permite mais iterações, gerando mais instâncias com aleatoriedade. O

teste de Friedman foi executado utilizando os atrasos dessas versões da GRASP e chegou ao valor- $p=0,000$. Considerando um nível de significância igual a 0,05, o teste indica haver evidências para afirmar que existe diferença estatística significativa entre os resultados dessas versões da GRASP. Assim, como a BL-INS-AI permitiu obter melhores resultados, ela foi selecionada para compor a GRASP em desenvolvimento. Os resultados da versão da GRASP com BL-ARC-AI estão disponíveis no Apêndice C.

Elaboração da lista restrita de candidatas

Um ponto fundamental da meta-heurística GRASP é a maneira através da qual a LRC é elaborada, visto que isso é parte fundamental do processo de geração de soluções da meta-heurística. Por ser um método fortemente baseado no processo construtivo, é necessário que a geração de soluções da GRASP seja capaz de promover diversidade. Com isso são gerados pontos de partida suficientemente distribuídos no espaço de soluções, os quais devem ser explorados pela BL que segue cada geração de solução. Ao mesmo tempo, é importante que essa diversidade promovida pelo caráter aleatório da fase construtiva seja controlada, mantendo certo nível de qualidade. Sem isso há o risco da BL ser incapaz de encontrar soluções satisfatórias a partir de soluções excessivamente degradadas pela aleatoriedade da fase construtiva. Assim, considerando a importância dessa etapa para a GRASP, na sequência são apresentadas três propostas para a elaboração da LRC.

A primeira proposta foi compor a LRC com um número fixo de elementos, o que é denominado por Feo e Resende (1995) como restrição por cardinalidade. Para não se correr o risco de desperdiçar a qualidade das soluções da EDD-AAF, a primeira iteração da GRASP executa uma construção determinística. No restante das iterações as construções incorporam aleatoriedade. Durante a resolução, as regras de prioridade são utilizadas para estimar, de forma indireta, o impacto da escolha de cada operação candidata na solução em construção. Isso significa que este trabalho considerou as prioridades das operações em vez de avaliar diretamente as consequências da escolha de cada uma das operações no atraso total. Dessa forma foi possível continuar utilizando as regras de prioridade desenvolvidas, bem como evitar o alto custo computacional de avaliar o impacto da escolha de cada uma das operações candidatas envolvidas no JSF.

Foram testadas LRC com diferentes quantidades de elementos, sempre com igual probabilidade de escolha entre as operações candidatas incluídas na LRC. Os melhores resultados foram obtidos com uma LRC com 2 elementos. Com esse ajuste, a operação da primeira posição da LRC é a aquela para a qual a regra indica a maior prioridade, ao passo

que a operação da segunda posição da LRC é aquela para a qual a regra indica a segunda prioridade mais elevada. Então foram avaliadas maneiras de variar as probabilidades de escolha de uma das operações da LRC. Constatou-se que melhores resultados foram obtidos quando a probabilidade de escolha da operação da primeira posição da LRC aumentou. Isso foi feito através de uma função de tendência, ou *bias function*, conforme Resende e Ribeiro (2010). Foram testadas diferentes probabilidades de escolha das operações da LRC, sendo que nas maiores dimensões a escolha mais frequente da operação da primeira posição permitiu melhores resultados. Os testes indicaram que, independentemente do tamanho do problema, um determinado número de operações da segunda posição da LRC deveria ser selecionado. Resultados de experimentos com diversas dimensões de problemas convergiram para um mesmo valor, indicando que a cada geração de solução cerca de 6 operações da segunda posição da LRC deveriam ser alocadas. Essa convergência para um único valor provavelmente é decorrente do aumento da dificuldade de resolução à medida que os problemas crescem. Em problemas pequenos, escolher 6 operações para as quais as regras não indicaram a maior prioridade gera consideráveis mudanças em relação à geração puramente gulosa. Basta mencionar que, por exemplo, numa instância 5×5 essa quantidade representa cerca de 30% das operações. Contudo, como o problema é pequeno, a fase de melhoria da meta-heurística consegue explorar boa parte da região factível e, assim, alcançar bons resultados. Por outro lado, com problemas de grande porte essa mesma quantidade de operações menos prioritárias é percentualmente menos significativa. Em uma instância de dimensão 100×10 , por exemplo, 6 operações equivalem a somente cerca de 0,8% do total. Tal valor para grandes problemas é coerente, pois limitar a escolha de operações não prioritárias a uma menor porcentagem é necessário. Caso contrário, a fase de melhoria da meta-heurística não consegue superar a deterioração causada pela admissão de muitas operações não prioritárias. Isso ocorre devido ao espaço de soluções ser muito grande. Com essa constatação se definiu que, a cada vez que uma operação precisa ser selecionada, a probabilidade da operação de maior prioridade ser sorteada deve ser igual a $[1 - (6/N)]$. Assim, essa probabilidade é baseada no número de operações do problema, o qual está diretamente relacionado ao número de *jobs* e de máquinas. Com a maior probabilidade de escolha da candidata de prioridade mais elevada, os resultados melhoraram consideravelmente e superaram os obtidos com escolhas não tendenciosas.

A segunda proposta envolveu experimentos nos quais a LRC foi elaborada com base na diferença de valores entre a melhor e a pior operação candidata, conforme Resende e

Ribeiro (2010). Para isso a diferença entre os valores das prioridades da operação de menor e de maior prioridade é calculada. Aqui vale reiterar que neste trabalho a operação de maior prioridade é aquela para a qual a regra fornece o menor valor. Também deve ser definido um percentual θ dessa diferença calculada, a qual é utilizada para admitir ou não operações na LRC. Essa admissão é feita do seguinte modo:

$$LRC = \{j \in O^{Candidatas} : Valor_j \leq Valor_{\min} + \theta (Valor_{\max} - Valor_{\min})\} \quad (55)$$

onde $O^{Candidatas}$ é o conjunto das operações candidatas em um dado momento; $Valor_j$ indica o valor fornecido pela regra de prioridade para a operação j ; $Valor_{\min}$ e $Valor_{\max}$ indicam os valores fornecidos pela regra para as operações de $O^{Candidatas}$ com maior e menor prioridade, respectivamente; e θ , com $0 \leq \theta \leq 1$, restringe as candidatas incluídas na LRC. São incluídas operações cuja prioridade seja menor ou igual à soma do valor da operação de maior prioridade, a de menor valor, com uma porcentagem, determinada por θ , da diferença máxima de prioridades. Desse modo, são admitidas na LRC somente as operações cujos valores de prioridade se encontrem dentro de determinada faixa aceitável da diferença máxima entre as operações avaliadas.

Como essa forma de elaborar a LRC é dinâmica, foi preciso ajustar as probabilidades de escolha de cada uma das operações da LRC também dinamicamente. Inicialmente se trabalhou com igual probabilidade de escolha para todas as operações da LRC e, depois, se investigou o efeito de aumentar a probabilidade de sorteio das operações mais prioritárias. Diferentes maneiras de distribuir as probabilidades foram testadas e o uso de uma função exponencial, similar a Bresina (1996), alcançou os melhores resultados. Assim como ocorreu com a LRC de tamanho fixo, se constatou que em problemas de grandes dimensões a probabilidade de que a operação de maior prioridade fosse sorteada deveria ser maior. Assim, a probabilidade de uma operação j , pertencente à LRC, ser sorteada foi fornecida por:

$$Probabilidade(j) = \frac{\frac{1}{Posição(j)^{Expoente}}}{\sum_{j \in LRC} \left[\frac{1}{Posição(j')^{Expoente}} \right]} \quad (56)$$

onde $Posição(j)$ indica a posição da operação j na lista de operações candidatas ordenadas por prioridade decrescente e $Expoente$ é utilizado para calibrar a probabilidade de seleção das operações. Assumindo valores mais altos, $Expoente$ aumenta a probabilidade de escolha da operação de maior prioridade, o que favorece os resultados das instâncias de maior porte. Na implementação aqui descrita, após a investigação de diversos valores, se verificou que o

parâmetro *Expoente* deveria ser ajustado de acordo com as dimensões das instâncias em função do número de operações N . Foram testados os valores $N^{1/2}$, $N^{1/3}$, $N^{1/4}$, $N^{1/5}$ e $N^{1/6}$, sendo que os melhores resultados foram observados com $Expoente=N^{1/4}$. Tal valor permite maior diversidade de soluções em instâncias de menor porte e evita deteriorações excessivas em instâncias de grande porte. Esse ajuste se mostrou coerente com o encontrado para a LRC de tamanho fixo, uma vez que com instâncias pequenas se mostrou necessário trabalhar com maior aleatoriedade, enquanto que com instâncias maiores a aleatoriedade precisou ser mais limitada.

Para o parâmetro θ foram testados os valores 0,01, 0,015, 0,02 e 0,05 e valores variando de 0,1 a 1,0 com passo 0,1. Apesar de as diferenças provocadas pelo uso de diferentes θ não serem muito acentuadas, os resultados indicaram a tendência de que as menores dimensões exigem valores de θ mais elevados. Basta mencionar que a dimensão 5×5 teve os menores atrasos com $\theta=1,0$. Ao mesmo tempo, as instâncias de maior porte tendem a apresentar melhor desempenho com tolerâncias menores. Por exemplo, na dimensão 100×10 , os melhores resultados foram obtidos com $\theta=0,05$. Novamente a provável razão para que em diferentes dimensões o melhor desempenho seja alcançado com diferentes valores de θ deve ser a diferença entre os espaços de soluções. Em problemas com espaços de soluções reduzidos é possível admitir na LRC operações de prioridade inferior, pois a BL é capaz de explorar boa parte de todo o espaço de soluções. Em problemas com grande número de soluções possíveis, a seleção de operações mal avaliadas se torna prejudicial, porque a BL não tem um alcance tão grande sobre toda a região factível. Esses experimentos indicaram que a elaboração da LRC baseada em uma porcentagem, determinada por θ , da diferença máxima entre as candidatas permite resultados superiores aos obtidos com a LRC de tamanho fixo.

Por fim, para a terceira proposta foram feitos testes nos quais a LRC passou a admitir operações cujos valores das prioridades estivessem dentro de determinada tolerância percentual, determinada por φ , da prioridade mais elevada. Esse tipo de forma de elaborar a LRC é chamada de restrição por valor, conforme Feo e Resende (1995). Nesses experimentos a LRC foi definida conforme segue:

$$LRC = \{j \in O^{Candidatas} : Valor_j \leq Valor_{\min} \cdot (1+\varphi)\} \quad (57)$$

onde φ , com $0 \leq \varphi \leq 1$, limita as operações candidatas àquelas cujo valor seja, no máximo, certa porcentagem dada por φ maior que o valor da melhor candidata. Foram testados valores de φ iguais a 0,04, 0,06, 0,08 e φ variando de 0,1 a 1,0 com passo 0,1. Os resultados também indicaram a leve tendência de que as menores dimensões têm melhor desempenho com

valores de φ mais elevados. Na dimensão 5×5 os menores atrasos ocorreram com o maior valor de φ , enquanto que nas dimensões maiores os melhores resultados foram obtidos com valores reduzidos de φ . Os resultados da GRASP com construções baseadas na tolerância φ sobre a melhor candidata foram comparados aos obtidos com a geração baseada em uma porcentagem definida por θ da diferença entre a melhor e a pior candidata. Em quase todas as dimensões a porcentagem baseada em θ apresentou melhores resultados, assim como melhor média geral.

A partir da observação dos resultados das três propostas de elaboração da LRC, se decidiu que a GRASP proposta utilizaria a segunda proposta, baseada na porcentagem dada por θ da diferença entre a melhor e a pior candidata, conforme a Equação (55). O parâmetro θ foi ajustado, segundo os melhores desempenhos observados, em faixas de acordo com o número de operações com: $\theta=1,0$ para $N \leq 100$; $\theta=0,1$ para $100 < N \leq 350$; e $\theta=0,05$ para $N > 350$.

Calibragem do número de iterações

Assim como feito com a meta-heurística ILS, para a determinação do número de iterações da GRASP também foram efetuados experimentos com quantidades elevadas de iterações. Esses experimentos objetivaram permitir observar o ritmo da redução do atraso em função das iterações do método, identificando o momento em que o atraso total se aproxima da estagnação. Foram utilizadas as mesmas 60 instâncias envolvidas no processo equivalente da ILS, assim como o mesmo número de iterações, $100000/n$. Como esperado, se verificou que, normalmente, as iterações iniciais promovem melhorias mais significativas e, à medida que o processo de resolução avança, essas melhorias se tornam mais modestas. Para exemplificar, as curvas de evolução do atraso total das 6 instâncias de dimensão 15×10 utilizadas nesses testes estão presentes no gráfico da Figura 32. Esse processo se estendeu por mais de 6000 iterações, contudo ambos os eixos do gráfico foram restritos para permitir melhor visualização. A evolução da média dessas instâncias é apresentada na cor preta.

Figura 32 - Atraso total em função do número de iterações da GRASP.

Fonte: elaborada pelo autor.

As curvas da Figura 32 mostram que as iterações iniciais são responsáveis pelas maiores reduções do atraso total. Atendo-se à média dos valores se pode observar que o atraso médio inicial, igual a 1896,0, decai expressivamente até a iteração 3, quando atinge o valor 1569,5. Nessa iteração a GRASP alcança 17,22% de redução de atraso em relação à EDD-AAF. Cabe lembrar que a solução da heurística construtiva da GRASP na primeira iteração é determinística. Em seguida, novas melhorias são observadas, de modo que na iteração 14 a redução do atraso chega a 21,47%. O ritmo das melhorias diminui e somente na iteração 42 a redução do atraso atinge 24,21%, já tendo passado por breves períodos de estagnação. Na iteração 157 a melhoria média é igual a 27,34%, enquanto o atraso tem média 1377,7. A partir de então se seguem períodos mais prolongados de estagnação, separados por pequenas melhorias, indicando que o atraso tende à estabilização. De fato as linhas tracejadas horizontais ajudam a notar como quase que a totalidade das melhorias ocorre na primeira metade das iterações representadas no gráfico. Essas linhas indicam o atraso total inicial médio, 1896,0, e o atraso total médio observado após 800 iterações, igual a 1349,0. Nesses experimentos, até a iteração 400 a GRASP conseguiu 27,73% de redução do atraso total em relação à média inicial, enquanto nas demais 400 iterações o método conseguiu reduzir o atraso em menos de 22 unidades. Desse modo, 96,13% da melhoria obtida com 800 iterações ocorreu até a iteração 400, enquanto a outra metade do tempo de resolução, cerca de 28,6 segundos por instância, foi responsável por somente 3,87% da melhoria. Esses valores indicam a necessidade de limitar adequadamente o número de iterações para que, ao mesmo tempo, melhorias expressivas sejam alcançadas e o processamento computacional não seja desperdiçado. Para tanto, experimentos similares foram realizados com todas as dimensões de

problemas. Os resultados, bem como os gráficos gerados, indicaram que o mesmo número de iterações anteriormente encontrado para a ILS também se mostrou adequado para a GRASP, embora esta gaste um pouco menos de tempo por iteração que o método de trajetória. Desse modo, o número de iterações da GRASP foi ajustado pela expressão $NI=60000/(n \cdot m)$.

GRASP Reativa

A GRASP Reativa, proposta por Prais e Ribeiro (2000), é uma versão da meta-heurística que, em vez de utilizar um único θ para a elaboração da LRC, disponibiliza um conjunto $\Theta=\{\theta_1, \dots, \theta_w\}$ de w valores previamente determinados para θ . Essa versão da GRASP utiliza o conhecimento adquirido em iterações anteriores para aumentar a chance de escolha de valores de θ que forneçam bons resultados. A utilização desse conhecimento histórico permite que a meta-heurística, de certo modo, se autoajuste para cada instância, o que melhora a robustez do método e, ao mesmo tempo, facilita sua calibragem.

A cada iteração da GRASP Reativa um dos diferentes valores contidos no conjunto Θ é selecionado. Com isso, a cada iteração, diferentes quantidades de elementos podem ser admitidas na LRC, o que promove a variabilidade das soluções geradas. Esse processo de seleção do valor de θ a ser utilizado em dada iteração é realizado com base em uma distribuição de probabilidades *Probabilidade(l)*, com $l=1, \dots, w$. Essas probabilidades são inicializadas com *Probabilidade(l)*= $1/w$, para $l=1, \dots, w$. Posteriormente, esses valores são atualizados utilizando informações obtidas no decorrer das iterações. Para tanto se considera $\overline{T_l}$ como o valor histórico médio do atraso total obtido pelas soluções construtivas que utilizaram $\theta=\theta_l$. Então, para $l=1, \dots, w$, é calculado:

$$q_l = \left(\frac{f(s^*)}{\overline{T_l}} \right)^\gamma \quad (58)$$

onde $f(s^*)$ indica o atraso total da melhor solução obtida até a última iteração e o expoente γ pode servir para explorar diferentes atenuações das probabilidades. Depois disso, pela normalização dos valores q_l , as probabilidades *Probabilidade(l)*, para $l=1, \dots, w$, são obtidas por:

$$Probabilidade(l) = \frac{q_l}{\sum_{l'=1}^w q_{l'}} \quad (59)$$

Assim como proposto por Prais e Ribeiro (2000), a GRASP Reativa desenvolvida para o JSF trabalhou com um conjunto Θ com 10 elementos. Diferentes valores foram testados para γ e os melhores resultados foram obtidos com $\gamma=10$, mesmo valor sugerido por

Prais e Ribeiro (2000). Para a realização dos experimentos o conjunto Θ foi definido com base nos valores encontrados para o parâmetro θ nas etapas anteriores do processo de desenvolvimento da GRASP. Os conjuntos utilizados foram: $\Theta=\{0,91; 0,92; \dots; 1,0\}$, para $N \leq 100$; $\Theta=\{0,2; 0,4; \dots; 2,0\}$, para $100 < N \leq 350$; e $\Theta=\{0,01; 0,02; \dots; 1,00\}$, para $N > 350$. Sendo assim, os elementos dos conjuntos Θ utilizados com instâncias com 100 ou mais operações estão distribuídos em torno dos θ anteriormente calibrados. Já os valores do conjunto utilizado com instâncias com menos de 100 operações não puderam ser distribuídos em torno do θ anteriormente calibrado, pois o mesmo tem valor igual a 1,0. Então, nesse conjunto, foram utilizados valores de uma faixa próxima a 1,0 com os quais bons resultados foram observados durante a calibragem precedente de θ .

Na GRASP Reativa implementada, inicialmente as probabilidades de escolha de todos os valores de θ são iguais e, à medida que as iterações ocorrem, as mesmas são atualizadas. Como, neste trabalho, o número de iterações varia com o porte das instâncias, se optou por atualizar as probabilidades a cada iteração, e não por atualizá-las a cada determinada quantidade de iterações. Nessa atualização, as probabilidades de escolha de θ ainda não utilizados são calculadas utilizando o valor médio histórico de atraso obtido pelos demais valores de θ utilizados. Foram realizados experimentos nos quais as probabilidades *Probabilidade(l)* eram calculadas utilizando os atrasos totais históricos obtidos pela heurística construtiva com aleatoriedade. Também foram feitos experimentos nos quais essas probabilidades eram calculadas utilizando o atraso total alcançado após a execução da BL. Observou-se melhor desempenho do método quando a atualização dos valores de *Probabilidade(l)* se baseou nos atrasos das soluções da heurística construtiva com aleatoriedade, assim como proposto por Prais e Ribeiro (2000). Esses resultados foram comparados aos obtidos anteriormente pela GRASP clássica. Em 6 das 10 dimensões a GRASP Reativa encontrou melhores médias, assim como melhor média geral. Apesar disso o teste de Friedman obteve valor- $p=0,637$, não indicando haver diferença estatística significativa entre ambas as versões da GRASP. Experimentos adicionais com outros ajustes do número de iterações também resultaram em melhor média de atraso total da GRASP Reativa. Assim, pelo fato da versão reativa ter tido melhores resultados com tempos equivalentes e por ser mais robusta quanto à calibragem, ela foi escolhida. Desse modo a meta-heurística GRASP Reativa passa a ser denominada como GRASP proposta. Por outro lado, a versão anterior do método é denominada GRASP clássica e seus resultados podem ser consultados no Apêndice C.

7.6.2 Resultados da GRASP proposta

Definidos os componentes e os parâmetros da meta-heurística, seus resultados podem ser apresentados e discutidos. A GRASP proposta neste trabalho se constitui da GRASP Reativa e utiliza o algoritmo *Heur_Constr_Aleat* associado à regra de prioridade EDD-AAF e também à BL-INS-AI. Os elementos são admitidos na LRC de acordo com porcentagens da diferença entre as operações com maior e com menor prioridade. Essas porcentagens são selecionadas dentre um conjunto de porcentagens previamente definidas. O critério de parada é a execução de um número de iterações igual a $60000/(n \cdot m)$. Na Tabela 20 são apresentadas as diferenças percentuais das médias da GRASP proposta em relação à EDD-AAF. Essas diferenças estão estratificadas por nível de flexibilidade e por dimensão do problema, assim como os tempos médios de resolução.

Tabela 20 - Melhorias da GRASP em relação à EDD-AAF.

Dimensão ($n \times m$)	Flexibilidade 20%		Flexibilidade 50%		Flexibilidade 80%		Geral	
	%Dif	Tempo (s)	%Dif	Tempo (s)	%Dif	Tempo (s)	%Dif	Tempo (s)
5×5	-11,61	0,501	-60,67	1,060	-70,37	0,910	-34,88	0,824
10×5	-30,61	2,123	-49,87	2,946	-53,47	3,015	-36,64	2,695
10×10	-50,97	4,785	-49,51	7,899	-49,68	8,556	-50,37	7,080
15×10	-45,29	9,283	-47,75	13,123	-55,39	14,284	-47,55	12,230
15×15	-45,85	19,102	-43,80	29,074	-37,73	39,433	-43,91	29,203
30×15	-27,62	66,406	-29,84	103,901	-29,10	148,393	-28,49	106,233
50×5	-6,70	27,986	-7,81	96,055	-10,48	156,218	-8,17	93,419
50×10	-14,63	86,557	-8,73	152,497	-9,62	219,210	-11,82	152,755
50×15	-14,92	163,494	-11,98	320,329	-13,58	476,969	-13,79	320,264
100×10	-5,33	318,168	-3,30	575,831	-3,34	821,083	-4,08	571,694
Média geral	-25,35	69,840	-31,33	130,271	-33,28	188,807	-27,97	129,640

Os percentuais de melhoria presentes na Tabela 20 mostram que a meta-heurística GRASP foi capaz de obter melhorias significativas em todos os subconjuntos de instâncias. Com exceção do subconjunto 5×5 com 20% de flexibilidade, em problemas com até 30 *jobs* a redução do atraso total em relação à EDD-AAF sempre foi superior a 20%, chegando a 70,37% com a dimensão 5×5 e 80% de flexibilidade. Considerando os níveis de flexibilidade separadamente, nas dimensões com 50 *jobs* ou mais as melhorias foram, no mínimo, de 3,30% e, no máximo, iguais a 14,92%.

É interessante notar, na penúltima coluna, que nas 3 maiores dimensões a GRASP obteve melhorias mais expressivas que a ILS, embora seus respectivos tempos de resolução sejam inferiores. Por outro lado, da dimensão 10×10 à dimensão 30×15, a GRASP teve desempenho inferior ao método de trajetória, embora tenha consumido mais tempo de resolução. Uma análise comparativa mais profunda entre as diferentes meta-heurísticas será

apresentada na Seção 7.8, quando as mesmas serão executadas com o mesmo tempo de resolução. De modo geral houve a tendência de melhorias mais expressivas com problemas de menor porte e com mais alta flexibilidade, padrão também constatado com a ILS. Com relação ao tempo computacional se pode considerar que a GRASP proposta teve desempenho satisfatório. Seu tempo médio de resolução por instância foi de 129,640 segundos e na maior dimensão com 80% de flexibilidade consumiu, em média, menos de 14 minutos.

Os resultados da GRASP também foram analisados em cada um dos 4 tipos de data de entrega. Com as datas de entrega mais restritivas, as melhorias alcançadas pela meta-heurística tiveram média de 19,55%. Com as datas de entrega intermediárias essa média atingiu 35,66%. As maiores melhorias foram observadas com as datas mais folgadas, resultando em 52,31% de redução do atraso total. Já com as instâncias com datas de entrega mistas a melhoria média foi igual a 36,18%. Uma melhor avaliação da GRASP, através da comparação de seus resultados com as soluções ótimas e os limitantes fornecidos pelo modelo matemático implementado em CPLEX, pode ser feita através da Tabela 21.

Tabela 21 - Resultados do método exato e da GRASP.

Dimensão ($n \times m$)	CPLEX			GRASP			%Dif
	T_{total}	Tempo (s)	Melhores soluções	T_{total}	Tempo (s)	Melhores soluções	
5×5	120,3*	1,679	60	122,2	0,824	45	1,58
10×5	404,6	1919,324	34	351,5	2,695	43	-13,14
10×10	550,8	2895,763	28	291,4	7,080	40	-47,09
15×10	1248,7	3382,768	8	454,3	12,230	60	-63,62
15×15	—	—	—	633,9	29,203	—	—
30×15	—	—	—	1967,0	106,233	—	—
50×5	—	—	—	15937,8	93,419	—	—
50×10	—	—	—	9913,7	152,755	—	—
50×15	—	—	—	7539,8	320,264	—	—
100×10	—	—	—	62041,2	571,694	—	—
Média parcial (5×5 a 15×10)	581,1	2049,884	—	304,8	5,707	—	-30,57

* Subconjunto no qual todas as soluções ótimas foram encontradas.

Comparando os valores obtidos pela GRASP aos fornecidos pelo CPLEX com 1 hora de processamento se nota que o método heurístico teve desempenho satisfatório. Considerando o subconjunto de menor dimensão, os atrasos da meta-heurística foram somente 1,58% superiores aos valores ótimos. Considerando somente as 15 instâncias desse subconjunto para as quais a GRASP não encontrou a solução ótima, seu atraso foi 14,16% maior que os valores ótimos. No geral a GRASP obteve 78 das 117 soluções ótimas fornecidas pelo CPLEX, sendo: 45 em instâncias com 5 *jobs*, 17 na dimensão 10×5, 8 na dimensão 10×10 e 8 na dimensão 15×10. O atraso médio da GRASP nessas 117 instâncias foi

igual a 228,3, sendo 6,49% superior aos ótimos. Como já havia sido constatado anteriormente com outros métodos de resolução, a partir da dimensão 10×5 os resultados do CPLEX deixam de ser competitivos com os resultados heurísticos. Basta notar que nessa dimensão o atraso da GRASP, com tempo médio de 2,695 segundos, é 13,14% inferior ao do CPLEX e, na dimensão 15×10, o atraso da GRASP é 63,62% menor. Comparações mais detalhadas podem ser feitas com o auxílio da Tabela 22.

Tabela 22 - Resultados do CPLEX e da GRASP por nível de flexibilidade.

Nível de flexibilidade (%)	Dimensão ($n \times m$)	Instâncias	CPLEX		GRASP		%Dif
			T_{total}	Melhores soluções	T_{total}	Melhores soluções	
20	5×5	20	279,4*	20	279,4	20	0,00
	10×5	20	769,3*	20	816,6	3	6,16
	10×10	20	491,6	14	496,5	8	1,00
	15×10	20	1279,5	2	881,2	20	-31,13
50	5×5	20	54,3*	20	57,1	11	5,16
	10×5	20	245,6	7	153,7	20	-37,43
	10×10	20	645,0	7	228,4	16	-64,59
	15×10	20	1269,3	3	280,0	20	-77,94
80	5×5	20	27,3*	20	30,2	14	10,64
	10×5	20	199,1	7	84,1	20	-57,76
	10×10	20	515,8	7	149,3	16	-71,05
	15×10	20	1197,4	3	201,6	20	-83,17

* Subconjunto no qual todas as soluções ótimas foram encontradas.

Os resultados da Tabela 22 mostram que, com as maiores dimensões e mais altas flexibilidades, a resolução exata esbarra em limitações computacionais, enquanto a GRASP mantém seu desempenho. No subconjunto mais desfavorável os atrasos da GRASP ficam 10,64% acima dos atrasos ótimos do CPLEX. Já no subconjunto mais favorável a meta-heurística consegue média 83,17% inferior aos indicados pela resolução exata limitada a 1 hora. Por fim, o gráfico de barras da Figura 33 é utilizado para uma visualização rápida do desempenho da GRASP e do CPLEX em cada instância.

Figura 33 - Melhoria percentual entre CPLEX e GRASP por instância.

Fonte: elaborada pelo autor.

Observando-se a alternância de cores do gráfico se constata que os resultados da GRASP, apesar de melhores nos problemas mais complexos, não tiveram tanta dominância sobre os resultados do CPLEX limitado a 1 hora. Basta notar que por diversas ocasiões os resultados do CPLEX, em vermelho, foram melhores na dimensão 10×10. Em comparação com a ILS, cujo tempo de resolução foi mais baixo nessa dimensão, a GRASP teve desempenho inferior. Já na maior dimensão, 15×10, se nota grande predominância dos resultados da GRASP sobre os resultados do método exato.

7.7 ARTIFICIAL BEE COLONY

Esta seção contém as análises dos resultados alcançados pela meta-heurística *Artificial Bee Colony* (ABC). Nesse método uma população inicial é gerada e os procedimentos *employed bee*, *onlooker bee* e *scout bee*, os quais se inspiram em funções exercidas pelos membros de uma colônia de abelhas, são empregados para obter melhores soluções. Primeiramente os experimentos preliminares realizados para a seleção e o ajuste dos componentes da ABC são descritos. Em seguida é apresentada a análise dos resultados da ABC proposta.

7.7.1 Seleção e ajuste de componentes

A elaboração da ABC proposta exigiu definições a respeito do processo de geração da população inicial, dos mecanismos de melhoria utilizados nas funções *employed bee* e *onlooker bee*, da forma de obtenção de novas soluções pela função *scout bee* e do ajuste de diversos parâmetros. Também foi investigada uma modificação estrutural encontrada na literatura para a ABC. Sob alguns aspectos é possível afirmar que o conhecimento adquirido

no desenvolvimento da ILS e da GRASP facilitou os ajustes da ABC, por exemplo, na calibração das perturbações e na elaboração da LRC. A seguir as etapas do desenvolvimento do método proposto são descritas detalhadamente.

Geração da população inicial

A primeira fase da resolução da ABC é a geração de uma população inicial, a partir da qual as funções *employed bee* e *onlooker bee* buscam melhorias. Para isso é importante que essa população inicial contenha soluções de qualidade. Ao mesmo tempo deve existir certa diversidade, pois isso pode ser fundamental para explorar diferentes áreas da região factível e chegar a soluções de qualidade.

Inicialmente, visando gerar uma população de alta qualidade, cada uma das *NS* soluções foi obtida com uma regra de prioridade diferente, todas funcionando juntamente com o algoritmo construtivo proposto. Nesses experimentos iniciais foi utilizada uma população com 10 soluções por ser esse o tamanho indicado por Tasgetiren *et al.* (2011) para uma aplicação da ABC ao ambiente *flowshop*. Para tanto foram utilizadas as regras de prioridade que apresentaram os melhores desempenhos entre as onze regras determinísticas descritas anteriormente. As regras foram utilizadas segundo a ordem decrescente de seus atrasos médios gerais, conforme segue: EDD-AAF, MDD-AAF, DMA-AF, DMA, MDD, EDD, SCR, PRTTa, SPT, e MOD.

Depois, com o intuito de aumentar a diversidade das soluções iniciais, foram realizados experimentos nos quais as prioridades não eram consideradas. Nesses experimentos, a cada vez que uma máquina se tornava ociosa, o algoritmo construtivo escolhia aleatoriamente uma das operações candidatas para ser alocada. Essa geração completamente aleatória se assemelha à ABC aplicada à otimização contínua da literatura. Contudo esse método de geração disponibilizou soluções iniciais de baixa qualidade. Com isso, apesar de as melhorias posteriormente obtidas, a meta-heurística não foi capaz de alcançar bons resultados, especialmente com problemas de grande porte. Tanto os resultados obtidos com a geração que utiliza as melhores regras quanto os obtidos com a geração completamente aleatória podem ser consultados no Apêndice D.

O uso de uma população inicial com as melhores regras de prioridade, muitas vezes, conduz a várias soluções idênticas. Isso homogeneiza a população e dificulta a obtenção de bons resultados. Por outro lado, a geração completamente aleatória compromete a qualidade da população inicial, o que interfere no restante da resolução. Por tal razão, foi utilizado um método intermediário que incorporou aleatoriedade à melhor heurística construtiva, a

EDD-AAF, de modo similar à geração de soluções da meta-heurística GRASP. Esse método corresponde ao algoritmo *Heur_Constr_Aleat*, introduzido na Seção 6.2, associado à EDD-AAF. Nesse processo uma das soluções da população é obtida deterministicamente, utilizando o algoritmo *Heur_Constr* com a EDD-AAF, para que a qualidade da solução determinística da EDD-AAF seja incorporada à população inicial. Utilizar uma heurística construtiva com aleatoriedade também tem o intuito de aproveitar o bom desempenho da EDD-AAF, a qual conseguiu melhorias superiores a 20% sobre a melhor regra da literatura, com desempenho especialmente bom nos problemas com maiores dimensões. Na sequência são apresentadas as 3 propostas para a construção da LRC na ABC, as quais são equivalentes às propostas apresentadas anteriormente para a GRASP.

Primeiramente foram feitos experimentos nos quais a ABC trabalhou com uma LRC que utilizou restrição por cardinalidade. Foi utilizada uma LRC composta por 2 elementos e uma probabilidade de escolha da primeira operação da LRC igual a $[1-(6/N)]$. Os resultados obtidos superaram os resultados gerados com a utilização das melhores regras em todas as dimensões de instâncias. Em seguida foram realizados experimentos com a LRC definida com base em uma margem θ da diferença de valores entre a melhor e a pior candidata, segundo a Equação (55), a mesma utilizada para a GRASP. As probabilidades de escolha de cada uma das operações da LRC foram definidas de acordo com a Equação (56) com o parâmetro $Expoente=N^{1/4}$. Para o parâmetro θ foi testado o valor 0,05 e valores variando de 0,1 a 1,0 com passo 0,1. Assim como na GRASP, os resultados indicaram a tendência de que as menores dimensões exigem valores de θ mais elevados. Na dimensão 5×5 os menores atrasos foram gerados com $\theta=0,4$, enquanto na maior dimensão os melhores resultados foram obtidos com $\theta=0,05$. Por fim foram feitos testes com a LRC formada por operações cujos valores das prioridades estivessem dentro de certa tolerância φ da prioridade mais elevada, conforme a Equação (57). Para o ajuste de φ foi testado o valor 0,05 e valores variando de 0,1 a 1,0 com passos iguais a 0,1. Foi observada uma leve tendência de que as menores dimensões têm melhor desempenho com valores de φ mais elevados. Na dimensão 5×5 os melhores atrasos foram obtidos com $\varphi=0,8$, enquanto que na dimensão 100×10 os atrasos mais baixos foram alcançados com $\varphi=0,05$. Os resultados da ABC com população gerada com base na tolerância φ e com população gerada com base em uma margem θ foram comparados entre si nas 10 dimensões de instâncias. Em 8 dimensões a porcentagem baseada em θ apresentou melhores resultados, assim como na média geral.

Com esses resultados a forma de elaboração da LRC selecionada para a ABC proposta foi a baseada na porcentagem, determinada por θ , da diferença entre a melhor e a pior candidata, conforme a Equação (55). O parâmetro θ foi ajustado, segundo os melhores desempenhos observados, em faixas de acordo com o número de operações, com $\theta=0,4$ para $N \leq 100$; $\theta=0,1$ para $100 < N \leq 350$; e $\theta=0,05$ para $N > 350$. Os resultados da ABC com o melhor método de elaboração da LRC estão disponíveis no Apêndice C.

Mecanismos das funções *employed bee* e *onlooker bee*

Estabelecida uma população inicial, a fase de melhoria da meta-heurística ABC pode começar. Essa fase se constitui dos procedimentos executados pelas funções *employed bee* e *onlooker bee*, cujo desenvolvimento é detalhado na sequência.

Neste trabalho a primeira definição tomada a respeito das funções *employed bee* e *onlooker bee* se refere à confirmação da BL a ser empregada. Na proposta apresentada no Capítulo 6 tais funções de melhoria primeiramente realizam uma leve perturbação, cuja calibragem será detalhada posteriormente. Depois é aplicado o procedimento Exploração de Vizinhança única baseada em Inserção e Atualizações Internas (EV-INS-AI), o qual equivale à BL-INS-AI restrita somente à exploração da primeira vizinhança. Contudo, apesar do melhor desempenho apresentado pela BL-INS-AI na Seção 7.4, a BL-ARC-AI foi bastante mais rápida. Por isso foram elaboradas duas versões da ABC, uma com o procedimento EV-INS-AI e outra com um procedimento equivalente que explora uma única vizinhança da BL-ARC-AI. Para essas duas versões da ABC foi concedido o mesmo tempo de resolução. A diferença de desempenho entre essas duas versões da meta-heurística ABC foi bastante similar ao observado quando se comparou a ILS proposta, com BL-INS-AI, com a versão da ILS que utilizou a BL-ARC-AI. A ABC com BL-INS-AI conseguiu melhores resultados que a ABC com BL-ARC-AI em todas as dimensões, encontrando atrasos médios mais de 10% inferiores. Apesar da literatura utilizar os mesmos procedimentos de melhoria nas funções *employed bee* e *onlooker bee*, também foram realizados experimentos usando o procedimento EV-INS-AI na *onlooker bee* e seu equivalente da BL-ARC-AI na *employed bee*. Os resultados foram ligeiramente piores aos obtidos com a utilização exclusiva do procedimento EV-INS-AI. Por tais razões esse procedimento foi mantido na meta-heurística, tanto na função *employed bee* quanto na função *onlooker bee*.

Em seguida se investigou as consequências da não utilização de qualquer procedimento similar a uma BL, deixando as melhorias a cargo das leves perturbações realizadas pelas funções *employed bee* e *onlooker bee*. Obviamente essas perturbações não

têm o mesmo direcionamento às melhores soluções que as BLs ou que o procedimento EV-INS-AI. Contudo, sem a exploração de qualquer vizinhança pelo procedimento EV-INS-AI, resta muito mais tempo para as leves perturbações aleatórias. Com isso seria possível que melhorias obtidas ao acaso resultassem em soluções ainda melhores. Desse modo foram realizados experimentos com uma versão da ABC que não utilizou o procedimento EV-INS-AI ou qualquer exploração de vizinhança. Ao contrário, essa versão da meta-heurística contou somente com melhorias ao acaso, promovidas pelas perturbações. Para a execução desse método foi concedido o mesmo tempo de resolução da versão com o procedimento EV-INS-AI. Surpreendentemente o desempenho da versão sem exploração de qualquer vizinhança foi bom e praticamente se igualou ao desempenho da versão que utiliza o procedimento EV-INS-AI. Na comparação direta a ABC sem o procedimento EV-INS-AI chegou a obter médias minimamente melhores que a ABC com EV-INS-AI em diversas dimensões. Contudo as diferenças foram extremamente pequenas com diferença média igual a 0,04%. Então se investigou melhor se seria conveniente substituir um método mais orientado para melhorias, com o uso do procedimento EV-INS-AI, por outro método que fosse completamente dependente do acaso, como a ABC sem EV-INS-AI. Primeiramente o teste de Friedman foi utilizado para verificar se existiam evidências suficientes que indicassem diferença estatística significativa entre a ABC com o procedimento EV-INS-AI e a ABC sem qualquer exploração de vizinhança. O teste resultou valor- $p=0,150$. Considerando um nível de significância igual a 0,05 não é possível descartar a hipótese de que os resultados dos métodos não são significativamente diferentes entre si. Depois disso, novos experimentos foram conduzidos para verificar se o desempenho relativo dessas versões da ABC se mantinha com diferentes tempos de resolução. Esses experimentos indicaram que a ABC com EV-INS-AI parece ser mais robusta que a ABC sem exploração de vizinhanças. Por exemplo, considerando cada dimensão, a diferença percentual máxima entre os atrasos dos métodos nos primeiros testes foi de 0,12% a favor da ABC sem EV-INS-AI. Quando novos testes foram feitos com mudança no tempo de resolução, essa diferença máxima chegou a 3,51% a favor da ABC com EV-INS-AI. Desse modo, o comportamento mais estável da ABC com EV-INS-AI, além do fato de não terem sido constatadas diferenças significativas entre os métodos, fez com que se optasse por manter a utilização do procedimento EV-INS-AI. Os resultados da ABC sem o procedimento EV-INS-AI podem ser consultados no Apêndice D.

Por fim se verificou o desempenho da ABC com a BL-INS-AI no lugar do procedimento EV-INS-AI. Nessa versão da ABC as funções *employed bee* e *onlooker bee* primeiramente realizam uma leve perturbação e, em seguida, aplicam a BL-INS-AI até atingir

um ótimo local. Para essa versão da meta-heurística foi concedido o mesmo tempo limite de execução da versão que utiliza o procedimento EV-INS-AI. Com isso, evidentemente, menos iterações são possíveis. Porém, a cada iteração as melhorias tendem a ser mais expressivas, uma vez que múltiplas vizinhanças são exploradas. A ABC com BL-INS-AI conseguiu médias sutilmente melhores que as da ABC com EV-INS-AI em diversas dimensões. Novamente o teste de Friedman foi utilizado para verificar a existência de diferenças significativas. O teste indicou valor- $p=0,050$. O valor fica no limite para que a hipótese de que os métodos não diferem entre si não seja rejeitada. De fato a diferença a favor da ABC com BL-INS-AI foi bastante pequena, próxima a 0,05%. Então, da mesma forma que foi feito com a ABC sem qualquer exploração de vizinhança, mais experimentos foram realizados para que os métodos fossem melhor comparados. Esses novos testes novamente indicaram um comportamento mais constante da ABC com EV-INS-AI, principalmente ao se variar o tempo de resolução. Por exemplo, trabalhando com tempos de execução mais restritivos, a ABC com EV-INS-AI chegou a obter atraso médio até 3,15% inferior ao da ABC com BL-INS-AI em uma dimensão e, no geral, atraso médio 0,22% inferior. Nesses mesmos testes ficou claro que a ABC com BL-INS-AI perdeu desempenho à medida que as instâncias se tornaram maiores. Nas 5 maiores dimensões a ABC com EV-INS-AI foi melhor e suas margens de vantagem foram maiores que as margens obtidas pela ABC com BL-INS-AI nos primeiros testes. Além disso, um novo teste de Friedman chegou a valor- $p=0,011$. Tal resultado favorece a ABC com EV-INS-AI e é mais expressivo que o obtido a partir dos primeiros experimentos que compararam o uso de EV-INS-AI e da BL-INS-AI. Com isso, e por considerar que utilizar uma BL completa distancia a proposta da meta-heurística original da literatura, se preferiu utilizar a ABC com EV-INS-AI. Os primeiros resultados gerados pela ABC com BL-INS-AI estão disponíveis no Apêndice D.

Após tais estudos se verificou que o primeiro procedimento proposto teve o desempenho mais satisfatório. Assim ficou definido que na ABC proposta o mecanismo utilizado pelas funções *employed bee* e *onlooker bee* seria o procedimento EV-INS-AI, precedido de uma leve perturbação a ser tratada juntamente com a calibragem dos demais parâmetros na sequência do texto.

Calibragem dos parâmetros

Foram realizados experimentos com o intuito de identificar o tamanho ideal da população, definição importante para se alcançar bons resultados em tempo computacional aceitável. Isso ocorre na ABC porque, apesar de terem maiores chances de melhoria,

populações mais numerosas exigem maior processamento computacional quando o número de iterações é mantido. Contudo, o tempo computacional não pode crescer indefinidamente, pois o método pode se tornar inaplicável, além de dificultar comparações com outras técnicas de resolução. Ao mesmo tempo, limitar o tempo de execução ao se trabalhar com populações muito numerosas pode impossibilitar a exploração adequada das soluções existentes, o que pode prejudicar os resultados.

Visando encontrar o tamanho ideal, testes foram efetuados com populações compostas por 4, 6, 8, 10 e 12 elementos, sendo que para todos esses casos foi concedido o mesmo tempo computacional. Primeiramente se observou uma tendência de melhoria à medida que maiores populações eram utilizadas. Porém, em dado momento essa tendência se inverteu e aumentar ainda mais o número de soluções da população passou a causar perda de desempenho. A tendência de melhorias ocorreu até a população constituída por 10 soluções, sendo que com populações maiores os resultados pioraram. Além disso, quando comparada às demais populações, a resolução com população de 10 soluções permitiu menor média de atraso em 5 das 10 dimensões. Esse melhor desempenho provavelmente indica o ponto de equilíbrio entre a diversificação, promovida por grandes populações, e a intensificação, promovida por populações reduzidas. Assim, o número de soluções da população foi fixado em $NS=10$. Os resultados com esse tamanho de população confirmaram a coerência com o trabalho de Tasgetiren *et al.* (2011), o qual utiliza uma população de igual tamanho na resolução do *flowshop* permutacional. Vale ressaltar que na meta-heurística ABC clássica, bem como neste trabalho, o número de soluções da população é igual ao número de *employed bees* e de *onlooker bees*.

A ABC, conforme descrito no Capítulo 6, utiliza o mesmo procedimento de perturbação da ILS nas suas três funções. Nas funções *employed bee* e *onlooker bee*, a perturbação visa simplesmente efetuar um leve deslocamento aleatório da solução em exploração no espaço de soluções para que uma vizinhança diferente seja investigada. Na função *scout bee*, o objetivo da perturbação é criar uma nova solução para substituir uma solução estagnada por determinado número de iterações. Assim, o número de inserções aleatórias efetuadas pelo procedimento de perturbação evocado pela *scout bee* deve ser maior que o executado nas outras duas funções da ABC. Além disso, a *scout bee* gera uma perturbação sobre a solução incumbente, enquanto *employed bee* e *onlooker bee* perturbam a solução da população que está sendo explorada. Para as funções *employed bee* e *onlooker bee* foram testados os seguintes número de trocas aleatórias de posição e de máquina: $NTP_{EO}=1$ e $NTM_{EO}=0$; $NTP_{EO}=0$ e $NTM_{EO}=1$; $NTP_{EO}=1$ e $NTM_{EO}=1$; e $NTP_{EO}=2$ e

$NTM_{EO}=1$. Como esperado, os melhores resultados foram alcançados com as perturbações mais sutis, sendo que a configuração com melhor desempenho utilizou 1 troca de posição e nenhuma troca de máquina. Esses valores representam perturbações mais moderadas inclusive que a perturbação gerada por uma única troca de máquina, pois tal troca implica, necessariamente, também em uma troca de posição. Para a função *scout bee* foram investigadas todas as combinações de NTP_S variando de 1 a 4 e de NTM_S assumindo 0 ou 1. O melhor desempenho foi encontrado com 3 trocas de posição e 1 troca de máquina, ajuste coerente com o encontrado para a ILS. Então, os níveis de perturbação da ABC foram ajustados conforme segue: $NTP_{EO}=1$, $NTM_{EO}=0$, $NTP_S=3$ e $NTM_S=1$.

Outro parâmetro importante da ABC é o limite de iterações consecutivas sem melhoria (LSM), o qual indica por quantas iterações uma solução pode permanecer estagnada antes de ser substituída por outra solução fornecida pela *scout bee*. Foram realizados experimentos nos quais o valor desse parâmetro variou de 1 a 7. Os resultados indicaram que com instâncias de maior porte o método precisa ser mais tolerante com as soluções não melhoradas. Por outro lado, com problemas menores não é vantajoso preservar soluções estagnadas. Para exemplificar, basta mencionar que na dimensão 5×5 o menor atraso total médio foi conseguido com $LSM=1$, significando que uma única iteração sem melhoria é o suficiente para que a solução não melhorada seja descartada. Já na dimensão 100×10 os melhores resultados foram obtidos com $LSM=4$, indicando que o método preserva uma solução que permaneça por até 3 iterações consecutivas sem melhoria. Esses diferentes valores para diferentes dimensões mantêm coerência com o esperado. Em problemas pequenos o melhor é descartar rapidamente soluções estagnadas para que os procedimentos de melhoria investiguem novas regiões e, assim, cubram grande parte da região factível. Em problemas grandes o ideal é insistir mais nas soluções existentes, porque a região factível e as vizinhanças são muito extensas. Com isso, iniciar explorações prematuras e superficiais em diversos pontos não conduzem a bons resultados, fora o fato de que cada exploração de vizinhança exige maior empenho computacional. Desse modo, o limite de iterações consecutivas sem melhoria foi ajustado por faixas em função do número de operações do problema. Os valores utilizados foram $LSM=1$ para $N \leq 200$; $LSM=2$ para $200 < N \leq 350$; $LSM=3$ para $350 < N \leq 600$; e $LSM=4$ para $N > 650$.

Por fim é detalhado o ajuste do número de iterações da ABC, definido de forma idêntica à realizada com ILS e GRASP. O mesmo conjunto de 60 instâncias, com 6 instâncias de cada dimensão, foi utilizado para acompanhar redução do atraso total promovida pela ABC no decorrer de $100000/n$ iterações. Para cada uma das 10 dimensões foram gerados gráficos

do atraso em função das iterações. Assim como foi observado nos experimentos com ILS e GRASP, nas primeiras iterações da ABC as reduções do atraso são mais acentuadas. Depois as melhorias vão decaindo e o atraso tende à estabilização.

De modo geral, a ABC se aproximou da estabilização do atraso com uma quantidade menor de iterações que ILS e GRASP. Contudo, o tempo gasto pela ABC a cada iteração se mostrou bastante mais elevado que os tempos gastos pelas outras duas meta-heurísticas. Assim, avaliando o ponto ideal de interrupção da meta-heurística em cada dimensão, se concluiu que a expressão $NI=30000/(n \cdot m)$ seria a ideal para permitir à ABC aproveitar as melhorias mais expressivas da parte inicial das curvas. Para exemplificar, a Figura 34 traz as curvas dos atrasos médios das 6 instâncias com 15 *jobs* e 10 máquinas utilizadas nesses experimentos, bem como a curva média, no decorrer de 200 iterações da ABC.

Figura 34 - Atraso total em função do número de iterações da ABC.

Fonte: elaborada pelo autor.

Nos experimentos que geraram o gráfico da Figura 34 o atraso médio inicial da ABC, com soluções da EDD-AAF com aleatoriedade, foi de 1815,2. Ao final da primeira iteração o atraso total médio cai para 1640,0 e, ao término da iteração seguinte, esse valor já é igual a 1566,8. Depois disso o atraso total continua sendo reduzido rapidamente, mas a redução mostra tendência de desaceleração. A resolução é interrompida com 200 iterações, após as curvas apresentarem boa estabilidade. Cabe informar que, mesmo com a ILS e a GRASP empregando o dobro de iterações da ABC, o tempo de resolução da meta-heurística populacional proposta é mais elevado. Por esse motivo nas comparações entre os métodos, a serem realizadas na Seção 7.8, o mesmo tempo de execução é concedido para as diferentes meta-heurísticas.

Modified Artificial Bee Colony

A *Modified Artificial Bee Colony* (MABC) foi proposta por Gao e Liu (2012), para aplicação à otimização contínua, como uma nova maneira de organizar e utilizar os componentes da ABC. As modificações propostas têm o intuito de diminuir a velocidade de convergência da meta-heurística e de melhorar seus resultados. Segundo Gao e Liu (2012) isso é possível pelo fato da MABC utilizar uma variável que define a probabilidade de uma solução ser abandonada, caso sua exploração não permita que uma melhoria seja encontrada.

A primeira diferença do método de Gao e Liu (2012) é a forma de gerar a população inicial, uma vez que a ABC parte de soluções aleatórias e a MABC utiliza procedimentos baseados em outras técnicas como *chaotic systems* e *opposition-based learning method*. A segunda diferença é que, enquanto a ABC executa a *employed bee* NS vezes, uma vez sobre cada solução, e a *onlooker bee* também NS vezes, privilegiando as melhores soluções, a MABC utiliza somente uma dessas funções. A cada iteração da MABC essa função de melhoria é aplicada uma única vez sobre cada solução da população. Por isso, neste trabalho se optou por manter na MABC a nomenclatura *employed bee*. Tal função se assemelha mais com a função de melhoria dessa nova meta-heurística do que a *onlooker bee* devido à escolha das soluções exploradas, embora ambas as funções explorem uma dada solução da mesma maneira.

Outra diferença fundamental da MABC em relação à ABC é que, a cada exploração da *employed bee* que não permita melhoria da solução explorada, um sorteio é realizado para se decidir se a solução explorada infrutiferamente deverá ou não ser substituída por outra gerada pela *scout bee*. Desse modo não é utilizado o conceito de iterações consecutivas sem melhoria, mas somente uma probabilidade de abandono de cada solução a partir da qual uma investigação não encontre melhoria.

Assim, o algoritmo MABC recebe quase as mesmas entradas utilizadas pelo algoritmo ABC. A diferença é que, em vez da entrada *LSM*, a MABC utiliza o parâmetro *Prob*, o qual determina a probabilidade de uma solução ser substituída quando a mesma for explorada pela função *employed bee* sem que se obtenha melhoria. Nesse algoritmo, uma variável ($Atr_Tot^{Prévio}$) é utilizada para verificar se uma dada execução da *employed bee* surtiu melhoria e o procedimento auxiliar *Valor_Aleatório*(0,1) retorna um valor randômico do tipo real entre 0 e 1. O algoritmo segue.

Algoritmo MABC (entrada: $Inst_JSF, NS, NI, Prob, NTP_EO, NTP_EO, NTP_S, NTP_S$)

```

1. início;
2.  $Atr\_Tot^{Sol\_Inc} \leftarrow \infty$ ;
3. Gerar_Pop_Ini(NS);
4. para Contador  $\leftarrow 1$  a NI faça:
5.     para  $r \leftarrow 1$  a NS faça:
6.          $Atr\_Tot^{Prévio} \leftarrow Atr\_Tot[r]$ ;
7.         Employed_bee( $r, NTP\_EO, NTP\_EO$ );
8.         se ( $Atr\_Tot[r] = Atr\_Tot^{Prévio}$ ) faça:
9.             se ( $Valor\_Aleatório(0,1) < Prob$ ) faça:
10.                Scout_bee( $r$ );
11.             fim se;
12.         fim se;
13.     fim para;
14. fim para;
15. Devolva ( $Sol\_Inc$ );
16. fim.
```

Na MABC, assim como na ABC, primeiramente uma população inicial é gerada, conforme a linha 3. Em seguida, as NI iterações tem início na linha 4. A cada iteração, as NS soluções da população são exploradas, de acordo com o laço iniciado na linha 5. Para cada solução explorada é feita uma verificação, na linha 8, que identifica se tal solução foi melhorada ou não pela função *employed bee*. Caso não se tenha obtido melhoria, a solução em questão pode ser abandonada, sendo substituída por outra solução gerada pela *scout bee*.

Considerando os bons resultados da MABC para otimização contínua, foram feitos experimentos para verificar se tal método poderia superar a ABC na resolução do JSF. Para garantir uma comparação justa, a MABC trabalhou com o mesmo número de soluções e utilizou a mesma população inicial da ABC. Da mesma forma, a função *employed bee* utilizada em ambos os métodos foi idêntica e o tempo concedido à MABC foi igual ao utilizado pela ABC. Para a calibragem do parâmetro *Prob* foram feitos testes com o mesmo assumindo valores 0,04, 0,06, 0,08, 0,1, 0,2, 0,3, 0,4, 0,5, 0,6, 0,7 e 0,8. Foi constatado que os problemas com menos *jobs* e menos máquinas têm melhores resultados com probabilidades mais altas. Ao mesmo tempo, problemas de maior porte têm melhor desempenho com probabilidades mais baixas. Contudo, mesmo na menor dimensão, os resultados indicaram que a probabilidade não deve ser maior que 0,6, pois as soluções começam a perder qualidade quando valores mais altos são utilizados. Assim, o parâmetro foi ajustado para se adequar às dimensões das instâncias através da equação $Prob=500/(N^{0,7})$.

Com os resultados dos experimentos não se pode afirmar que a MABC de Gao e Liu (2012) adaptada para otimização discreta é capaz de superar a ABC proposta. Em todas as 10 dimensões a ABC conseguiu melhores resultados com atrasos, em média, 2,64% inferiores aos da MABC. A diferença máxima ocorreu na dimensão 15×15, na qual o atraso total médio

da ABC foi 6,66% menor que o da MABC. As quantidades de melhores soluções de cada método deixam a vantagem da ABC ainda mais nítida. A meta-heurística ABC obteve 526 melhores resultados contra apenas 265 da MABC. O teste de Friedman, indicando valor- $p=0,000$, reforça a existência de diferença de desempenho. Uma provável razão para a MABC aplicada ao JSF não ter superado a ABC, como ocorreu em Gao e Liu (2012), pode ser o fato da ABC e da MABC utilizadas para otimização contínua em Gao e Liu (2012) terem populações iniciais diferentes. Com isso não é possível afirmar que a diferença de desempenho se deve à diferente estruturação dos métodos. No presente trabalho o fato da ABC e da MABC partirem da mesma população inicial permite medir a força da fase de melhoria de cada meta-heurística. Outra provável razão para o melhor desempenho da ABC proposta é a tendência de que a função *onlooker bee* foque seus esforços nas circunvizinhanças das soluções de mais alta qualidade. Com isso, boa parte do esforço computacional da ABC é investido nas regiões mais promissoras. Já a MABC, ao contrário, distribui seus esforços igualmente entre as soluções da população, gastando com as piores soluções o mesmo tempo gasto com as melhores. Os resultados alcançados pela MABC aplicada ao JSF estão disponíveis no Apêndice D.

7.7.2 Resultados da ABC proposta

Uma vez selecionados os componentes e ajustados os parâmetros, os resultados obtidos com a ABC podem ser apresentados e discutidos. A ABC proposta neste trabalho gera a população inicial de $NS=10$ soluções utilizando o algoritmo *Heur_Constr_Aleat* com a regra EDD-AAF. A LRC é elaborada com base em porcentagens da diferença entre as operações com maior e com menor prioridade. As funções *employed bee* e *onlooker bee* utilizam $NTP_{EO}=1$ e $NTM_{EO}=0$, além do procedimento EV-INS-AI. O número de *employed bees* e de *onlooker bees* é igual a NS . A função *scout bee* utiliza $NTP_S=3$ e $NTM_S=1$. A meta-heurística é executada por $30000/(n \cdot m)$ iterações.

A Tabela 23 traz os percentuais de melhoria obtidos pela ABC em relação à regra EDD-AAF utilizada de forma determinística, embora a população inicial possa conter soluções melhores por usar a EDD-AAF associada à aleatoriedade. Os percentuais são discriminados pelas dimensões e pelos níveis de flexibilidade.

Tabela 23 - Melhorias da ABC em relação à EDD-AAF.

Dimensão (<i>n</i> × <i>m</i>)	Flexibilidade 20%		Flexibilidade 50%		Flexibilidade 80%		Geral	
	%Dif	Tempo (s)	%Dif	Tempo (s)	%Dif	Tempo (s)	%Dif	Tempo (s)
5×5	-11,61	3,140	-59,50	4,749	-64,03	4,636	-33,43	4,175
10×5	-33,56	8,126	-56,31	11,284	-60,89	11,507	-40,72	10,306
10×10	-57,47	11,214	-59,49	18,624	-63,52	24,428	-59,01	18,089
15×10	-52,32	21,687	-53,18	28,667	-58,05	40,799	-53,49	30,384
15×15	-56,08	34,714	-57,90	55,375	-47,66	82,169	-54,98	57,419
30×15	-34,11	93,374	-27,51	160,350	-30,91	206,322	-31,78	153,349
50×5	-11,04	71,200	-7,71	222,294	-12,17	325,364	-10,43	206,286
50×10	-16,81	136,230	-7,14	294,839	-6,88	443,342	-11,80	291,470
50×15	-16,65	220,152	-8,72	451,183	-10,84	654,846	-13,07	442,060
100×10	-5,34	476,851	-2,70	1093,288	-2,39	1629,522	-3,61	1066,554
Média geral	-29,50	107,669	-34,02	234,065	-35,73	342,293	-31,23	228,009

Com os resultados da Tabela 23 se constata que as melhorias geradas pela ABC em relação à EDD-AAF foram, em geral, expressivas e que têm padrão similar ao observado com as melhorias da ILS. As melhorias foram maiores em instâncias de menor porte e flexibilidade mais elevada. Para confirmar isso basta verificar que, com as instâncias de até 15 *jobs* dos dois níveis de flexibilidade mais altos, as melhorias ficaram acima de 50%, com exceção do subconjunto 15×15 com 80% de flexibilidade. A melhoria máxima constatada foi de 64,03% no subconjunto 5×5 com flexibilidade de 80%.

Também foram feitas análises a respeito do desempenho da ABC nos quatro tipos de data de entrega. O padrão observado com as duas outras meta-heurísticas se manteve. Com datas mais restritivas, a melhoria geral da ABC foi igual a 21,94%. Com datas intermediárias esse valor foi igual a 39,31%. Com as datas menos restritivas a média foi de 54,13%. Por fim, com as datas mistas a melhoria média alcançou 42,32%.

O desempenho da ABC com os maiores problemas, além dos procedimentos de melhoria propostos, se deve à inclusão de aleatoriedade na geração da população inicial. Como nessas instâncias o espaço de soluções é muito grande, as explorações de vizinhanças são percentualmente menos produtivas, conforme constatado nos resultados das BLs na Seção 7.4. Com uma geração que inclui determinado grau de aleatoriedade, as soluções obtidas são melhor distribuídas pela região factível. Ao mesmo tempo, elas mantêm certa qualidade auferida pelo uso da EDD-AAF. Em instâncias pequenas a melhoria obtida com a aleatoriedade da geração não é tão significativa, pois de qualquer modo os procedimentos de melhoria são capazes de varrer boa parte da região factível. As melhorias mais abrangentes da ABC em relação à ILS em instâncias grandes podem ser notadas, por exemplo, na dimensão 100×10 com 80% de flexibilidade, onde o método populacional alcançou 2,39% de redução

do atraso. Essa redução, no entanto, foi inferior à obtida pela GRASP, provavelmente porque a meta-heurística construtiva investe em mais soluções com aleatoriedade. Como existe diferença nos tempos de resolução de ABC, ILS e GRASP, as comparações definitivas entre as meta-heurísticas serão realizadas na Seção 7.8 com ambas sendo executadas com o mesmo tempo computacional.

Um ponto importante a destacar na ABC proposta é a geração da população inicial. O método baseado na melhor regra e que incorpora aleatoriedade foi capaz de gerar melhorias em relação à EDD-AAF determinística. Essas melhorias foram percentualmente mais significativas nas menores dimensões, mas seus impactos finais foram mais notados nas maiores dimensões. Isso porque com problemas pequenos as fases de melhoria da meta-heurística têm facilidade de encontrar melhorias, ao contrário do que ocorre com problemas grandes. Assim, uma parte das melhorias alcançadas pela ABC, especialmente nas maiores dimensões, deve ser atribuída à proposta de geração da população inicial. Isso mantém coerência com a necessidade que as meta-heurísticas têm de possuir diversidade de soluções para alcançar bons resultados. Os atrasos totais médios de todas as soluções das populações de cada dimensão, bem como o atraso médio das melhores soluções de cada população inicial, estão disponíveis no Apêndice D.

Com relação ao tempo computacional é possível notar que, proporcionalmente, a ABC empregou mais tempo na resolução dos menores problemas do que na resolução dos maiores problemas. Basta observar que o tempo médio gasto pela ABC em problemas de dimensão 100×10 foi cerca de 255 vezes maior do que o gasto com os problemas de dimensão 5×5 . Enquanto isso, na dimensão 100×10 a ILS gastou cerca de 885 vezes mais tempo do que na dimensão 5×5 . Com a GRASP o tempo médio das maiores instâncias foi em torno de 693 vezes o tempo médio das menores. De modo geral, apesar de consideravelmente mais elevados que os da ILS e da GRASP, se pode dizer que os tempos da ABC são aceitáveis. Na maior das dimensões o tempo médio ficou próximo a 18 minutos, o que possibilita aplicações práticas. Além disso, os tempos são compatíveis com os tempos de outros trabalhos da literatura que lidam com problemas de porte e complexidade similares, como Yang e Xu (2013b).

O desempenho da ABC proposta também pode ser avaliado através de comparações com os resultados obtidos pela resolução do modelo matemático implementado em CPLEX. Tais resultados são replicados juntamente aos resultados da ABC na Tabela 24.

Tabela 24 - Resultados do método exato e da ABC.

Dimensão ($n \times m$)	CPLEX			ABC			%Dif
	T_{total}	Tempo (s)	Melhores soluções	T_{total}	Tempo (s)	Melhores soluções	
5×5	120,3*	1,679	60	124,9	4,175	43	3,84
10×5	404,6	1919,324	34	328,8	10,306	55	-18,73
10×10	550,8	2895,763	19	240,7	18,089	53	-56,30
15×10	1248,7	3382,768	8	402,8	30,384	60	-67,74
15×15	–	–	–	508,8	57,419	–	–
30×15	–	–	–	1876,4	153,349	–	–
50×5	–	–	–	15545,0	206,286	–	–
50×10	–	–	–	9916,0	291,470	–	–
50×15	–	–	–	7603,3	442,060	–	–
100×10	–	–	–	62348,9	1066,554	–	–
Média parcial (5×5 a 15×10)	581,1	2049,884	–	274,3	15,738	–	-34,73

* Subconjunto no qual todas as soluções ótimas foram encontradas.

Pela Tabela 24 primeiramente se nota que os resultados da ABC na dimensão 5×5 se aproximam dos atrasos das soluções ótimas. Na média, o atraso total da ABC é somente 3,84% superior aos ótimos, sendo que a meta-heurística populacional encontrou 43 soluções ótimas. Nas demais 17 instâncias desse subconjunto o atraso médio da ABC foi igual a 83,1, enquanto os respectivos atrasos ótimos tiveram média igual a 66,8. Nas 3 outras dimensões para as quais o CPLEX foi capaz de encontrar limitantes superiores, o atraso médio da ABC foi melhor. Como anteriormente observado com as demais meta-heurísticas, o desempenho da ABC também melhorou em relação ao CPLEX à medida que os problemas passaram a envolver mais *jobs* e máquinas. Para confirmar isso basta notar que, na última coluna da tabela, as reduções de atraso da ABC em relação ao CPLEX vão de 18,73%, na dimensão 10×5, a 67,74%, na dimensão 15×10. Ainda mais expressivas são as quantidades de melhores soluções alcançadas pela ABC. Em 211 instâncias a ABC encontrou resultado melhor ou igual ao CPLEX limitado a 1 hora de processamento. Ao mesmo tempo, o CPLEX foi melhor ou igual à ABC em 121 ocasiões. Isso significa que, desconsiderando os empates, a ABC foi melhor que o CPLEX em 119 problemas, enquanto o CPLEX foi melhor em 29 das 240 instâncias.

Com relação às soluções ótimas de todas as dimensões, a ABC encontrou 91 dos 117 ótimos indicados pelo CPLEX, sendo: 43 em instâncias com 5 *jobs*, 29 na dimensão 10×5, 11 na dimensão 10×10 e 8 na dimensão com 15 *jobs*. O atraso médio da ABC foi somente 2,24% superior ao atraso médio desses 117 ótimos conhecidos. Considerando somente as 26 instâncias para as quais somente o CPLEX encontrou a solução ótima, o atraso médio da ABC

foi 7,77% superior. Na Tabela 25 as comparações com as soluções ótimas e com os limitantes obtidos pelo CPLEX podem ser feitas separadamente em cada nível de flexibilidade.

Tabela 25 - Resultados do CPLEX e da ABC por nível de flexibilidade.

Nível de flexibilidade (%)	Dimensão ($n \times m$)	Instâncias	CPLEX		ABC		%Dif
			T_{total}	Melhores soluções	T_{total}	Melhores soluções	
20	5×5	20	279,4*	20	279,4	20	0,00
	10×5	20	769,3*	20	781,9	15	1,64
	10×10	20	491,6	11	430,7	15	-12,40
	15×10	20	1279,5	2	768,0	20	-39,97
50	5×5	20	54,3*	20	58,8	12	8,29
	10×5	20	245,6	7	133,9	20	-45,47
	10×10	20	645,0	4	183,3	19	-71,59
	15×10	20	1269,3	3	250,9	20	-80,23
80	5×5	20	27,3*	20	36,6	11	34,31
	10×5	20	199,1	7	70,7	20	-64,49
	10×10	20	515,8	4	108,3	19	-79,01
	15×10	20	1197,4	3	189,6	20	-84,17

* Subconjunto no qual todas as soluções ótimas foram encontradas.

Com a Tabela 25 se verifica que, na comparação com o CPLEX, o desempenho da ABC foi melhor nas instâncias de maior porte e nos níveis de flexibilidade mais altos. Em cada dimensão se nota que o atraso da ABC se distancia do atraso do CPLEX à medida que a flexibilidade aumenta. Também se observa que, dentro de cada nível de flexibilidade, as margens de vantagem da ABC crescem com o aumento do número de *jobs* ou de máquinas. Um destaque é feito na dimensão 10×5 com 20% de flexibilidade, subconjunto no qual o CPLEX encontrou todos os ótimos e a ABC ficou somente 1,64% acima desses valores. Outro destaque ocorre com as instâncias com 15 *jobs*, cujo atraso médio da ABC chegou a ser 84,17% menor que o do CPLEX. No geral, se constata que o atraso médio e o número de melhores soluções do CPLEX somente foram melhores em subconjuntos nos quais tal método obteve todas soluções ótimas. Nos demais subconjuntos, tanto o atraso médio quanto a quantidade de melhores resultados da ABC sempre foram melhores.

Comparações visuais entre os resultados da ABC e do CPLEX podem ser feitas com o auxílio do gráfico da Figura 35. As barras mostram o desempenho de ambos os métodos em cada uma das 240 instâncias utilizadas. Em uma dada instância, a cor vermelha indica melhor resultado do CPLEX e a cor azul indica melhor resultado da ABC.

Figura 35 - Melhoria percentual entre CPLEX e ABC por instância.

Fonte: elaborada pelo autor.

Assim como havia sido observado nas comparações com a ILS e GRASP, o CPLEX consegue se destacar nos problemas com 5 *jobs* e na dimensão 10×5 com 20% de flexibilidade. Isso ocorre porque nessas 80 instâncias menos complexas o CPLEX conseguiu encontrar todas as soluções ótimas. Contudo, mesmo entre tais problemas, a dominância dos resultados do CPLEX sobre os resultados da ABC não é tão expressiva quanto foi na comparação com a ILS. Basta notar que há mais espaços em branco, indicando que a ABC empatou com o CPLEX ao também encontrar a solução ótima. No restante do gráfico, o que equivale a 160 instâncias, a dominância da ABC pode ser notada com clareza. Em toda essa região à direita do gráfico, em apenas 7 problemas o CPLEX superou a ABC. Enquanto isso ocorreu empate em 34 ocasiões e a ABC obteve resultado superior em 119 problemas. No geral, a cor azul predominou, confirmando o bom desempenho da meta-heurística populacional nas maiores dimensões.

7.8 COMPARAÇÕES ENTRE META-HEURÍSTICAS

A seguir é apresentada uma análise comparativa dos resultados alcançados pelas meta-heurísticas ILS, GRASP e ABC. Cada uma delas pertence, respectivamente, a uma dessas três categorias de meta-heurísticas: trajetória, construtiva e populacional. As meta-heurísticas de trajetória, em geral, concentram seus esforços sobre uma única solução armazenada. A partir dessa solução são investigadas modificações que podem levar a melhorias. Meta-heurísticas construtivas, normalmente, despendem considerável parte de seus esforços na geração de soluções completamente novas. Tais soluções, em seguida, podem ser investigadas, por exemplo, com uma BL. Já as meta-heurísticas populacionais, usualmente,

dividem seus esforços sobre um conjunto de soluções. Isso se justifica pela expectativa de que a diversidade da população, dispersa sobre a região factível, possibilite melhorias. As comparações a seguir visam identificar se alguma dessas estratégias se mostra mais adequada para a resolução do problema em estudo.

Para que as meta-heurísticas propostas pudessem ser comparadas de forma justa, foram realizados experimentos nos quais o mesmo tempo computacional foi concedido para cada um dos três métodos. Para cada instância, foram identificados os tempos de resolução das três meta-heurísticas propostas, cada qual com a limitação de iterações determinada durante as calibrações apresentadas nas seções anteriores. Então, para cada instância, foi tomado o maior dos três tempos de resolução, o qual, usualmente, equivale ao tempo da ABC. Os demais ajustes anteriormente realizados com cada meta-heurística foram mantidos. A Tabela 26 fornece, para cada uma das 10 dimensões de problemas, as médias dos limites de tempo estipulados e os atrasos totais médios obtidos por cada método proposto com esses tempos. A melhor média de cada dimensão é destacada em negrito e o número de melhores soluções, incluindo empates, é fornecido entre parênteses.

Tabela 26 - Resultados das meta-heurísticas executadas com o mesmo tempo.

Dimensão ($n \times m$)	T_{total} (Melhores soluções)			Limite de tempo (s)
	ILS	GRASP	ABC	
5x5	125,7 (45)	121,8 (53)	124,9 (45)	4,175
10x5	329,5 (44)	344,7 (32)	328,8 (49)	10,306
10x10	240,0 (46)	284,0 (21)	240,7 (42)	18,089
15x10	393,9 (47)	446,6 (25)	402,8 (44)	30,384
15x15	499,0 (46)	626,4 (25)	508,8 (39)	57,419
30x15	1782,7 (44)	1959,9 (24)	1876,4 (25)	153,349
50x5	15540,2 (29)	15879,6 (11)	15545,0 (20)	206,286
50x10	10011,8 (14)	9860,7 (36)	9916,0 (10)	291,470
50x15	7685,0 (14)	7489,5 (38)	7603,3 (8)	442,060
100x10	63013,3 (5)	61897,0 (47)	62348,9 (8)	1066,554
Média geral (Total)	9962,1 (334)	9891,0 (312)	9889,6 (290)	228,009 (-)

Analisando os valores da Tabela 26 é possível notar a diversidade de desempenho das meta-heurísticas de diferentes categorias com instâncias de diferentes portes, embora não exista grande desequilíbrio entre as médias gerais. Os frequentes destaques em negrito ressaltam o melhor desempenho da ILS em instâncias com até 50 *jobs* e 5 máquinas, conseguindo menor atraso médio em 5 das primeiras 7 dimensões listadas. Do mesmo modo é possível observar o melhor desempenho da GRASP com as 3 maiores dimensões. A GRASP ainda obteve menor média na dimensão 5×5. Enquanto isso a ABC chegou ao menor atraso médio na dimensão 10×5. O melhor desempenho da GRASP na menor dimensão pode estar relacionado ao maior número de iterações em relação às demais dimensões. Isso permitiu investigar mais soluções com aleatoriedade justamente com os problemas com as menores regiões de busca.

É interessante notar que, nas ocasiões em que obteve a menor média de atraso, a ILS abriu margens relativamente confortáveis em relação à GRASP. Na dimensão 15×15, por exemplo, o atraso médio do método de trajetória foi 20,34% inferior ao do método baseado em construção. Por outro lado, suas vantagens sobre os valores médios da ABC foram mais modestas. A maior diferença ocorreu com a dimensão 30×15, na qual a ILS foi 4,99% melhor que o método populacional. Já nas 3 maiores dimensões, o desempenho da GRASP se distanciou mais do desempenho da ILS que do desempenho da ABC. Enquanto, nessas instâncias, a maior margem da GRASP sobre a ILS foi igual a 2,54%, na dimensão 50×15, a maior vantagem da GRASP sobre a ABC foi de somente 1,50%, na mesma dimensão. As grandes diferenças, nas menores e nas maiores dimensões, deixam claro como os desempenhos da ILS e da GRASP foram distintos entre si. Ao mesmo tempo, fica claro como a ABC, apesar de obter menor média em somente 1 dimensão, sempre conseguiu ser mais competitiva em relação ao melhor método de cada dimensão. Basta observar que nas 9 ocasiões em que a ABC não teve a melhor média, ela foi a segunda melhor, nunca sendo a pior das três. Além disso, os atrasos médios da ABC normalmente não apresentaram grandes diferenças em relação ao método de melhor desempenho. Já as meta-heurísticas ILS e GRASP tiveram comportamentos extremos, pois, com exceção da dimensão em que a ABC foi a melhor, cada uma delas sempre ocupou o primeiro ou o terceiro lugar. No caso da GRASP, quando obteve a melhor média, ficou com boa vantagem, especialmente em relação à ILS, mas quando não foi a melhor, ficou bastante distante das concorrentes. Ao contrário, ILS e ABC caminharam mais próximas entre si.

Graças ao comportamento mais equilibrado da ABC em diferentes dimensões, sua média geral igual a 9889,6 foi a melhor dentre as três, mesmo sendo o método com menor quantidade de melhores resultados, 290 de um total de 600 problemas. Em seguida veio a GRASP com atraso médio geral igual a 9891,0 e 312 melhores soluções. A média geral mais alta foi da ILS, 9962,1, apesar de ter sido a meta-heurística com maior quantidade de melhores soluções, 334. É interessante informar que quando os métodos são comparados aos pares, a ABC tem quantidades maiores de melhores soluções. Comparando somente ILS e ABC, a meta-heurística populacional tem 413 melhores soluções contra 381 da meta-heurística de trajetória. Quando comparada somente com a GRASP, a ABC tem 409 melhores soluções contra 347 da meta-heurística baseada em construção de soluções. Tais números favorecem a avaliação da ABC. Entre ILS e GRASP, a ILS alcança 406 melhores resultados contra 346 da concorrente. Para novas análises, os resultados das meta-heurísticas são estratificados por nível de flexibilidade na Tabela 27.

Tabela 27 - Atrasos médios das meta-heurísticas por nível de flexibilidade.

Dimensão ($n \times m$)	Flexibilidade 20%			Flexibilidade 50%			Flexibilidade 80%		
	ILS	GRASP	ABC	ILS	GRASP	ABC	ILS	GRASP	ABC
5x5	279,4	279,4	279,4	58,5	56,6	58,8	39,3	29,3	36,6
10x5	782,2	806,6	781,9	136,9	145,8	133,9	69,4	81,8	70,7
10x10	430,6	485,5	430,7	177,4	220,9	183,3	112,0	145,7	108,3
15x10	783,5	871,8	768,0	220,9	271,5	250,9	177,4	196,5	189,6
15x15	865,1	1075,3	874,5	316,8	427,8	330,5	315,1	376,3	321,4
30x15	2791,9	3212,4	2929,3	1430,9	1447,3	1500,4	1125,5	1220,1	1199,6
50x5	18896,2	19908,3	19033,8	13724,6	13619,1	13708,7	13999,9	14111,6	13892,5
50x10	13562,9	14022,5	13693,4	8887,1	8445,4	8647,5	7585,5	7114,2	7407,3
50x15	10428,0	10655,3	10513,7	6534,0	6139,7	6403,7	6092,9	5673,7	5892,5
100x10	70270,2	69749,5	69888,7	56881,5	55599,4	56075,5	61888,2	60342,2	61082,5
Média geral	11909,0	12106,6	11919,3	8836,8	8637,3	8729,3	9140,5	8929,1	9020,1

Os atrasos da Tabela 27 guardam coerência com os valores da Tabela 26, pois mostram que houve uma tendência de que a ILS obtivesse melhor desempenho com problemas menos complexos. Isso pode ser constatado ao se notar que, com 20% de flexibilidade, a ILS obteve menor atraso médio em 7 das 10 dimensões. Tais instâncias formam o subconjunto com resolução menos complexa, visto que com mais baixa flexibilidade o número de soluções possíveis é menor. A tendência da ILS, em geral, ter obtido melhores resultados que a GRASP nos problemas menos complexos pode indicar uma desvantagem do método baseado em construção. Uma razão pode ser o fato da GRASP se valer somente da incorporação de aleatoriedade e da BL, não possuindo nenhum artifício mais

elaborado direcionado à melhoria das soluções, o que pode ser mais eficiente com regiões de busca não muito extensas.

À medida que a flexibilidade aumentou, a GRASP passou a ter maior destaque. Uma razão para isso pode ser o aumento da dificuldade de efetuar uma BL em instâncias de grande porte, especialmente com alta flexibilidade. Isso ocorre porque com mais *jobs*, máquinas ou mais alta flexibilidade o número de soluções possíveis cresce mais que a vizinhança considerada na BL. Basta notar que o mecanismo de geração de vizinhança investiga, no máximo, uma ordem de N^2 soluções, pois cada operação pode ser hipoteticamente inserida antes ou depois de cada outra operação. Enquanto isso, o problema chega a ter hipoteticamente $n!^m \cdot m^{n \cdot m}$ soluções, lembrando que N é proporcional à relação $n \cdot m$. Desse modo, fica computacionalmente mais custoso efetuar uma busca na vizinhança e, com uma região factível maior, cada investigação de vizinhança tem menor rendimento na redução do atraso. Esse tipo de dificuldade decorrente do aumento do porte dos problemas já havia ficado evidente nos experimentos com as BLs, quando as melhorias obtidas com as maiores instâncias foram consideravelmente menos expressivas que as obtidas com instâncias menores. Esses aspectos podem levar as meta-heurísticas de trajetória à perda de desempenho em problemas de grande porte.

Levando em consideração os fatores anteriormente mencionados, os resultados das três meta-heurísticas sugerem que, para instâncias de grande porte, pode ser vantajoso investir em métodos que exploram melhor a etapa construtiva, em vez de investir todo o trabalho em uma única solução. Essa constatação advém dos bons resultados que a GRASP obteve com as instâncias de maior porte e também com as instâncias de flexibilidades mais altas. Embora utilize a mesma BL que a ILS, a GRASP emprega parte de seus esforços na construção de novas soluções com o uso de aleatoriedade. Com isso, muitas vezes, ela consegue obter soluções de partida com atrasos inferiores aos alcançados pela ILS através de sucessivas execuções de perturbação e BL sobre uma única solução.

Além dos contrastes entre ILS e GRASP, os resultados estratificados por nível de flexibilidade reafirmaram o bom desempenho geral da ABC. Em todos os níveis de flexibilidade, independentemente da diferença entre ILS e GRASP, a ABC foi capaz de acompanhar as melhores médias. Esse comportamento mostra como o método populacional foi capaz de se adaptar a diferentes situações, apresentando bom desempenho tanto com os problemas mais simples quanto com os mais complexos. Além da estratificação por nível de flexibilidade, análises foram realizadas em cada um dos quatro diferentes tipos de data de entrega, contudo não foram observadas diferenças de desempenho significativas entre os

métodos. A Tabela 28 apresenta o atraso médio e o número de melhores soluções de cada meta-heurística com cada tipo de data de entrega.

Tabela 28 - Resultados das meta-heurísticas por tipo de data de entrega.

Multiplicador para data de entrega	T_{total} (Melhores soluções)		
	ILS	GRASP	ABC
0,8	15431,7 (62)	15386,9 (54)	15360,5 (53)
1,0	10125,7 (85)	10030,0 (79)	10058,0 (70)
1,2	6143,3 (103)	6056,2 (109)	6078,0 (98)
CA	8147,7 (84)	8090,9 (70)	8061,8 (69)

Os resultados da Tabela 28, assim como os da Tabela 26, mostram que, em relação às datas de entrega, os desempenhos de ABC e GRASP se mantiveram próximos. Quanto à ILS, sua média por data de entrega foi pior que as médias das outras meta-heurísticas.

Com relação aos atrasos médios gerais apresentados, não se pode deixar de considerar o fato da GRASP ser favorecida pelos seus bons resultados nas maiores instâncias. Tal favorecimento ocorre porque, nessas dimensões, os atrasos têm ordem de grandeza maior que a ordem de grandeza em várias outras dimensões. Isso também explica a média geral mais elevada da ILS. Uma maneira de comparar os desempenhos com menor interferência da diferença de ordem de grandeza dos atrasos é utilizando o RDI de cada método em cada instância. A Tabela 29 traz, por dimensão, o RDI médio de cada meta-heurística.

Tabela 29 - RDIs médios das meta-heurísticas.

Dimensão ($n \times m$)	ILS	GRASP	ABC
5x5	0,240	0,092	0,220
10x5	0,172	0,437	0,133
10x10	0,133	0,629	0,110
15x10	0,103	0,554	0,137
15x15	0,077	0,573	0,120
30x15	0,140	0,556	0,365
50x5	0,373	0,652	0,344
50x10	0,693	0,321	0,509
50x15	0,689	0,268	0,613
100x10	0,887	0,121	0,479
Média geral	0,351	0,420	0,303

Observando-se na Tabela 29 o desempenho dos métodos para cada dimensão é possível notar como o desempenho da ABC foi competitivo. Em todas as dimensões a ABC

apresentou o menor ou o segundo menor RDI médio. Nas dimensões 10×10 e 50×5 , apesar da menor média de atraso ser fornecida pela ILS, a ABC obteve o melhor RDI médio. Com isso a meta-heurística populacional conseguiu o menor RDI médio geral, igual a 0,303. O segundo melhor desempenho foi apresentado pela ILS, com RDI médio geral 0,351. Por fim, a GRASP apresentou o maior valor, 0,420. Essa inversão de melhor desempenho entre ILS e GRASP está diretamente relacionada às grandezas dos atrasos, o que tem menor interferência sobre o RDI médio geral do que sobre o atraso médio geral.

Um gráfico *boxplot* é utilizado para verificar a existência de diferenças entre os quartis dos atrasos obtidos pelas meta-heurísticas. Tal gráfico é apresentado na Figura 36.

Figura 36 - Gráfico *boxplot* com as meta-heurísticas ILS, GRASP e ABC.

Fonte: elaborada pelo autor.

Pelo gráfico da Figura 36 não é possível observar grandes diferenças entre as meta-heurísticas e, por isso, é preciso considerar os valores obtidos. A ILS apresentou mediana igual a 808, primeiro quartil igual a 11 e terceiro quartil igual a 10640,3. A GRASP apresentou mediana igual a 925, primeiro quartil igual a 42,25 e terceiro quartil igual a 10504,3. A ABC apresentou mediana igual a 863, primeiro quartil igual a 14,5 e terceiro quartil igual a 10723,5. Tais valores podem ser reflexo do bom desempenho da ILS com as instâncias de menor porte e, ao mesmo tempo, do desempenho ruim da GRASP com essas mesmas instâncias.

Outra comparação entre as meta-heurísticas é feita por meio de seus perfis de desempenho, delineados na Figura 37. Para cada curva, dado um valor τ , $Probabilidade(a, \tau)$ permite identificar a porcentagem de problemas em que o método a consegue obter uma solução até τ vezes pior que a melhor solução alcançada pelos métodos envolvidos na comparação. Ambos os eixos foram limitados para permitir melhor visualização.

Figura 37 - Perfis de desempenho de ILS, GRASP e ABC.

Fonte: elaborada pelo autor.

Essas curvas evidenciam o equilíbrio entre ABC e ILS, enquanto ambas mantêm certo distanciamento da GRASP. A curva da ILS, em azul, inicia com 55,67% de melhores soluções, enquanto a GRASP começa com 52,00% e a ABC, com 48,33%. À medida que o fator τ aumenta, ILS e ABC alternam a dominância. Com $\tau=1,5$, a ILS atinge valor 0,9683, indicando que em 96,83% dos problemas seu atraso foi, no máximo, 1,5 vezes maior que o atraso da melhor solução. Neste mesmo ponto a ABC alcança uma porcentagem igual a 96,16%. Com $\tau=2,68$ a ABC supera a ILS e alcança $Probabilidade(a, \tau)=1$ com $\tau=39$. Enquanto isso a curva da ILS atinge o valor 1 no eixo vertical com $\tau=136$. Já a GRASP somente alcança valor 1 com $\tau=193$. Esses dados reforçam a tese de que a ABC, embora não obtenha a maior quantidade de melhores soluções, consegue se manter constantemente próximo ao melhor resultado. Por outro lado, a GRASP, apesar de ter excelente desempenho com as maiores instâncias, não é capaz de acompanhar os demais métodos nas outras dimensões. Já a ILS tem um comportamento intermediário, mais próximo da ABC que da GRASP. De fato essa maior proximidade entre os resultados de ILS e ABC é confirmada pelo teste de Friedman. Ao envolver essas duas meta-heurísticas o teste indicou valor- $p=0,112$. Enquanto isso, o teste envolvendo ILS e GRASP resultou em valor- $p=0,005$. Considerando GRASP e ABC, esse mesmo valor- p foi igual a 0,300. Trabalhando-se com um nível de significância igual a 0,05, é possível afirmar que há indícios de que existe diferença estatisticamente significativa entre ILS e GRASP, assim como entre GRASP e ABC. Contudo não é possível afirmar o mesmo entre ILS e ABC.

Nos experimentos realizados se acompanhou a redução do atraso no decorrer do tempo concedido para a execução dos métodos. Para tanto, em cada método, o atraso de cada problema foi coletado a cada 1/10 do tempo de execução. Foi possível notar que o atraso total

evolui de modo parecido nas três meta-heurísticas. Entre as três, a GRASP apresentou uma leve tendência de obter melhorias mais significativas que as demais na parte inicial da resolução para, depois, passar a encontrar melhorias mais discretas.

Também foram feitos testes nos quais a ILS e a ABC foram executadas com os tempos da GRASP, assim como testes nos quais a GRASP e a ABC foram executadas com os tempos da ILS. Com relação aos métodos com os melhores atrasos médios por dimensão, não houve alterações em relação à Tabela 26. Em cada dimensão, mesmo com diferentes tempos de execução, o método com melhor média sempre foi o mesmo. Comparando o desempenho dos três métodos com os tempos de resolução da ILS proposta, a GRASP passou a obter a menor média geral, igual a 9921,1. A ABC obteve atraso médio geral igual a 9942,0 e a ILS forneceu média 10014,3. Na dimensão 10×10 a ILS superou a ABC e passou a obter o menor atraso total médio. Quanto aos demais valores utilizados nas comparações, não houve alterações significativas. Considerando agora os métodos executados com os tempos da GRASP proposta, novamente a GRASP encontrou a menor média geral de atraso, 9925,3. Depois vieram ABC, com média geral igual a 9937,1, e ILS, com média igual a 10024,9. A GRASP também passou a ter o maior número de melhores soluções, 313, contra 292 da ILS e 272 da ABC. Além disso, seu RDI médio geral passou a ser o segundo melhor, igual a 0,409. A ABC obteve o melhor RDI médio geral, igual a 0,330, enquanto a ILS alcançou valor 0,416. Não foram observadas outras alterações significativas em relação aos desempenhos vistos com os tempos da Tabela 26.

A avaliação geral é de que nenhuma das meta-heurísticas exerceu completa dominância sobre as demais. A meta-heurística baseada em construção, GRASP, se mostrou a mais adequada para resolver problemas de grande porte, especialmente dos níveis de flexibilidade mais altos. Contudo, exceção feita à dimensão 5×5, seus resultados nas menores dimensões foram bastante inferiores aos obtidos por ILS e ABC. O método de trajetória, ao contrário, se mostrou mais adequado para resolver instâncias com até 50 *jobs* e 5 máquinas. Porém, o desempenho da ILS nas maiores dimensões ficou consideravelmente aquém dos desempenhos de GRASP e ABC. Assim, devido a ter obtido bons resultados em quase todas as ocasiões e ter conseguido o atraso médio geral mais baixo, a ABC pode ser considerada o método com melhor desempenho. Além disso, tanto a avaliação com RDI quanto a avaliação com perfis de desempenho, indicam que seu desempenho foi melhor que os desempenhos das demais meta-heurísticas implementadas.

7.9 META-HEURÍSTICAS COM ÊNFASE NA FASE CONSTRUTIVA

Devido ao bom desempenho das meta-heurísticas GRASP e ABC, as quais geram várias soluções com aleatoriedade, com instâncias de grande porte, nesta seção são propostas estratégias que intensificam a utilização da heurística construtiva com aleatoriedade. Enquanto a ABC gera um número limitado de soluções para compor a população inicial, a GRASP constrói uma nova solução com aleatoriedade a cada iteração. Diferentemente, a ILS gera uma única solução determinística que serve como ponto de partida para as melhorias. Essas características dos métodos podem ter sido responsáveis pelas sensíveis diferenças de desempenho observadas nas maiores instâncias entre a ILS e as demais meta-heurísticas, ou mesmo entre GRASP e ABC. Sendo assim, uma exploração mais intensa da fase construtiva com aleatoriedade poderia ser capaz de levar a resultados ainda melhores sem aumento do tempo total de resolução.

Outro fator a ser considerado é o contraste entre o crescimento dos tempos de execução da fase construtiva e da BL, base das meta-heurísticas desenvolvidas, à medida que os problemas ficam maiores. Por exemplo, com instâncias 5×5 , tanto a heurística construtiva empregando EDD-AAF quanto a BL-INS-AI consomem, em média, cerca de 1 milissegundo. Já com instâncias 100×10 o tempo médio da construtiva é de 0,028 segundo, enquanto o tempo da BL-INS-AI chega a 9,245 segundos. Além de consumir mais tempo, as melhorias das BLs e das meta-heurísticas nas maiores dimensões não são tão significativas quanto nos menores problemas. Como analisado em seções anteriores, isso provavelmente se deve à diminuição da eficiência da exploração de cada vizinhança à medida que o porte dos problemas aumenta.

A exploração mais intensa da fase construtiva também é fomentada pelos números levantados por Martí; Resende e Ribeiro (2013), os quais indicam um grande crescimento do número de publicações que utilizam métodos *multi-start*. Ainda é mencionado que, em métodos *multi-start*, não há consenso se é mais vantajoso utilizar um procedimento de melhoria complexo, gerando melhorias expressivas, ou um procedimento simples e rápido, permitindo mais iterações e, assim, gerando maior quantidade de soluções de partida.

No presente trabalho os procedimentos de melhoria propostos não são alterados, mas é feita uma investigação que visa encontrar melhores resultados sem aumento do tempo de execução total. Essa investigação analisa os efeitos da transferência de parte dos esforços, ou do tempo computacional, da fase de melhoria para a fase construtiva. Nas subseções a seguir

são fornecidos mais detalhes. Primeiramente os métodos com ênfase construtiva são descritos. Depois seus resultados são apresentados e discutidos.

7.9.1 Implementações e ajustes da ênfase construtiva

Em cada meta-heurística, a ênfase na fase construtiva se deu pela substituição de algumas execuções da fase de melhoria por execuções, equivalentes em tempo, da heurística construtiva com aleatoriedade. A ILS com Ênfase Construtiva (ILS-EC), em vez de partir da solução gerada pela heurística construtiva determinística, utiliza também a heurística construtiva com aleatoriedade para gerar uma série de soluções. A primeira solução é obtida pela heurística determinística e as demais são geradas com aleatoriedade, sendo que em todo esse processo somente a melhor solução gerada é armazenada. Nos experimentos com ênfase construtiva as soluções com aleatoriedade são geradas da mesma forma que na geração apresentada para a GRASP proposta na Seção 7.6. Após a fase multiconstrutiva ser encerrada, a melhor solução obtida se torna o único ponto de partida da fase seguinte. Na fase de melhoria a ILS-EC funciona exatamente como a ILS tradicional, alternando a BL e o procedimento de perturbação, até que o tempo total de resolução estipulado seja alcançado.

A GRASP com Ênfase Construtiva (GRASP-EC) separa a execução da heurística construtiva da execução da BL. Uma vez definido o tempo dedicado à fase multiconstrutiva, é feita uma estimativa de quantas vezes a GRASP-EC poderá executar a BL no tempo restante de resolução. Para tanto os tempos anteriormente obtidos com a execução da GRASP proposta em cada instância são utilizados para calcular o tempo gasto por iteração. Com isso é possível estimar quantas execuções da BL-INS-AI poderão ser realizadas no tempo restante dedicado à fase de melhoria da GRASP-EC, determinando quantas soluções deverão ser armazenadas durante as múltiplas construções. Na fase multiconstrutiva a primeira solução é gerada deterministicamente e são armazenadas as melhores soluções obtidas. Como na GRASP-EC a fase de melhoria corresponde somente à aplicação da BL, sua fase multiconstrutiva não permite o armazenamento de soluções idênticas. Encerrada a etapa multiconstrutiva, tem início a aplicação da BL sobre as soluções armazenadas. Esse processo segue até que todas as soluções sejam exploradas pela BL ou até que o tempo de execução total seja utilizado. Para assegurar que as melhores soluções geradas sejam exploradas, as mesmas são ordenadas segundo seus atrasos e a BL é iniciada a partir da melhor solução construída. Para diminuir o risco da GRASP-EC aplicar a BL sobre todas as soluções armazenadas e ainda assim restar tempo disponível, o número de soluções armazenadas é 10% maior que a estimativa de execuções da BL em sua fase de melhoria.

A ABC com Ênfase Construtiva (ABC-EC), em vez de gerar uma população de *NS* soluções, utiliza o tempo dedicado às múltiplas construções para gerar quantas soluções for possível. Assim como a ABC proposta, a primeira solução gerada é determinística e as demais soluções são geradas envolvendo aleatoriedade. Nesse processo são armazenadas as *NS* melhores soluções, não sendo admitidas repetições. Finalizada a etapa multiconstrutiva, a ABC-EC inicia o processo de melhorias, idêntico ao da ABC proposta, executado até que o tempo limite de resolução seja atingido.

Os experimentos com ênfase construtiva foram realizados com os mesmos limites de tempo de execução utilizados na Seção 7.8 para as comparações entre as meta-heurísticas. Foram realizados testes iniciais para determinar a porcentagem desse tempo que deveria ser utilizada exclusivamente pela fase multiconstrutiva. Para cada uma das meta-heurísticas, foram feitos experimentos com porcentagens de tempo para a etapa multiconstrutiva que variaram de 10% a 90% com passos iguais a 10%. Em todos os casos o restante do tempo de resolução foi dedicado somente à fase de melhoria. Também foram realizados experimentos com 100% do tempo utilizado pela fase de construções, não deixando tempo para a fase de melhoria. Contudo, para nenhum método esse percentual foi o mais vantajoso em qualquer dimensão. A Figura 38 utiliza gradação de cores para ilustrar, separadamente por meta-heurística e para cada dimensão, as porcentagens de tempo dedicadas à fase multiconstrutiva que geraram melhores e piores resultados. As porcentagens de tempo utilizadas estão em cada eixo horizontal. Para cada dimensão, quanto mais próximo da cor verde estiver um elemento, melhor a média de atraso obtida com a porcentagem de tempo correspondente. Do mesmo modo, tons mais próximos à cor vermelha indicam os piores resultados. Resultados intermediários apresentam a cor amarela.

Figura 38 - Diferentes porcentagens de tempo dedicado à ênfase construtiva.

Fonte: elaborada pelo autor.

As diferentes distribuições das cores da Figura 38 mostram que o uso da etapa multiconstrutiva beneficiou as meta-heurísticas de maneiras distintas. Na Figura 38 (a) se nota que a ILS-EC, na maioria das vezes, obteve melhores resultados com 10% do tempo dedicado à etapa multiconstrutiva. Os tons verdes são frequentes à esquerda da imagem. Ao contrário, à medida que a porcentagem de tempo dedicado à construção aumenta, as cores se aproximam do vermelho. A exceção da ILS-EC é a dimensão 100×10 , cuja melhor média foi obtida com 70% do tempo aplicado às construções com aleatoriedade. É necessário ressaltar que, apesar de representarem os extremos, nem sempre a cor vermelha indica uma média muito pior que a cor verde. Isso ocorreu, por exemplo, com a dimensão 5×5 da ILS-EC, onde a pior média foi somente 0,77% mais alta que a melhor.

Na Figura 38 (b) é possível observar com nitidez que a GRASP-EC conseguiu melhor desempenho quando grande parte do tempo, geralmente 60% ou mais, foi utilizada exclusivamente para as construções com aleatoriedade. Esse padrão já poderia ser esperado, uma vez que a GRASP-EC, assim como a GRASP proposta, não possui mecanismos sofisticados de melhoria. Pelo contrário, é a geração com aleatoriedade que permite à meta-heurística obter melhorias mais significativas que as obtidas pela BL isolada. Por isso é coerente que uma exploração mais intensa da fase construtiva conduza a melhores resultados. Durante os experimentos se observou que a GRASP-EC, com exceção da dimensão 5×5 com 10% do tempo para a fase multiconstrutiva, sempre conseguiu melhores resultados que a GRASP proposta executada com o mesmo tempo computacional. Em contraste, em seus piores percentuais de tempo dedicado às construções, ILS-EC e ABC-EC chegaram a médias 7,12% e 11,09% superiores à ILS proposta e à ABC proposta, respectivamente.

Através da Figura 38 (c) se nota que a ABC-EC teve um padrão de desempenho mais parecido com o da ILS-EC do que com o da GRASP-EC. Seus melhores resultados frequentemente ocorreram com 20% do tempo dedicado às multiconstruções. Além disso, é possível notar uma maior tendência de que, com maiores dimensões, passa a ser mais vantajoso aumentar o tempo dedicado às construções. Basta notar que com as 3 maiores dimensões os tons em verde se deslocam à direita da imagem. Em algumas situações se notou que, a partir de determinada porcentagem de tempo dedicado às construções, a geração com aleatoriedade passou a não conseguir gerar mais soluções de qualidade suficiente para substituir outra solução da população. Isso ocorreu, principalmente, com instâncias de pequeno porte.

Uma vez analisados os resultados iniciais, para cada meta-heurística, foram definidas as porcentagens de tempo a serem dedicadas à etapa multiconstrutiva. Para cada método essas porcentagens foram definidas em 3 faixas associadas ao número de operações dos problemas, N , o qual é fortemente relacionado às quantidades de *jobs* e de máquinas. A ILS-EC utilizou as porcentagens: 10%, para $N \leq 360$; 40%, para $360 < N \leq 650$; e 70%, para $N > 650$. A GRASP-EC empregou: 70%, para $N \leq 250$; 80%, para $250 < N \leq 650$; e 90%, para $N > 650$. A ABC-EC utilizou 20%, para $N \leq 450$; 60%, para $450 < N \leq 650$; e 70%, para $N > 650$.

7.9.2 Resultados das meta-heurísticas com ênfase construtiva

A seguir são apresentados e discutidos os resultados obtidos pelas versões das meta-heurísticas propostas com ênfase construtiva. Primeiramente são feitas comparações de cada um desses métodos com seu correspondente sem ênfase construtiva. Depois os resultados dos métodos com ênfase construtiva são comparados entre si. Na Tabela 30 estão as melhorias percentuais obtidas pelo uso da ênfase construtiva em relação aos métodos originalmente propostos. Os valores são apresentados, para cada método, em cada nível de flexibilidade, além da melhoria geral de cada dimensão. Os resultados foram obtidos com todos os métodos, com ou sem ênfase construtiva, sendo executados com os mesmos tempos computacionais utilizados nos experimentos da Seção 7.8.

Tabela 30 - Melhorias obtidas com a ênfase construtiva por nível de flexibilidade.

Dimensão ($n \times m$)	%Dif ILS-EC e ILS				%Dif GRASP-EC e GRASP				%Dif ABC-EC e ABC			
	20%	50%	80%	Geral	20%	50%	80%	Geral	20%	50%	80%	Geral
5x5	0,00	-3,42	-25,45	-3,18	0,00	-1,94	-4,10	-0,63	0,00	-4,76	-24,73	-3,16
10x5	1,33	-4,20	-6,77	-0,01	-1,78	-6,86	-12,90	-3,37	-0,35	-3,44	-4,31	-1,05
10x10	0,70	-5,89	-9,87	-2,57	-2,57	-14,55	-15,41	-7,88	-1,79	-4,12	-6,05	-3,02
15x10	-1,86	0,66	-0,87	-1,24	-4,93	-11,03	-7,08	-6,48	1,08	-11,60	-9,29	-3,18
15x15	-0,80	-4,51	-9,14	-3,34	-3,94	-12,94	-11,72	-7,55	2,84	-3,21	-7,72	-0,69
30x15	-2,96	-6,48	-0,61	-3,41	-8,43	-5,65	-7,79	-7,62	-5,42	-7,45	-6,02	-6,09
50x5	-0,20	-2,61	-1,79	-1,39	-0,82	-1,37	-1,32	-1,13	0,17	-1,97	-0,89	-0,78
50x10	-2,74	-8,92	-8,84	-6,11	-4,14	-4,16	-3,19	-3,92	-4,36	-5,87	-6,73	-5,38
50x15	-4,39	-9,27	-10,12	-7,29	-4,55	-4,98	-4,54	-4,67	-3,91	-8,00	-7,60	-6,01
100x10	-3,05	-3,90	-3,64	-3,50	-2,01	-1,85	-1,42	-1,77	-2,41	-2,60	-2,40	-2,46
Média geral	-1,40	-4,86	-7,71	-3,20	-3,32	-6,53	-6,95	-4,50	-1,41	-5,30	-7,57	-3,18

Considerando que os valores presente na Tabela 30 foram alcançados sem aumento do tempo computacional, se pode afirmar que a utilização da ênfase construtiva permitiu melhorias consideravelmente importantes. Em cada um dos três métodos, a coluna com as melhorias gerais indica que houve ganhos significativos em quase todas as dimensões. De fato, das 600 instâncias utilizadas, a ILS-EC obteve melhoria em relação à ILS proposta em

331 ocasiões e piora em 90 ocasiões. A GRASP-EC chegou a 408 melhorias em relação à sua correspondente sem ênfase construtiva e encontrou piores resultados em somente 15 problemas. A ABC-EC conseguiu 333 melhorias sobre a ABC proposta e em 78 instâncias ficou com desempenho inferior.

Comparando as melhorias obtidas entre si, se nota que a ILS-EC foi a meta-heurística que mais se beneficiou nas instâncias com 50 *jobs* ou mais, justamente com as quais a ILS proposta apresentou o pior desempenho na comparação com a GRASP e com a ABC propostas. Na dimensão 50×15, por exemplo, a melhoria geral foi de 7,29%. Na dimensão 100×10 a ILS-EC conseguiu 3,50% de melhoria sobre a ILS proposta e, com isso, a ênfase construtiva resultou em 5,99% de melhoria sobre a EDD-AAF, enquanto a ILS proposta obteve 2,58%. Assim, com as instâncias com 100 *jobs*, a ILS-EC mais que dobrou as melhorias da ILS proposta sem acréscimo de tempo computacional. Analisando as melhorias separadamente por nível de flexibilidade, em geral, a ILS-EC conseguiu maior proveito das múltiplas construções à medida que a flexibilidade aumentou. Tanto os maiores ganhos da ILS-EC com as maiores dimensões quanto seus maiores ganhos com flexibilidades mais altas, reforçam a conjectura de que, com problemas mais complexos, é vantajoso transferir esforços da fase de melhoria para a fase construtiva.

A GRASP-EC, em contraste com a ILS-EC, foi a meta-heurística que conseguiu as maiores melhorias nas menores instâncias, especificamente de 10×5 a 30×15. A maior melhoria geral ocorreu com a dimensão 10×10, com 7,88% de redução do atraso em relação à GRASP proposta. Essas maiores melhorias da GRASP-EC poderiam ser esperadas, porque as demais meta-heurísticas já haviam conseguido bons resultados nessas instâncias, deixando a GRASP proposta distante delas. Nas 3 maiores dimensões a GRASP-EC foi a que obteve os menores ganhos. Mesmo assim, na maior dimensão, as melhorias com o uso da ênfase construtiva foram 39,34% maiores que as obtidas pela GRASP proposta com o mesmo tempo de execução, ambas em relação à EDD-AAF.

Já a ABC-EC foi a menos beneficiada pela inclusão da fase multiconstrutiva. Sua maior redução de atraso médio ocorreu com a dimensão 30×15 e foi igual a 6,09%. Com 100 *jobs* a melhoria da ABC-EC em relação à EDD-AAF foi 65,82% maior que a obtida pela ABC proposta. Nas dimensões 50×10 e 50×15 essa diferença foi próxima a 40%. Com todas as meta-heurísticas, tomando por base as médias de todas as dimensões, a utilização da ênfase construtiva sempre propiciou melhorias. Isso deixa claro como a transferência de esforços da fase de melhoria para a fase construtiva pode ser proveitosa e, de algum modo, reforça os

argumentos de Martí; Resende e Ribeiro (2013) a favor de métodos com múltiplas construções.

Mais detalhes sobre os ganhos obtidos pela implementação da ênfase construtiva podem ser verificados na Tabela 31. Essa tabela compara o desempenho das meta-heurísticas sem e com ênfase construtiva, todas executadas com os mesmos limites de tempo da Seção 7.8, com os resultados da resolução exata limitada a 1 hora. As diferenças percentuais entre as médias das 4 menores dimensões de problemas são apresentadas da segunda à quinta linhas da tabela. Na sexta linha, as porcentagens indicam o quão distante cada meta-heurística ficou dos 117 ótimos encontrados pelo CPLEX. A sétima linha indica o quão pior foram os atrasos dos métodos propostos nos problemas em que os mesmos não alcançaram as soluções ótimas fornecidas pelo CPLEX. Assim, os valores dessa linha envolvem uma quantidade diferente de problemas para cada método. Essas quantidades equivalem à subtração da quantidade de ótimos obtida pelo CPLEX, igual a 117, pela quantidade de ótimos obtidos por cada método, informação presente na última linha da mesma tabela.

Tabela 31 - Resultados da ênfase construtiva em relação ao CPLEX.

	Informação	ILS	ILS-EC	GRASP	GRASP-EC	ABC	ABC-EC
%Dif Meta-heurística e CPLEX	5×5	4,49	1,16	1,19	0,55	3,84	0,55
	10×5	-18,58	-18,58	-14,81	-17,68	-18,73	-19,59
	10×10	-56,43	-57,55	-48,44	-52,50	-56,30	-57,61
	15×10	-68,45	-68,85	-64,24	-66,55	-67,74	-68,77
	117 ótimos do CPLEX	2,77	2,64	5,32	3,64	2,24	1,45
	Não ótimos heurísticos	7,40	5,75	8,60	6,25	7,78	6,31
Número de ótimos obtidos pela meta-heurística		89	94	84	86	91	102

Embora a utilização da ênfase construtiva seja mais vantajosa com instâncias de grande porte, os valores da Tabela 31 indicam que seu uso também promove ganhos com instâncias menores. No caso da ABC, a dedicação de parte do tempo para as múltiplas construções permitiu que a média de seus atrasos na dimensão 5×5 ficasse somente a 0,55% dos ótimos. Enquanto isso a ABC proposta, com os mesmos tempos de execução, ficou 3,84% acima dos valores ótimos. Considerando somente as 117 instâncias para as quais o CPLEX indicou solução ótima, a ABC-EC obteve atraso médio somente 1,45% superior, ao passo que a ABC proposta apresentou média 2,24% superior. A ILS executada com os tempos de resolução da Seção 7.8 encontrou 89 soluções ótimas. Seu atraso médio nas 28 demais instâncias em que o CPLEX obteve ótimo foi 7,40% pior. Já a ILS-EC alcançou 94 ótimos e seu atraso nos 23 problemas restantes em que o CPLEX obteve ótimo foi 5,75% pior. A ABC-EC foi o método heurístico que encontrou mais soluções ótimas, 102, onze a mais que a

ABC proposta. Além de todas as melhorias observadas, o teste de Friedman também indica haver evidências para se afirmar que a ênfase construtiva tem efeitos sobre os resultados. Tal teste estatístico foi feito envolvendo cada meta-heurística com ênfase construtiva com sua correspondente sem esse recurso. Nos três casos o teste de Friedman resultou em valor- $p=0,000$. Isso indica que, em cada método, a inclusão da fase multiconstrutiva gerou diferenças estatisticamente significativas.

Em seguida as novas versões das meta-heurísticas são comparadas entre si. A Tabela 32 contém as médias dos atrasos e as quantidades de melhores soluções que ILS-EC, GRASP-EC e ABC-EC obtiveram em cada dimensão. Também são explicitadas as médias dos limites de tempo estabelecidos.

Tabela 32 - Resultados das meta-heurísticas com ênfase construtiva.

Dimensão ($n \times m$)	T_{total} (Melhores soluções)			Limite de tempo (s)
	ILS-EC	GRASP-EC	ABC-EC	
5×5	121,7 (53)	121,0 (52)	121,0 (58)	4,175
10×5	329,5 (42)	333,1 (34)	325,4 (47)	10,306
10×10	233,8 (43)	261,6 (23)	233,5 (44)	18,089
15×10	389,0 (45)	417,6 (30)	390,0 (46)	30,384
15×15	482,3 (51)	579,2 (26)	505,3 (33)	57,419
30×15	1722,0 (45)	1810,7 (23)	1762,1 (29)	153,349
50×5	15324,9 (33)	15700,6 (7)	15424,4 (20)	206,286
50×10	9400,0 (24)	9474,3 (18)	9382,1 (19)	291,470
50×15	7124,9 (21)	7140,1 (26)	7146,0 (15)	442,060
100×10	60808,9 (18)	60801,5 (28)	60813,6 (16)	1066,554
Média geral (Total)	9593,7 (375)	9664,0 (267)	9610,3 (327)	228,009 (-)

Os valores médios da Tabela 32 mostram como as meta-heurísticas com ênfase construtiva ficaram mais próximas entre si que as versões propostas originais, especialmente nas maiores dimensões. Nas 3 maiores dimensões, é possível notar com clareza que a ILS-EC conseguiu reduzir a desvantagem que a ILS proposta apresentou em relação às demais meta-heurísticas na Seção 7.8. Tanto que em 2 ocasiões a ILS-EC superou a GRASP-EC e,

também 2 vezes, superou a ABC-EC. A maior proximidade de desempenho da ILS-EC em relação à GRASP-EC e à ABC-EC confirma que as múltiplas construções podem permitir ganhos importantes, em especial, com problemas de grande porte. Por outro lado, a utilização mais intensa de múltiplas construções com aleatoriedade não foi capaz de prover melhorias equivalentes para a GRASP-EC. Basta notar que com dimensões de 10×5 a 50×5 , assim como ocorreu com a GRASP proposta, a GRASP-EC ficou distante das outras meta-heurísticas. Isso poderia ser esperado, uma vez que a GRASP proposta já utilizava múltiplas construções, embora de outro modo. Além disso, ILS e ABC possuem mecanismos de melhoria mais sofisticados que a simples BL da GRASP. Sobre a ABC-EC, seu comportamento acompanhou o da ILS-EC, não havendo agora grande vantagem nas maiores dimensões. No geral, a ILS-EC obteve o menor atraso total, igual a 9593,7, seguida pela ABC-EC, com média 9610,3, e, depois, pela GRASP-EC, cuja média foi 9664,0. As quantidades de melhores soluções seguiram a mesma ordem. Na comparação direta, a ILS-EC conseguiu por mais vezes obter melhores soluções que a GRASP, 457 a 313. Nesse quesito a ILS-EC também apresentou vantagem sobre a ABC-EC, 417 a 393. Já entre ABC-EC e GRASP-EC os números foram favoráveis ao método populacional, 446 a 324.

Como mencionado anteriormente, os resultados dos experimentos nos quais 100% do tempo foi dedicado às múltiplas construções não foram os melhores, em nenhuma dimensão, para nenhum dos métodos com ênfase construtiva. Assim, utilizar todo o tempo para as construções com aleatoriedade não permitiu encontrar nenhum resultado melhor que os apresentados na Tabela 32. Contudo, é interessante relatar que esses experimentos, em algumas ocasiões, geraram resultados melhores que os obtidos com os métodos originais sem ênfase construtiva com o mesmo tempo de execução. Nas 3 maiores dimensões, os resultados desses experimentos foram melhores que as médias da ILS proposta e da ABC proposta, além de superarem a GRASP proposta nas dimensões 50×10 e 100×10 .

Visando melhores análises, para cada instância foram calculados os valores de RDI das três meta-heurísticas. No geral a ILS-EC apresentou o menor valor médio, 0,275. A ILS-EC também obteve os menores RDIs nas dimensões de 10×10 a 50×10 . O segundo menor valor médio geral, igual a 0,308, foi da ABC-EC. O método ficou com os menores RDIs nas 2 menores dimensões. Já a GRASP-EC ficou com RDI médio geral igual a 0,473 e obteve os menores RDIs médios nas 2 maiores dimensões.

O teste de Friedman resultou em valor- $p=0,000$ quando se comparou a GRASP-EC com cada uma das duas outras meta-heurísticas. Envolvendo ILS-EC e ABC-EC tal teste

indicou valor- $p=0,224$. Assim, com um nível de significância igual a 0,05, não se pode afirmar que existe diferença estatisticamente significativa entre os resultados de ILS-EC e ABC-EC. Porém, há indícios de que a GRASP-EC se difere das demais. Também foram identificados os perfis de desempenho dos três métodos, os quais podem ser verificados na Figura 39. Os eixos do gráfico foram restritos, permitindo melhor visualização.

Pela Figura 39 se observa que o distanciamento da curva da GRASP-EC em relação às curvas da ILS-EC e da ABC-EC foi ainda maior do que o anteriormente observado com os métodos correspondentes sem ênfase construtiva. As curvas de ILS-EC e ABC-EC, a exemplo de seus métodos de origem, se alternam. A curva da ABC-EC é a primeira a alcançar valor 1, com $\tau=43$. A segunda curva a alcançar valor 1 é a da GRASP-EC, com $\tau=155$. Já a ILS-EC somente chega ao topo do gráfico com $\tau=226$. Apesar disso, o desempenho da ILS-EC pode ser considerado o melhor entre os que utilizam múltiplas construções, seguida pela ABC-EC e, depois, pela GRASP-EC.

8 EXPERIMENTOS COMPUTACIONAIS COM O JSFMD

Este capítulo descreve os experimentos numéricos realizados com o problema *Job Shop Flexível* com Montagem e Desmontagem (JSFMD), descrito na Seção 3.2. Inicialmente é descrita a geração das instâncias. Em seguida são apresentados os resultados obtidos com o modelo matemático, com as heurísticas construtivas adaptadas ao JSFMD e com as meta-heurísticas. Tanto os experimentos com o modelo matemático, implementados em CPLEX versão 12.2 de 32 bits, quanto os experimentos com os algoritmos, implementados em linguagem C, foram conduzidos em um computador com um processador Intel Core i7 870, com 8 núcleos e 2,93GHz, com 16GB de RAM e Windows 7 de 64 bits.

8.1 GERAÇÃO DE INSTÂNCIAS JSFMD

As instâncias com *jobs* que possuem operações de montagem e de desmontagem foram geradas com base na geração das instâncias do *Job Shop Flexível* (JSF) tradicional, mas com algumas alterações sugeridas em Birgin *et al.* (2013). Antes de descrever essas alterações, alguns conceitos são detalhados com o auxílio da Figura 40. Nessa figura os três tipos de *jobs* presentes nas instâncias geradas são exemplificados utilizando grafos.

Figura 40 - Tipos dos *jobs* das instâncias JSFMD.

Fonte: elaborada pelo autor.

Na Figura 40 o primeiro *job* possui 1 operação de montagem e não possui operação de desmontagem, sendo classificado como um *job* do tipo M. Esse exemplar de *job* do tipo M é composto por três ramos. O primeiro ramo é formado pelas operações 1, 2 e 3; o segundo ramo, pelas operações 4, 5 e 6; e o terceiro, pelas operações 7, 8 e 9. Esses ramos se unem na

operação de montagem 10. O segundo *job*, classificado como do tipo D, possui 1 operação de desmontagem e não possui operação de montagem, apresentando, nesse exemplo, dois ramos paralelos. O terceiro *job* possui ambas as operações de montagem e desmontagem, sendo classificado como do tipo MD. Em cada *job* das instâncias geradas neste trabalho pode haver, no máximo, 1 operação de montagem e 1 operação de desmontagem e os *jobs* são classificados de acordo com os três tipos descritos. Cada *job* pode ter, independente de seu tipo, dois ou três ramos paralelos. A figura ainda indica os estágios dos *jobs* exemplificados. O número de estágios do *job* i , R_i , indica o máximo número de operações contidas em um caminho simples que parta de uma operação inicial e termine em uma operação final. Nas instâncias geradas, todas as ramificações paralelas de um mesmo *job* têm o mesmo número de operações. Assim, todos os caminhos que partem de uma operação inicial e terminem em uma operação final têm o mesmo número de operações. No primeiro *job* da Figura 40, qualquer caminho que parta de uma de suas operações iniciais (1, 4 ou 7) e chegue à sua operação final (11) envolve 5 operações, o que define seu número de estágios. O exemplo de *job* do tipo D possui 7 estágios, o que equivale ao número de vértices envolvidos no caminho simples da operação 1 até a operação 7, ou até a operação 12. Já as operações do *job* do tipo MD da figura estão distribuídas em 6 estágios.

Com tais definições é possível explicar as alterações efetuadas em relação à geração das instâncias JSF. Em cada instância JSFMD foi definido aleatoriamente, para cada *job* i , o número de estágios R_i em $U[\lceil m/2 \rceil, m]$. Em seguida se definiu, também aleatoriamente, a classificação de cada *job* entre os tipos M, D e MD, além do número de ramificações paralelas em $U[2, 3]$. Para cada *job* do tipo M precisou se definir aleatoriamente, em $U[2, R_i]$, em que estágio ficaria a operação de montagem. Para cada *job* do tipo D se definiu aleatoriamente, em $U[1, R_i-1]$, o estágio da operação de desmontagem. Para cada *job* MD se escolheu aleatoriamente o estágio de montagem R_i^{montagem} , em $U[1, R_i-2]$, e o estágio da operação de desmontagem, em $U[R_i^{\text{montagem}}+2, R_i]$. Vale ressaltar que $\lceil m/2 \rceil$ deve ser maior ou igual a 3 para permitir gerar o tipo de *job* MD, o que se garantiu pelo fato da geração utilizar 5 ou mais máquinas. Tendo tais informações o grafo de cada *job* se encontra definido.

Então, em cada instância, o número de máquinas alternativas de cada operação $|M_j|$ é obtido em uma faixa de flexibilidade, como na geração das instâncias JSF. Com a primeira faixa de flexibilidade o valor $|M_j|$ é obtido em $U[\lceil 0,1 \cdot m \rceil, \lceil 0,3 \cdot m \rceil]$. Com a segunda faixa, se utiliza $U[\lceil 0,3 \cdot m \rceil, \lceil 0,7 \cdot m \rceil]$. Já com a terceira faixa de flexibilidade se utiliza $U[\lceil 0,6 \cdot m \rceil, \lceil 1,0 \cdot m \rceil]$. Assim são geradas instâncias JSFMD com 20%, 50% e 80% de

flexibilidade média, respectivamente. Como na geração das instâncias JSF, o tempo de processamento de cada operação j na sua primeira máquina alternativa k' , t_{jk} , é obtido em [1, 99]. Os tempos t_{jk} das demais máquinas alternativas são obtidos em $[t_{jk}, \min(3 \cdot t_{jk}, 99)]$. Para determinar as datas de entrega foi necessário calcular o tempo médio de processamento dos *jobs*. Para isso foi calculado o tempo médio de processamento de cada operação. Em seguida, para cada caminho que comece em uma operação inicial e termine em uma operação final, foi calculado o somatório dos tempos médios de suas operações. O tempo médio de processamento do *job* equivale ao tempo médio de seu caminho com maior tempo médio, uma vez que os *jobs* possuem operações que podem ser processadas paralelamente.

Nessa geração se utilizou somente um tipo de data de entrega porque nos experimentos precedentes esse fator não pareceu interferir no desempenho dos métodos. Foi escolhido o tipo de data de entrega Combinação Aleatória (CA) por ser o mais realista. Assim, a data de entrega de cada *job* i foi determinada pela multiplicação de seu tempo médio de processamento por $\delta = (n/m)^{1/2}$, com $n \geq m$, e por um valor aleatoriamente escolhido entre 0,8; 1,0 e 1,2. Os valores dos componentes das instâncias geradas são indicados na Tabela 33.

Tabela 33 - Valores dos componentes das instâncias JSFMD geradas.

Componente	Valores utilizados
n	5, 10, 15, 30, 50, 100
m	5, 10, 15
R_i	$U[\lceil m/2 \rceil, m]$
$ M_j $	$U[\lceil 0,1 \cdot m \rceil, \lceil 0,3 \cdot m \rceil]$; $U[\lceil 0,3 \cdot m \rceil, \lceil 0,7 \cdot m \rceil]$; $U[\lceil 0,6 \cdot m \rceil, m]$
M_j	$ M_j $ máquinas sorteadas utilizando $U[1, m]$
t_{jk}	$U[1, 99]$, para uma máquina $k' \in M_j$; e $U[t_{jk}, \min(3 \cdot t_{jk}, 99)]$, para as demais máquinas de M_j
δ	Se $n \geq m$, $\delta = (n/m)^{1/2}$, senão $\delta = 1$
d_i	CA

As quantidades de *jobs* e de máquinas, bem como as combinações $n \times m$, são idênticas às utilizadas na geração de instâncias JSF: 5×5 , 10×5 , 10×10 , 15×10 , 15×15 , 30×15 , 50×5 , 50×10 , 50×15 e 100×10 . Para cada combinação foram geradas 5 instâncias, formando subconjuntos de 50 instâncias. Como foram utilizadas 3 faixas de flexibilidade e 1 tipo de data de entrega, foram gerados 3 subconjuntos de 50 instâncias, resultando em 150 instâncias para o JSFMD. Além das mesmas informações constantes em instâncias JSF, cada instância JSFMD apresenta os grafos dos *jobs*, permitindo identificar mais facilmente os tipos dos *jobs* envolvidos. O Apêndice E exibe, como exemplo, os grafos presentes em uma instância do JSFMD. Para ilustrar uma solução do JSFMD, a Figura 41 (a) traz os grafos dos 5 *jobs* de

uma das instâncias geradas de menor porte, enquanto a Figura 41 (b) ilustra uma solução obtida para essa instância. Cada *job* é representado por uma cor e todas as operações são enumeradas, desde a primeira operação do *job* 1 até a última operação do *job* 5. No gráfico de Gantt apresentado essa numeração é indicada entre parênteses.

Figura 41 - *Jobs* e gráfico de Gantt de uma instância JSFMD.

Fonte: elaborada pelo autor.

Na Figura 41 (a) os *jobs* 1, 2 e 3 são do tipo MD, o *job* 4 é do tipo M e o *job* 5 é do tipo D. É possível observar no gráfico de Gantt da Figura 41 (b) que, muitas vezes, os ramos de cada *job* são processados paralelamente. Isso pode ser notado com o *job* 3, em vermelho. Primeiramente a operação de desmontagem 12 é executada na máquina 2. Então o ramo formado pelas operações 15 e 16, ambas alocadas na máquina 1, é executado em paralelo com o ramo das operações 13 e 14, alocadas nas máquinas 4 e 2, respectivamente. Concluídos os ramos, é iniciada a operação de montagem 17 na máquina 5. No *job* 5 os ramos formados pelas operações 26 e 27 e pelas operações 28 e 29 também são processados paralelamente quase todo o tempo.

Apesar da geração de instâncias JSFMD ter utilizado as mesmas combinações $n \times m$ utilizadas na geração das instâncias JSF, numa dada dimensão, o JSFMD apresenta maior dificuldade de resolução por possuir mais operações por *job*. As médias de operações por instância do JSFMD são fornecidas, posteriormente, juntamente com os resultados das heurísticas construtivas. Além da maior quantidade de operações, o JSFMD também apresenta maior dificuldade de resolução, pois o movimento de inserção pode gerar ciclos não somente envolvendo operações de outros *jobs*, mas também outros ramos do mesmo *job*.

8.2 FORMULAÇÃO MATEMÁTICA

Os experimentos com resolução exata aplicada ao problema JSFMD envolveram o modelo matemático adaptado para minimização do atraso total descrito na Seção 3.2. Tal modelo considera explicitamente as restrições de precedência, o que permite resolver os problemas JSFMD. Foram utilizadas as instâncias com *jobs* com ramificações geradas neste trabalho. A Tabela 34 sintetiza os resultados encontrados pelo CPLEX para instâncias do JSFMD após 1 hora de processamento. Esse tempo de processamento equivale ao tempo de relógio e não ao tempo de CPU. Para cada subconjunto de 15 instâncias são exibidas as médias do atraso total e do *gap*, o número total de soluções ótimas obtidas e o tempo médio de resolução em segundos. Nas ocasiões em que o *software* de programação matemática não encontrou a solução ótima, foi utilizado o valor do limitante superior. Cada *gap* é calculado conforme a Equação (48) e o cálculo dos *gaps* médios exclui *gaps* infinitos, casos nos quais o limitante inferior foi igual a zero e o limitante superior foi maior que zero. Estes subconjuntos são identificados pelo símbolo ∞ .

Tabela 34 - Resultados do CPLEX para instâncias JSFMD.

Dimensão ($n \times m$)	Instâncias	T_{total}	<i>gap</i> (%)	Soluções ótimas	Tempo (s)
5×5	15	243,6	9,33	13	616,156
10×5	15	1210,5	99,19 ∞	1	3439,262
10×10	15	1691,4	2805,96 ∞	0	3600,000
Média geral	–	1048,5	971,49	–	2551,806

A Tabela 34 apresenta os resultados das 3 menores dimensões do JSFMD porque, a partir da dimensão 15×10, a resolução exata não foi capaz de encontrar limitantes superiores para todos os problemas do subconjunto, mesmo após 1 hora de processamento. Na dimensão 5×5 o CPLEX encontrou comprovadamente 13 soluções ótimas e na dimensão 10×5, 1 solução ótima. O tempo médio de resolução foi consideravelmente mais alto que o observado com as instâncias do JSF sem operações de montagem e de desmontagem. Com a menor dimensão do JSFMD o CPLEX gastou, em média, 616,156 segundos, sendo que em 2 ocasiões 1 hora não foi suficiente para obter os valores ótimos. Com as instâncias 5×5 do JSF sem ramificações o CPLEX encontrou todos os ótimos com tempo médio igual a 1,679 segundos. Outra diferença é que nos problemas sem ramificações a resolução exata foi capaz de encontrar limitantes superiores para todas as instâncias até o subconjunto 15×10, 1 subconjunto a mais que com as instâncias JSFMD. Tais fatores evidenciam a maior complexidade do JSFMD em relação ao JSF.

8.3 HEURÍSTICAS CONSTRUTIVAS PARA O JSFMD

Nesta seção são descritas as heurísticas construtivas utilizadas para a geração de soluções para o problema JSFMD. Primeiramente são propostas adaptações que permitem que as regras com melhor desempenho com o JSF melhorem seu desempenho com as instâncias do JSFMD. Depois, os resultados dessas regras adaptadas são analisados.

Sincronização dos ramos de um mesmo *job*

Inicialmente, as heurísticas construtivas anteriormente utilizadas para a resolução do JSF também foram utilizadas para gerar soluções para as instâncias do JSFMD. Os melhores resultados foram obtidos pela *Earliest Due Date* com Antevisão, Afinidade e Flexibilidade (EDD-AAF), seguida pela Data de Entrega Modificada com Antevisão, Afinidade e Flexibilidade (DMA-AF) e pela *Modified Due Date* com Antevisão, Afinidade e Flexibilidade (MDD-AAF), nessa ordem. Porém, tais regras determinam a prioridade de cada operação sem levar em consideração a situação das demais operações do mesmo *job*. Isso significa que cada prioridade é determinada independentemente da situação dos demais ramos do mesmo *job*. Ocorre que, por existirem *jobs* com ramos paralelos, pode haver ramos com grandes adiantamentos ou grandes atrasos em relação aos demais ramos do mesmo *job*. Além disso, o atraso de um *job* é definido pelo instante de término de sua última operação. Dessa forma não existe vantagem quando o processamento de um ramo de um *job* termina muito antes de outro ramo do mesmo *job*. Por tal razão, pode não ser vantajoso considerar o valor da prioridade fornecido pela regra sem considerar a situação dos demais ramos. Ao contrário, pode ser conveniente diminuir a prioridade das operações do ramo adiantado, permitindo que operações de outros *jobs* sejam processadas e que o atraso total seja reduzido. Desse modo, este trabalho propõe utilizar uma penalização para diminuir a prioridade de operações de ramos adiantados. A associação dessa penalização a uma regra é exemplificada com a regra DMA-AF adaptada para operações de Montagem e Desmontagem (DMA-AF-MD), determinada para uma operação *j*, conforme segue:

$$\text{DMA-AF-MD}(j) = \text{DMA-AF}(j) \cdot \text{Penalização}(j) \quad (60)$$

sendo que a penalização da operação *j*, pertencente ao *job i*, é obtida de acordo com a equação a seguir:

$$\text{Penalização}(j) = 1 + \left[\frac{\text{Término mínimo do job } i - \text{Término mínimo do ramo de } j}{\text{Término mínimo do job } i} \right] \cdot \beta \quad (61)$$

onde β ($0 \leq \beta \leq 1$) pondera a penalização que a operação j deve receber e, com isso, se garante $Penalização(j) \geq 1$. Caso o ramo ao qual a operação j pertence não esteja adiantado, o numerador da Equação (61) resulta em zero e o valor de $Penalização(j)$ assume 1, não interferindo na prioridade da operação fornecida pela Equação (60). Caso tal ramo esteja adiantado, a operação recebe uma penalização proporcional ao adiantamento, sendo que o parâmetro β delimita essa penalização a uma fração do valor fornecido pela regra de prioridade. Esse artifício evita que um ramo adiantado dispute as máquinas com ramos atrasados do mesmo *job* ou com outros *jobs*. Isso faz com que os diferentes ramos de cada *job* sejam processados com maior sincronismo, avançando simultaneamente e permitindo a redução do atraso total.

Os experimentos com as instâncias JSFMD envolveram as três regras com melhor desempenho nas instâncias tradicionais, sendo elas EDD-AAF, MDD-AAF e DMA-AF. Para determinar o valor do parâmetro de penalização, para cada regra, foram realizados experimentos nos quais β variou de 0,00 a 2,00 com passo 0,05. Em todos os casos, inicialmente, os resultados melhoraram à medida que o valor cresceu. Isso ocorreu até determinado ponto, a partir do qual houve leve tendência de piora. Então foram feitos testes com β variando em torno desse ponto de inflexão com passo 0,01. Desse modo, para a EDD-AAF adaptada para operações de Montagem e Desmontagem (EDD-AAF-MD) se definiu $\beta=0,22$. Para a MDD-AAF adaptada para operações de Montagem e Desmontagem (MDD-AAF-MD) se fixou $\beta=0,5$. Já para a DMA-AF-MD o valor estabelecido foi $\beta=0,51$.

Resultados das regras com sincronização de ramos

Os resultados obtidos com as regras EDD-AAF-MD, MDD-AAF-MD e DMA-AF-MD estão presentes na Tabela 35, na qual cada linha representa 15 instâncias. Para cada dimensão é fornecido o número médio de operações por instância. Para cada método são apresentados os atrasos totais médios, as diferenças percentuais em relação aos respectivos atrasos médios das regras sem penalização e as quantidades de melhores soluções, as quais consideram as três heurísticas com sincronização de ramos.

Tabela 35 - Atrasos médios e melhorias das regras com sincronização de ramos.

Dimensão ($n \times m$)	N	EDD-AAF-MD			MDD-AAF-MD			DMA-AF-MD		
		T_{total}	%Dif	Melh. soluções	T_{total}	%Dif	Melh. soluções	T_{total}	%Dif	Melh. soluções
5×5	31,7	532,2	-2,29	9	540,1	-4,58	7	540,1	-5,28	7
10×5	67,9	1758,3	-1,73	5	1736,8	-4,17	9	1743,5	-3,73	9
10×10	126,3	1632,7	-2,36	9	1626,6	-5,98	6	1623,4	-6,80	6
15×10	182,6	2874,9	-4,71	8	2893,4	-4,67	7	2903,3	-3,73	6
15×15	282,5	3291,6	-7,04	10	3400,2	-5,48	5	3400,2	-5,56	5
30×15	548,4	10933,5	-6,97	7	10932,7	-10,50	7	10908,5	-10,82	7
50×5	334,9	51783,7	-3,40	3	49632,9	-14,04	10	49634,4	-14,11	10
50×10	630,2	39638,7	-5,19	2	38549,7	-12,97	12	38489,4	-13,52	11
50×15	937,2	35732,9	-4,97	2	34398,8	-12,25	13	34349,5	-12,28	12
100×10	1237,3	180603,0	-4,08	5	178204,7	-15,64	7	178280,7	-15,46	9
Média	–	32878,2	-4,27	–	32191,6	-9,03	–	32187,3	-9,13	–
Total	–	–	–	60	–	–	83	–	–	82

Pela Tabela 35 se verifica que em todas as dimensões a penalização dos ramos adiantados permitiu que as regras obtivessem melhorias, sendo que com a DMA-AF-MD essas melhorias ficaram em torno de 9,13% sobre a DMA-AF. A DMA-AF-MD ainda foi a regra que apresentou a menor média de atraso total, 32187,3, e o maior número de melhores soluções, 82 dentre as 150 instâncias. A penalização utilizada na MDD-AAF-MD possibilitou cerca de 9,03% de redução do atraso em relação à MDD-AAF. Já na EDD-AAF-MD a penalização proveu 4,27% de melhoria. O teste de Friedman indica que a sincronização de ramos foi efetiva. Mesmo ao comparar a regra que obteve o menor ganho com a penalização, a EDD-AAF-MD, o teste resultou em valor- $p=0,000$ na comparação com a EDD-AAF. Entre as regras que empregam sincronização, o teste indica haver diferenças estatisticamente significativas entre a EDD-AAF-MD e as demais. Na comparação com a MDD-AAF-MD foi obtido valor- $p=0,018$ e envolvendo a DMA-AF-MD foi encontrado valor- $p=0,041$. Entre DMA-AF-MD e MDD-AAF-MD o teste resultou valor- $p=1,000$, não permitindo afirmar que tais regras se diferem. Com relação ao tempo computacional, todas as regras foram bastante rápidas. A EDD-AAF-MD foi a mais veloz e gastou 3,6 segundos para resolver os 150 problemas. MDD-AAF-MD e DMA-AF-MD consumiram, respectivamente, 3,8 e 3,9 segundos. Com as instâncias mais demoradas, 100×10, as regras consumiram, em média, cerca de 0,1 segundo.

Na comparação com as soluções obtidas pela resolução exata limitada a 1 hora, as regras de prioridade não foram capazes de encontrar, em qualquer ocasião, uma solução ótima. Suas médias foram consideravelmente piores que as médias do CPLEX nas 2 menores dimensões e suas quantidades de melhores soluções foram inferiores. Porém, na dimensão 10×10 as três regras de prioridade conseguiram média de atraso melhor que o CPLEX, mesmo

com tempo médio de cerca de 1 milissegundo por instância, enquanto o CPLEX consumiu, em média, 1 hora por instância. Nessa dimensão cada uma das regras também passou a obter por mais vezes melhores soluções que a resolução exata.

8.4 META-HEURÍSTICAS APLICADAS AO JSFMD

Esta seção apresenta os ajustes efetuados e a análise dos resultados obtidos pelas meta-heurísticas *Iterated Local Search* (ILS), *Greedy Randomized Adaptive Search* (GRASP) e *Artificial Bee Colony* (ABC) aplicadas ao problema JSFMD. Diferentemente das regras de prioridade, as meta-heurísticas propostas não necessitaram de adaptações para manter a sincronização dos ramos de um mesmo *job*. Isso ocorre porque cada escolha das meta-heurísticas se baseia no atraso total a ser obtido, enquanto as regras se baseiam em prioridades. Com isso, os movimentos que promovem maior sincronismo e diminuem o atraso total são automaticamente selecionados. Por tal razão, as meta-heurísticas propostas para o JSF são aplicadas diretamente ao JSFMD sem adaptações estruturais. Sendo assim, a seguir são descritos pequenos ajustes realizados para a resolução do JSFMD. Depois os resultados obtidos são apresentados e discutidos.

Seleção da heurística construtiva e calibragem do número de iterações

Experimentos iniciais foram realizados para identificar a melhor regra de prioridade a ser utilizada para fornecer soluções iniciais para as meta-heurísticas aplicadas ao JSFMD. Pelos resultados observados a DMA-AF-MD se mostrou a mais adequada. Também foram feitos testes para verificar a necessidade de novas calibrações das meta-heurísticas. Para os principais parâmetros, foram investigados valores próximos aos utilizados para as resoluções do JSF. Como não foram observadas diferenças significativas de desempenho, a mesma calibragem utilizada com o JSF foi mantida com o JSFMD, exceto pelo número de iterações.

Como os experimentos com resolução exata indicaram, a resolução dos problemas JSFMD é mais complexa que a resolução do JSF. Por isso foi necessário investigar a quantidade necessária de iterações para que as melhorias das meta-heurísticas pudessem gerar soluções de boa qualidade sem estender demasiadamente o tempo de execução. Durante os experimentos anteriormente realizados com o JSF, foi possível observar que a meta-heurística ABC é, dentre as três meta-heurísticas implementadas, aquela que consome mais tempo de resolução. Por conseguinte a calibragem do número de iterações e, portanto, do tempo de resolução do JSFMD foi baseada nessa meta-heurística.

Foram realizados testes com 30 instâncias, 3 de cada dimensão, 1 de cada nível de flexibilidade, representando 20% do total de 150 instâncias do conjunto JSFMD. Nesses testes

cada problema foi resolvido pela ABC com $60000/(n \cdot m)$ iterações, o que equivale ao dobro de iterações fixadas para a ABC proposta aplicada ao JSF. Adicionalmente foi estabelecido o limite de 2 horas de resolução por instância. Para cada dimensão, foram traçadas as curvas de evolução do atraso em função do número de iterações, assim como realizado no Capítulo 7. Essas curvas indicam que, com o JSFMD, o valor do atraso total se aproxima antes da estabilização, quando comparado com o que ocorre com o JSF. Para ilustrar esse comportamento a Figura 42 exibe as curvas das 3 instâncias da dimensão 100×10 utilizadas para tal calibragem, além do atraso médio, representado na cor preta. Ambos os eixos sofreram limitações para melhor visualização. Linhas tracejadas horizontais indicam o atraso médio inicial e o atraso médio alcançado após a última iteração apresentada.

Figura 42 - Atraso total em função das iterações da ABC em instâncias JSFMD.

Fonte: elaborada pelo autor.

As curvas da Figura 42 foram limitadas a 25 iterações, embora os testes tenham envolvido mais iterações. Com as 3 instâncias a resolução foi interrompida devido ao limite de 2 horas de resolução. Isso permitiu para as instâncias 1, 2 e 3, respectivamente, alcançar 117, 42 e 28 iterações. Os níveis de flexibilidade dessas instâncias são, na ordem, 20%, 50% e 80%, o que mostra como o aumento da flexibilidade eleva o tempo de execução por iteração.

As linhas tracejadas horizontais permitem notar como as melhorias mais significativas ocorrem nas primeiras iterações. Após a iteração 15 as melhorias são menos perceptíveis, e existe uma maior tendência à estagnação. Aliado a isso, é preciso considerar que o tempo médio de resolução dessas 3 instâncias com somente 15 iterações foi próximo a 2400 segundos. Desse modo, para a dimensão 100×10 se definiu o limite de 15 iterações por instância. Com as demais dimensões foram feitos estudos semelhantes e se chegou ao número de iterações determinado por $NI = 15000(n \cdot m)$. Tal valor representa a metade do número de iterações da ABC proposta aplicada ao JSF, mas, devido à maior dificuldade de resolução do

JSFMD, equivale a maiores tempos de execução. Assim, a ABC aplicada ao JSFMD foi executada por $15000(n \cdot m)$ iterações, enquanto a ILS e a GRASP aplicadas ao JSFMD foram executadas com tempos idênticos aos que a ABC consumiu.

Resultados das meta-heurísticas aplicadas ao JSFMD

Os resultados apresentados a seguir foram obtidos com as meta-heurísticas ILS, GRASP e ABC propostas nas Seções 7.5, 7.6 e 7.7, respectivamente. As resoluções tiveram como pontos de partida soluções geradas pela DMA-AF-MD determinística, no caso da ILS, e também com aleatoriedade, nos casos de GRASP e ABC. Os limites de tempo são equivalentes à execução da ABC com $15000/(n \cdot m)$ iterações. Os demais parâmetros foram mantidos conforme as calibrações anteriormente realizadas para cada meta-heurística proposta. O atraso total médio obtido por cada método em cada dimensão está disponível na Tabela 36. Nessa mesma tabela consta, para cada dimensão, a média do limite de tempo utilizado para a resolução de cada instância, o número de instâncias e as quantidades de melhores soluções.

Tabela 36 - Resultados das meta-heurísticas para o JSFMD.

Dimensão ($n \times m$)	Instâncias	T_{total} (Melhores soluções)			Limite de tempo (s)
		ILS	GRASP	ABC	
5×5	15	372,9 (9)	379,2 (6)	366,5 (9)	8,590
10×5	15	1284,2 (6)	1333,9 (0)	1255,9 (9)	27,775
10×10	15	995,5 (7)	1149,3 (0)	986,8 (8)	56,090
15×10	15	1999,9 (12)	2236,3 (0)	2072,3 (3)	86,422
15×15	15	2355,3 (7)	2669,3 (0)	2368,1 (8)	166,563
30×15	15	9712,3 (7)	9948,3 (8)	9820,2 (2)	457,396
50×5	15	45972,1 (4)	46572,5 (2)	45841,5 (9)	475,355
50×10	15	37050,1 (3)	36640,7 (10)	36866,9 (2)	782,592
50×15	15	33296,1 (3)	32971,7 (10)	33174,3 (2)	1266,148
100×10	15	175023,6 (3)	173943,1 (10)	174781,7 (2)	2709,931
Média (Total)	— (150)	30806,2 (61)	30784,4 (46)	30753,4 (54)	603,686 (—)

Os resultados da Tabela 36 mostram que, com os problemas JSFMD, o comportamento relativo entre as meta-heurísticas observado com os problemas JSF não se alterou muito. Primeiramente com relação às instâncias de menor porte, especificamente até a dimensão 50×5 , se nota que a ILS sempre obteve bom desempenho, gerando as melhores médias para os problemas com 15 e 30 *jobs*. Similarmente, a ABC também alcançou bons resultados até a dimensão 50×5 , na qual obteve o menor atraso médio, 45841,5. O método populacional também alcançou as menores médias nas dimensões com 5 e 10 *jobs*. Obtendo os menores atrasos em 4 dimensões, a ABC apresentou um desempenho relativo melhor que o demonstrado com o JSF. A GRASP, por sua vez, sempre apresentou o pior desempenho para as 7 primeiras dimensões apresentadas. Por outro lado, quando são considerados os problemas com 50 *jobs* e 10 máquinas ou maiores, a GRASP sempre obteve os melhores resultados, assim como havia ocorrido com os problemas JSF. Com esses problemas maiores o desempenho da ILS ficou aquém das demais meta-heurísticas, enquanto a ABC apresentou resultados intermediários entre ILS e GRASP. No geral, a ABC novamente obteve o menor atraso médio, 30753,4, sendo seguida pela GRASP e, depois, pela ILS. De modo geral as diferenças percentuais entre os resultados dos métodos foram menores que as observadas com as instâncias JSF. Por exemplo, com instâncias do JSF a média da GRASP chegou a ser 25,53% superior à média obtida pela ILS. Isso ocorreu com a dimensão 15×15 . Com os problemas JSFMD a GRASP ficou, no máximo, a 16,47% da menor média. Isso ocorreu na dimensão 10×10 , cujo menor atraso foi da ABC.

Uma razão para essas melhorias menos expressivas pode ser o fato da infactibilidade estar mais presente no JSFMD do que no JSF. Isso ocorre porque no JSFMD a inserção de uma operação pode gerar ciclos não somente envolvendo operações de outros *jobs*, mas também outros ramos do mesmo *job*. No JSFMD as máquinas também ficam mais sobrecarregadas, uma vez que há mais operações a serem processadas pela mesma quantidade de máquinas. Além disso, há mais movimentos que não geram melhorias, pois antecipam ramos adiantados de um *job* atrasado, mas que consomem tempo de resolução.

Com relação às quantidades de melhores soluções envolvendo os três métodos, a ILS apresentou o melhor desempenho, chegando a melhores resultados em 61 ocasiões. A ABC veio em seguida, com 54 melhores soluções e, então, veio a GRASP, com 46. Porém, quando os métodos são comparados aos pares a ABC supera os demais métodos. Comparando ILS e ABC, o método populacional tem 80 melhores resultados, enquanto o método de trajetória obtém 76. Ao comparar ABC e GRASP, as quantidades de melhores soluções são,

respectivamente, 100 e 55. Entre ILS e GRASP, a meta-heurística de trajetória consegue 97 melhores soluções contra 59 do método construtivo.

A respeito dos tempos de resolução determinados pela ABC, se nota que, apesar de as execuções envolverem a metade de iterações, foi exigido um processamento computacional consideravelmente maior. Enquanto a média geral de tempo de resolução das instâncias JSF foi 228,009 segundos, o tempo médio de resolução de cada problema JSFMD foi de 600,686 segundos. Isso se deve ao fato dos problemas JSFMD, apesar de apresentarem as mesmas dimensões em termos de *jobs* e de máquinas, possuírem quantidades mais elevadas de operações. As instâncias JSFMD possuem, em média, 64,19% mais operações que as instâncias JSF, o que se deve à existência de ramos paralelos em cada *job*. Contudo, dada a complexidade e as dimensões do problema, os tempos podem ser considerados aceitáveis. Os desempenhos obtidos pelas meta-heurísticas em cada nível de flexibilidade e em cada dimensão podem ser consultados na Tabela 37. Nessa tabela são exibidos os percentuais de melhoria em relação aos atrasos médios das soluções iniciais fornecidas pela DMA-AF-MD.

Tabela 37 - Melhorias das meta-heurísticas em relação à DMA-AF-MD.

Dimensão (<i>n</i> × <i>m</i>)	%Dif ILS				%Dif GRASP				%Dif ABC			
	20%	50%	80%	Geral	20%	50%	80%	Geral	20%	50%	80%	Geral
5×5	-18,68	-47,93	-50,99	-30,95	-19,94	-44,53	-44,11	-29,79	-20,60	-51,02	-46,40	-32,14
10×5	-25,82	-26,02	-28,17	-26,34	-24,54	-23,11	-21,08	-23,49	-27,91	-30,50	-25,09	-27,97
10×10	-39,36	-39,48	-36,01	-38,68	-29,37	-28,21	-30,47	-29,20	-39,36	-38,98	-39,29	-39,21
15×10	-34,11	-26,81	-30,55	-31,12	-26,20	-15,73	-26,10	-22,97	-32,76	-20,66	-30,75	-28,62
15×15	-30,85	-27,79	-33,68	-30,73	-19,86	-20,20	-25,37	-21,49	-32,59	-26,67	-30,90	-30,35
30×15	-20,58	-5,33	-3,94	-10,97	-10,77	-7,50	-7,58	-8,80	-16,27	-6,23	-5,46	-9,98
50×5	-7,14	-6,97	-8,05	-7,38	-4,63	-7,07	-7,01	-6,17	-6,01	-7,49	-9,64	-7,64
50×10	-7,90	-1,21	-0,98	-3,74	-7,04	-3,69	-3,06	-4,80	-8,00	-1,81	-1,82	-4,22
50×15	-6,23	-0,80	-1,16	-3,07	-5,23	-3,52	-2,85	-4,01	-6,01	-1,46	-1,98	-3,42
100×10	-3,21	-1,00	-1,16	-1,83	-3,12	-1,96	-2,16	-2,43	-3,22	-1,25	-1,32	-1,96
Média geral	-19,39	-18,34	-19,47	-18,48	-15,07	-15,55	-16,98	-15,32	-19,27	-18,61	-19,27	-18,55

As melhorias percentuais da Tabela 37 mostram que, em geral, houve a tendência de que ganhos mais expressivos fossem obtidos com problemas de menor porte, especialmente com até 15 *jobs*. Com tais instâncias, em algumas ocasiões, é possível notar o crescimento das melhorias à medida que a flexibilidade aumenta. Isso pode ser visto com a ILS na dimensão 5×5. Já com os problemas com 30, 50 e 100 *jobs* foi mais frequente a tendência contrária, com as melhorias diminuindo com o aumento da flexibilidade. Isso pode ser visto com ILS e GRASP na dimensão 50×10. A maior redução do atraso total médio geral ocorreu com a ABC, a qual conseguiu 18,55% de melhoria sobre a DMA-AF-MD. Esse valor é ainda mais significativo se for considerado que ele equivale a quase 3 vezes a melhoria obtida pela

execução isolada da BL-INS-AI, a qual foi igual a 6,25%. Isso indica como os mecanismos de melhoria das meta-heurísticas foram importantes para a resolução do JSFMD.

Comparando as melhorias em cada nível de flexibilidade, se pode notar que a GRASP nunca conseguiu o maior ganho com 20% de flexibilidade. De fato, nesse nível de flexibilidade, os atrasos da ABC foram os menores em 6 dimensões e os atrasos da ILS foram os menores em 5 dimensões, ocorrendo empate na dimensão 10×10. Já com 50% ou 80% de flexibilidade, a GRASP foi a melhor nas 4 dimensões com mais operações: 30×15, 50×10, 50×15 e 100×10. Nas demais dimensões, em ambos os níveis de flexibilidade, ILS e ABC obtiveram 3 melhores médias cada uma.

As comparações entre as meta-heurísticas e os resultados ótimos e limitantes obtidos pelo CPLEX, envolvendo 45 instâncias, podem ser feitas através da Tabela 38. Nessa tabela são apresentados os percentuais que separaram os resultados heurísticos dos fornecidos pelo CPLEX. Esses percentuais são calculados com base nas médias de cada uma das 3 menores dimensões para as quais o CPLEX obteve todos os limitantes após 1 hora de processamento. Na quinta linha são fornecidos os percentuais que separaram o valor da função-objetivo obtido pelas meta-heurísticas dos 14 ótimos. Na sexta linha consta o quanto cada método se distanciou dos ótimos nas ocasiões em que não encontrou a solução ótima. Também são fornecidas as quantidades de soluções ótimas obtidas por cada método heurístico, além da quantidade de vezes que cada um deles alcançou soluções melhores que os limitantes fornecidos pelo *software* de programação matemática.

Tabela 38 - Resultados das meta-heurísticas em relação ao CPLEX com o JSFMD.

Informação		ILS	GRASP	ABC
%Dif Meta-heurística e CPLEX	5×5	53,09	55,67	50,44
	10×5	6,09	10,20	3,75
	10×10	-41,15	-32,05	-41,66
	14 ótimos do CPLEX	42,76	42,76	37,05
	Não ótimos heurísticos	46,23	46,23	40,05
Número de ótimos obtidos pela meta-heurística		1	1	1
Número de vezes que a meta-heurística superou o limitante		18	16	19

Na comparação com resultados ótimos fornecidos pelo CPLEX, o desempenho das meta-heurísticas aplicadas aos problemas JSFMD foi inferior ao observado com o JSF. Cada meta-heurística encontrou somente 1 solução ótima. Os percentuais da segunda linha da Tabela 38 mostram que a melhor meta-heurística obteve atrasos 50,44% superiores aos obtidos pelo CPLEX. Vale lembrar que nessa dimensão a resolução exata conseguiu quase todas as soluções ótimas. Na dimensão 10×5 as meta-heurísticas se aproximaram mais dos

resultados do CPLEX, chegando a ficar somente 3,75% acima dos seus atrasos. Na dimensão 10×10 as meta-heurísticas ILS, GRASP e ABC conseguiram boas margens sobre os resultados do CPLEX, chegando a obter atraso 41,66% menor. Considerando os 14 ótimos do CPLEX, 13 deles da dimensão 5×5, os atrasos obtidos pelos métodos heurísticos foram bastante superiores, sempre acima de 40%. Com relação às melhores soluções, a ABC superou a resolução exata em 19 dos 45 problemas, havendo 1 empate e 25 melhores resultados fornecidos pelo CPLEX. Apesar disso, quando é computado o atraso médio geral das 3 dimensões de problemas, as meta-heurísticas superam a resolução exata. Com esse cálculo o atraso médio do CPLEX foi igual a 1048,5. Em relação a tal valor a ILS consegue 15,67% de melhoria, a GRASP consegue 9,00% de melhoria e a ABC chega a um atraso médio 17,05% menor. Com isso se nota que, mesmo com problemas não muito grandes, a resolução através de meta-heurísticas, como esperado, se mostra vantajosa em relação ao CPLEX. Os três métodos heurísticos são comparados também através do *Relative Deviation Index* (RDI), cujas médias estão presentes na Tabela 39.

Tabela 39 - RDIs médios das meta-heurísticas para o JSFMD.

Dimensão ($n \times m$)	ILS	GRASP	ABC
5×5	0,294	0,584	0,298
10×5	0,444	0,924	0,270
10×10	0,235	0,984	0,220
15×10	0,093	0,952	0,417
15×15	0,140	1,000	0,174
30×15	0,477	0,419	0,519
50×5	0,438	0,768	0,287
50×10	0,704	0,265	0,662
50×15	0,704	0,271	0,616
100×10	0,633	0,210	0,662
Média geral	0,416	0,638	0,412

Verifica-se que ILS e ABC tiveram desempenhos parecidos. Seus RDIs médios estiveram próximos em todas as dimensões e, exceto pelos problemas com 30 *jobs*, destoaram dos RDIs obtidos pela GRASP. Tanto a meta-heurística de trajetória quanto a populacional conquistaram por 3 vezes o menor RDI médio. Já o método construtivo obteve o menor valor em 4 ocasiões, sempre em grandes dimensões. Considerando todas as dimensões, a ABC apresentou o melhor RDI médio geral, e, novamente, o padrão de desempenho com o JSFMD foi similar ao obtido com o JSF. Por fim foram delineados os perfis de desempenho de ILS, GRASP e ABC, apresentados na Figura 43.

Figura 43 - Perfis de desempenho das meta-heurísticas com instâncias JSFMD.

Fonte: elaborada pelo autor.

Na Figura 43, o distanciamento da curva da GRASP em relação às demais curvas mostra que a meta-heurística construtiva não acompanha o desempenho da ILS e da ABC. A curva da GRASP inicia com $Probabilidade(a, \tau)=0,30$ e alcança o extremo superior do gráfico com $\tau=2,97$, sempre sendo dominada por ILS e ABC. Entre ILS e ABC não existe dominância, mas, ao contrário, grande alternância. A ILS inicia com valor mais alto, igual a 0,40, em relação à ABC, cujo valor é 0,36. Porém, as curvas se cruzam em diversos pontos e a ABC alcança $Probabilidade(a, \tau)=1$ primeiramente, com $\tau=1,44$. Já a curva da ILS alcança o topo do eixo vertical com $\tau=2,25$.

Coerentemente com os perfis de desempenho, o teste de Friedman também indica haver diferenças estatisticamente significativas entre a GRASP e cada uma das demais meta-heurísticas aplicadas ao JSFMD. Entre ILS e GRASP, o teste indicou valor- $p=0,002$ e, entre GRASP e ABC, resultou valor- $p=0,000$. Por outro lado, o teste não detectou diferenças significativas entre ILS e ABC, uma vez que obteve valor- $p=0,739$. Esses resultados também seguem um padrão similar ao constatado com os problemas JSF.

Todos esses resultados indicam que a meta-heurística de trajetória e a meta-heurística populacional tem desempenho similar. Ambas se mostraram as mais adequadas para os problemas de menor porte. Por outro lado, a meta-heurística construtiva se mostrou mais eficiente para problemas de grande porte. Contudo, quando foi avaliado o desempenho geral, ABC e ILS se mostraram superiores à GRASP, com leve vantagem para o método populacional.

Adicionalmente foram realizados experimentos com as meta-heurísticas com ênfase construtiva aplicadas ao JSFMD. Foram utilizados como limites os mesmos tempos consumidos pela ABC aplicada ao JSFMD com $15000/(n \cdot m)$ iterações. As porcentagens de

tempo dedicado à fase construtiva por cada método foram as mesmas utilizadas na Seção 7.9. Os atrasos médios da ILS com Ênfase Construtiva (ILS-EC), da GRASP com Ênfase Construtiva (GRASP-EC) e da ABC com Ênfase Construtiva (ABC-EC) estão disponíveis na Tabela 40.

Tabela 40 - Meta-heurísticas com ênfase construtiva aplicadas ao JSFMD.

Dimensão ($n \times m$)	T_{total} (Melhores soluções)			Limite de tempo (s)
	ILS-EC	GRASP-EC	ABC-EC	
5×5	366,2 (10)	366,8 (6)	353,7 (11)	8,590
10×5	1261,7 (7)	1284,7 (1)	1234,3 (9)	27,775
10×10	979,1 (9)	1077,6 (0)	957,5 (6)	56,090
15×10	1951,9 (9)	2091,9 (1)	1995,0 (5)	86,422
15×15	2270,3 (8)	2509,1 (1)	2330,3 (6)	166,563
30×15	9355,6 (6)	9410,9 (7)	9419,7 (2)	457,396
50×5	44886,5 (11)	45716,1 (0)	45065,7 (4)	475,355
50×10	35536,4 (6)	35704,6 (5)	35668,8 (5)	782,592
50×15	31822,9 (6)	31887,4 (6)	31920,3 (5)	1266,148
100×10	171486,2 (4)	171315,3 (10)	171690,3 (1)	2709,931
Média geral (Total)	29991,7 (76)	30136,4 (37)	30063,6 (54)	603,686 (-)

Os resultados observados na Tabela 40 permitem constatar que a utilização da ênfase construtiva com o JSFMD teve efeitos similares aos anteriormente observados com o JSF. A ABC-EC obteve a melhor média nas 3 primeiras dimensões, a GRASP-EC foi a melhor com a última dimensão e a ILS-EC conseguiu os menores atrasos médios nas demais 6 dimensões. Além disso, a ILS-EC obteve a menor média geral, 29991,7, e a maior quantidade de melhores soluções, 76. Em seguida veio a ABC-EC, com atraso médio geral igual a 30063,6 e 54 melhores resultados. Por fim, a GRASP-EC, com média geral 30136,4 e 37 melhores soluções. Pode-se destacar que, na dimensão 50×15, a ILS-EC reduziu o atraso da DMA-AF-MD em 7,36%, um ganho cerca de 2,4 vezes maior que o obtido pela ILS sobre o mesmo referencial.

A mesma ordem de desempenho também foi observada quando foram calculados os RDIs das meta-heurísticas. A ILS-EC obteve o menor RDI médio geral, igual a 0,340. Em seguida veio a ABC-EC com 0,412 e, então, a GRASP-EC com o valor 0,647. A análise auxiliada pelos perfis de desempenho indicou, assim como antes, a GRASP-EC com desempenho inferior aos obtidos por ILS-EC e ABC-EC. Os três métodos alcançaram $Probabilidade(a, \tau)=1$ com valores de τ menores que 1,8, sendo que as curvas de ILS-EC e ABC-EC evoluíram próximas, inclusive se alternando. Por fim o teste de Friedman sugere que existem diferenças estatisticamente significativas entre ILS-EC e GRASP-EC, com valor- $p=0,000$, e entre ABC-EC e GRASP-EC, com valor- $p=0,000$. Entre ILS-EC e ABC-EC o resultado do teste ficou bastante próximo do nível de significância igual a 0,05, indicando valor- $p=0,049$. Assim, por pequena margem, o teste detectou indícios de que existem diferenças estatisticamente significativas entre ILS-EC e ABC-EC. Tal resultado ainda não havia ocorrido nas comparações entre a meta-heurística de trajetória e a meta-heurística populacional. De fato o desempenho da ILS-EC pode ser considerado superior ao da ABC-EC e ao da GRASP-EC, uma vez que se destacou em todos os tipos de análises realizadas.

9 CONCLUSÕES

Este trabalho abordou o problema de programação de tarefas *Job Shop* Flexível (JSF) com o critério de minimização do atraso total. Foram propostos e implementados modelos matemáticos, heurísticas construtivas, Buscas Locais (BL) e meta-heurísticas de diferentes categorias. Também foi estudada uma nova versão do problema, o JSF com Montagem e Desmontagem (JSFMD). Foram gerados dois conjuntos de instâncias, ambos com 10 dimensões de problemas que envolvem desde 5 *jobs* e 5 máquinas até 100 *jobs* e 10 máquinas. Um dos conjuntos é composto por 600 instâncias do JSF, as quais são distribuídas em três níveis de flexibilidade e quatro tipos de data de entrega. O outro conjunto é formado por 150 problemas do JSFMD e também é dividido em três níveis de flexibilidade. Tais conjuntos foram utilizados nos experimentos com o critério de minimização do atraso total e atendem à carência de instâncias do problema existente na literatura.

O modelo matemático adaptado que minimiza o *makespan* foi ligeiramente melhor em termos de resultados que o modelo da literatura. Isso pode indicar que a representação explícita dos arcos de precedência pode ser mais adequada para a resolução do problema. Além disso, ele é capaz de trabalhar com *jobs* que possuem operações de montagem e desmontagem, o que não foi encontrado na literatura. Com relação à minimização do atraso total, os resultados do modelo proposto mostraram que a formulação apresentada é capaz de fornecer resultados ótimos ou limitantes para instâncias de menor porte. Isso é importante, pois esses resultados balizam a avaliação de métodos não exatos. Constatou-se a dificuldade do CPLEX, mesmo após 1 hora de processamento, de obter limitantes superiores para instâncias com 15 *jobs* e 15 máquinas ou para instâncias maiores.

Com relação ao JSF, as heurísticas construtivas, baseadas nas diversas regras de prioridade propostas, cumpriram seu propósito, pois foram capazes de gerar soluções factíveis de qualidade aceitável demandando tempo computacional reduzido. Entre as regras básicas o melhor desempenho foi, em geral, da regra proposta Data de entrega Modificada com Antevisão (DMA), sendo que sua superioridade ocorreu especialmente nos problemas de maior porte. As estratégias baseadas na afinidade e na flexibilidade das operações permitiram explorar características específicas do JSF ao serem incorporadas às regras de prioridade. Essas estratégias, juntamente com a antevisão, permitiram que o atraso total médio fosse reduzido em até 21,92%, quando associadas à regra *Earliest Due Date* (EDD), sem aumento significativo do tempo computacional. Além disso, essa regra conseguiu superar a melhor

regra da literatura, a *Modified Due Date* (MDD), em mais de 81% das instâncias. Desse modo, fica claro o quão vantajoso pode ser utilizar o conhecimento a respeito do problema abordado. Além disso, na média, as heurísticas construtivas foram capazes de gerar melhores resultados que o *software* de programação matemática com processamento limitado a 1 hora. De modo geral, se pode afirmar que as estratégias de exploração e as regras propostas podem ser usadas para gerar soluções iniciais com rapidez, as quais podem servir como ponto de partida para métodos de melhoria mais sofisticados.

Foram implementadas quatro BLs, duas baseadas no movimento de inversão de arcos e duas baseadas no movimento de inserção de operações. Uma das BLs baseadas em cada movimento utilizou atualizações internas da melhor solução encontrada. Os resultados indicaram que as atualizações internas permitem grande redução no tempo de execução. No caso da BL baseada em inversão de arcos, as atualizações internas permitiram reduzir 78% do tempo em relação à BL equivalente sem tal característica. Com as BLs baseadas no movimento de inserção essa redução no tempo foi igual a 84%. Em relação às soluções iniciais, as melhorias obtidas pela BL baseada no movimento de inserção com atualizações internas foram próximas a 13%. Com isso, a mesma se mostrou a mais adequada para ser utilizada pelas meta-heurísticas.

Os resultados das meta-heurísticas *Iterated Local Search* (ILS), *Greedy Randomized Adaptive Search Procedure* (GRASP) e *Artificial Bee Colony* (ABC) mostraram que tais métodos foram capazes de obter melhorias significativas com tempos computacionais aceitáveis. Em algumas das dimensões a redução média do atraso total, em relação à solução inicial, ficou em torno de 60% e, em relação à BL, ultrapassou 40%. As melhorias alcançadas pelos métodos propostos superaram, com margem considerável, as melhorias obtidas por uma referência da literatura que também utiliza uma meta-heurística. Na comparação com a resolução exata, as meta-heurísticas ILS, GRASP e ABC conseguiram resultados expressivos em todas as dimensões. Na dimensão 5×5 a GRASP proposta ficou a 1,58% dos atrasos ótimos. Nas instâncias com 15 *jobs* e 10 máquinas o atraso total médio da ILS foi 66,98% inferior ao fornecido pelo CPLEX. No geral, a ABC proposta encontrou 91 dos 117 ótimos indicados pelo CPLEX e, nesses 117 problemas, alcançou atraso apenas 2,24% maior que os valores ótimos. Os tempos das três meta-heurísticas podem ser considerados aceitáveis e compatíveis, inclusive, com aplicações práticas.

A comparação entre ILS, GRASP e ABC executadas com os mesmos tempos indicou que o método de trajetória tem o melhor desempenho com as menores instâncias. Por outro lado, o método baseado em construção de soluções tem os melhores resultados com os

problemas de grande porte. Enquanto isso, os resultados da meta-heurística populacional se mantêm próximos dos melhores, resultando no melhor desempenho geral. Apesar de as análises baseadas no atraso total médio, no *Relative Deviation Index* (RDI) e nos perfis de desempenho confirmarem o melhor desempenho da ABC, a mesma não apresentou grande vantagem sobre a ILS. De modo geral, as comparações entre as três meta-heurísticas indicaram ser vantajoso, especialmente com problemas de grande porte, transferir esforços para a geração de soluções com aleatoriedade.

A estratégia de concentrar mais esforços na geração de soluções permitiu que a ILS com Ênfase Construtiva (ILS-EC) revertere a desvantagem que a ILS proposta apresentava com os maiores problemas. Em problemas com 100 *jobs*, o método de trajetória mais que dobrou a melhoria obtida em relação à solução inicial determinística, sem aumento do tempo total de resolução. No caso da GRASP com Ênfase Construtiva (GRASP-EC), em algumas ocasiões, foi vantajoso aplicar até 90% do tempo de resolução exclusivamente na fase construtiva com aleatoriedade. Com isso, foram conseguidas melhorias adicionais superiores a 10%, sem acréscimo do tempo total. A ênfase construtiva permitiu também que a ABC com Ênfase Construtiva (ABC-EC) ampliasse em 11 o número de soluções ótimas obtidas e que apresentasse atraso médio somente 1,45% superior ao atraso médio das 117 soluções ótimas conhecidas.

Com relação ao JSFMD, a incorporação da sincronização dos ramos de um mesmo *job* às heurísticas construtivas gerou melhorias superiores a 9%. As meta-heurísticas também foram capazes de reduzir significativamente os atrasos, embora tenham obtido melhorias menos amplas que as observadas com o JSF. Em algumas dimensões as melhorias ficaram próximas a 40%. De modo geral, os desempenhos relativos entre as meta-heurísticas aplicadas ao JSFMD foram similares aos observadas com o JSF. Os experimentos com as versões das meta-heurísticas com ênfase construtiva também resultaram em melhorias sem aumento do tempo de execução, com um melhor desempenho da ILS-EC em relação às demais.

Com esses resultados se pode considerar que as heurísticas construtivas e as meta-heurísticas desenvolvidas cumpriram seu propósito. Foram capazes de fornecer soluções de qualidade em tempo aceitável para instâncias desde pequeno até grande porte. Tais métodos, juntamente com os conhecimentos adquiridos a respeito do desempenho das meta-heurísticas de diferentes categorias, podem ser utilizados em trabalhos futuros. Esses trabalhos podem envolver a resolução de variações do problema, incluindo outros critérios de otimização.

REFERÊNCIAS

- AARTS, E.; LENSTRA, J. **Local search in combinatorial optimization**. Princeton University Press, 2003.
- AKAY, B.; KARABOGA, D. Solving integer programming problems by using artificial bee colony algorithm, **Lecture Notes in Computer Science**, v. 5883, p. 355–364, 2009.
- ALVAREZ-VALDES, R.; FUERTES, A.; TAMARIT, J. GIMÉNEZ, G.; RAMOS, R. A heuristic to schedule flexible job-shop in a glass factory, **European Journal of Operational Research**, v. 165, n. 2, p. 525–534, 2005.
- ARROYO, J.; PEREIRA, A. A GRASP heuristic for the multi-objective permutation flowshop scheduling problem, **International Journal of Advanced Manufacturing Technology**, v. 55, n. 5–8, p. 741–753, 2011.
- BAGCHI, T. **Multiobjective scheduling by genetic algorithms**. Springer, 1999.
- BAKER, K. **Introduction to sequencing and scheduling**. Wiley, 1974.
- BAKER, K. Sequencing rules and due-date assignments in a job shop, **Management Science**, v. 30, n. 9, p. 1093–1104, 1984.
- BAKER, K.; BERTRAND, J. A dynamic priority rule for scheduling against due-dates, **Journal of Operations Management**, v. 3, n. 1, p. 37–42, 1982.
- BALAS, E. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, **Operations Research**, v. 17, n. 6, p. 941–957, 1969.
- BAXTER, J. Local optima avoidance in depot location, **Journal of the Operational Research Society**, v. 32, n. 9, p. 815–819, 1981.
- BAYKASOĞLU, A.; ÖZBAKIR, L. Analyzing the effect of dispatching rules on the scheduling performance through grammar based flexible scheduling system, **International Journal Production Economics**, v. 124, n. 2, p. 369–381, 2010.
- BERTRAND, J.; FRANSOO, J. Modelling and simulation: operations management research methodologies using quantitative modeling, **International Journal of Operations and Production Management**, v. 22, n. 2, p. 241–264, 2002.
- BIRGIN, E.; FEOFILOFF, P.; FERNANDES, C.; MELO, E.; OSHIRO, M.; RONCONI, D. A MILP model for an extended version of the Flexible Job Shop Problem, **Optimization Letters**, aceito para publicação, 2013.
- BIRGIN, E.; FEOFILOFF, P.; FERNANDES, C.; MELO, E.; OSHIRO, M.; RONCONI, D. Um estudo sobre formulações de programação inteira mista para o problemas de job shop flexível, **Anais do XLIII Simpósio Brasileiro de Pesquisa Operacional**, Ubatuba, 2011.

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

BOWMAN, E. The scheduling sequence problem, **Operations Research**, v. 7, n. 5, p. 621–624, 1959.

BRANDIMARTE, P. Routing and scheduling in a flexible job shop by tabu search, **Annals of Operations Research**, v. 41, n. 3, p. 157–183, 1993.

BRESINA, J. Heuristic-biased stochastic sampling, **Proceedings of the AAAI-96**, p. 271–278, Portland, 1996.

BRUCKER, P.; SCHILE, R. Job-shop scheduling with multi-purpose machines, **Computing**, v. 45, n. 4, p. 369–375, 1990.

CHAN, F.; WONG, T.; CHAN, L. Flexible job-shop scheduling problem under resource constraints, **International Journal of Production Research**, v. 44, n. 11, p. 2071–2089, 2006.

CHEN, J.; CHEN, K.; WU, J.; CHEN, C. A study of the flexible job shop scheduling problem with parallel machines and reentrant process, **International Journal of Advanced Manufacturing Technology**, v. 39, n. 3–4, p. 344–354, 2008.

CHENG, R.; GEN, M.; TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithm – I, **Computers and Industrial Engineering**, v. 30, n. 4, p. 983–997, 1996.

CHU, C.; PORTMAN, M. Some new efficient methods to solve the $n / 1 / r_i / \sum T_i$ scheduling problem, **European Journal of Operational Research**, v. 58, n. 3, p. 404–413, 1992.

CONWAY, R.; MAXWELL, W.; MILLER, L. **Theory of scheduling**. Addison-Wesley Publishing, 1967.

CORDEAU, J.; GENDREAU, M.; LAPORTE, G.; POTVIN, J.; SEMET, F. A guide to vehicle routing heuristics, **Journal of the Operational Research Society**, v. 53, n. 5, p. 512–522, 2002.

CORMEN, T.; LEISERSON, C.; RIVEST, R.; STEIN, C. **Algoritmos: teoria e prática**. 2 ed. Tradução: Vanderberg de Sousa. Elsevier, 2002.

DAMODARAN, P.; VÉLEZ-GALLEGO, M.; MAYA, J. A GRASP approach for makespan minimization on parallel batch processing machines, **Journal of Intelligent Manufacturing**, v. 22, n. 5, p. 767–777, 2011.

DAUZÈRE-PÉRÈS, S.; PAULLI, J. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, **Annals of Operations Research**, v. 70, p. 281–306, 1997.

- DELLA CROCE, F.; GARAIX, T.; GROSSO, A. Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem, **Computers & Operations Research**, v. 39, n. 6, p. 1213–1217, 2012.
- DELL'AMICO, M.; TRUBIAN, M. Applying tabu search to the job-shop scheduling problem, **Annals of Operations Research**, v. 41, n. 3, p. 231 – 252, 1993.
- DEMIR, Y.; İŞLEYEN, K. Evaluation of mathematical models for flexible job-shop scheduling problems, **Applied Mathematical Modelling**, v. 37, n. 13, p. 977–988, 2013.
- DEROUSSI, L.; GOURGAND, M.; TCHERNEV, N. A simple metaheuristic approach to the simultaneous scheduling of machines and automated guided vehicles, **International Journal of Production Research**, v. 46, n. 8, p. 2143–2164, 2008.
- DOLAN, E.; MORE, J. Benchmarking optimization software with performance profiles, **Mathematical programming**, v. 91, n. 2, p. 201–213, 2002.
- DU, J.; LEUNG, J. Minimizing total tardiness on one machine is NP-Hard, **Mathematics of Operations Research**, v. 15, n. 3, p. 483–495, 1990.
- ESSAFI, I.; MATI, Y.; DAUZÈRE-PÉRÈS, S. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem, **Computers & Operations Research**, v. 35, n. 8, p. 2599–2616, 2008.
- FATTAHI, P.; MEHRABAD, M.; JOLAI, F. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, **Journal of Intelligent Manufacturing**, v. 18, n. 3, p. 331–342, 2007.
- FEO, T.; RESENDE, M. A probabilistic heuristic for a computationally difficult set covering problem, **Operations Research Letters**, v. 8, n. 2, p. 67–71, 1989.
- FEO, T.; RESENDE, M. Greedy randomized adaptive search procedures, **Journal of Global Optimization**, v. 6, n. 4, p. 109–133, 1995.
- FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance, **Journal of the American Statistical Association**, v. 32, n. 200, p. 675–701, 1937.
- GAO, W.; LIU, S. A modified artificial bee colony algorithm, **Computers & Operations Research**, v. 39, n. 3, p. 687–697, 2012.
- GAREY, M.; JOHNSON, D.; SETHI, R. The complexity of flowshop and jobshop scheduling, **Mathematics of Operations Research**, v. 1, n. 2, p. 117–129, 1976.
- GHOLAMI, M.; ZANDIEH, M. Integrating simulation and genetic algorithm to schedule a dynamic flexible job shop, **Journal of Intelligent Manufacturing**, v. 20, n. 4, p. 481–498, 2009.
- GRAHAM, R.; LAWLER, E.; LENSTRA, J.; RINNOOY KAN, A. Optimization and approximation in deterministic sequencing and scheduling: a survey em P. Hammer, E.

Johnson, e B. Korte (Ed.), **Annals of Discrete Mathematics**, v. 5, p. 287–326, Elsevier, 1979.

GUTIÉRREZ, C.; GARCÍA-MAGARIÑO, I. Modular design of a hybrid genetic algorithm for a flexible job-shop scheduling problem, **Knowledge-Based Systems**, v. 24, n. 1, p. 102–112, 2011.

HAN, Y.; PAN, Q.; LI, J.; SANG, H. An improved artificial bee colony algorithm for the blocking flowshop scheduling problem, **International Journal of Advanced Manufacturing Technology**, v. 60, n. 9–12, p. 1149–1159, 2012.

HEATH, S.; BARD, J.; MORRICE, D. A GRASP for simultaneously assigning and sequencing product families on flexible assembly lines, **Annals of Operations Research**, v. 203, n. 1, p. 295–323, 2013.

HO, N.; TAY, J.; LAI, E. An effective architecture for learning and evolving flexible job-shop schedules, **European Journal of Operational Research**, v. 179, n. 2, p. 316–333, 2006.

KACEM, I.; HAMMADI, S.; BORNE, P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, **IEEE Transactions on Systems, Man, and Cybernetics**, v. 32, n. 1, p. 1–13, 2002.

KAMPKE, E.; ARROYO, J.; SANTOS, A. Iterated Local Search with path relinking for solving parallel machines scheduling problem with resource-assignable sequence dependent setup times, **Lecture Notes in Computer Science**, v. 6022, n. , p. 107–118, 2010.

KARABOGA, D. **An idea based on honey bee swarm for numerical optimization**. Technical report – Engineering Faculty, Erciyes University. Kayseri. 2005.

KARABOGA, D.; BASTURK, B. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, **Journal of Global Optimization**, v. 39, n. 3, p. 459–471, 2007.

KARABOGA, D.; BASTURK, B. On the performance of artificial bee colony (ABC) algorithm, **Applied Soft Computing**, v. 8, n. 1, p. 687–697, 2008.

KOULAMAS, C. The total tardiness problem: review and extensions, **Operations Research**, v. 42, n. 6, p. 1025–1041, 1994.

LAURENT, B.; HAO, J. Iterated local search for the multiple depot vehicle scheduling problem, **Computers & Industrial Engineering**, v. 57, n. 1, p. 277–286, 2009.

LAWER, E.; LENSTRA, J.; RINNOOY KAN, A.; SHMOYS, D. Sequencing and scheduling: algorithms and complexity em S. Graves, A. Kan e P. Zipkin (Eds.), **Logistics of production and inventory**, North-Holland, 1993.

LI, J.; PAN, Q.; LIANG, Y. An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems, **Computers & Industrial Engineering**, v. 59, n. 4, p.647–662, 2010.

- LOURENÇO, H.; MARTIN, O.; STÜTZLE, T. Iterated local search: framework and applications em M. Gendreau e J. Potvin (Ed.), **Handbook of metaheuristics**, 2 ed., p. 363–397, Springer, 2010.
- MAINIERI, G.; RONCONI, D. New heuristics for total tardiness minimization in a flexible flowshop, **Optimization Letters**, v. 7, n. 4, p. 665–684, 2013.
- MANNE, A. On the job-shop scheduling problem, **Operations Research**, v. 8, n. 2, p. 219–223, 1960.
- MARTÍ, R.; RESENDE, M.; RIBEIRO, C. Multi-start methods for combinatorial optimization, **European Journal of Operational Research**, v. 226, n. 1, p. 1–8, 2013.
- MELO, E.; RONCONI, D. Regras de prioridade eficientes que exploram características do job shop flexível para a minimização do atraso total, **Production**, aceito para publicação, 2013.
- MISHRA, P.; PANDEY, P. Simulation modeling of batch job shop type flexible manufacturing systems, **Journal of Mechanical Working Technology**, v. 20, p. 441–450, 1989.
- MOUSAKHANI, M. Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness, **International Journal of Production Research**, v. 51, n. 12, p. 3476–3487, 2013.
- OSMAN, I. Preface: focussed issue on applied meta-heuristics. **Computers & Industrial Engineering**, v. 44, n. 2, p. 205–207, 2002.
- ÖZGÜVEN, C.; ÖZBAKIR, L.; YAVUZ, Y. Mathematical models for job-shop scheduling problems with routing and process plan flexibility, **Applied Mathematical Modelling**, v. 34, n. 6, p. 1539–1548, 2010.
- PAL, A.; CHAN, F.; MAHANTY, B.; E TIWARI M. Aggregate procurement, production, and shipment planning decision problem for a three-echelon supply chain using swarm-based heuristics, **International Journal of Production Research**, v. 49, n. 10, p. 2873–2905, 2011.
- PAN, Q.; RUIZ, R. Local search methods for the flowshop scheduling problem with flowtime minimization, **European Journal of Operational Research**, v. 222, n. 1, p. 31–43, 2012.
- PAN, Q.; TASGETIREN, M.; SUGANTHAN, P.; E CHUA, T. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, **Information Sciences**, v. 181, n. 12, p. 2455–2468, 2011.
- PANWALKAR, S.; ISKANDER, W. A survey of scheduling rules, **Operations Research**, v. 25, n. 1, p. 45–61, 1977.
- PEZZELLA, F.; MORGANTI, G.; CIASCHETTI, G. A genetic algorithm for the flexible job-shop scheduling problem, **Computers & Operations Research**, v. 35, n. 10, p. 3201–3212, 2008.

PINEDO, M. **Scheduling**: theory, algorithms, and systems. 4 ed. Springer, 2012.

PRAIS, M.; RIBEIRO, C. Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment, **INFORMS Journal on Computing**, v. 12, n. 3, p. 164–176, 2000.

RAJKUMAR, M.; ASOKAN, P.; ANILKUMAR, N.; PAGE, T. A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints, **International Journal of Production Research**, v. 49, n. 8, p. 2409–2423, 2011.

RAJKUMAR, M.; ASOKAN, P.; VAMSIKRISHNA, V. A GRASP algorithm for flexible jobshop scheduling with maintenance constraints, **International Journal of Production Research**, v. 48, n. 22, p. 6821–6836, 2010.

RESENDE, M.; RIBEIRO, C. Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications em M. Gendreau e J. Potvin (Ed.), **Handbook of metaheuristics**, 2 ed., p. 283–319, Springer, 2010.

RODRIGUEZ, F.; BLUM, C.; GARCÍA-MARTÍNEZ, C.; LOZANO, M. GRASP with path-relinking for the non-identical parallel machine scheduling problem with minimising total weighted completion times, **Annals of Operations Research**, v. 201, n. 1, p. 383–401, 2012.

RONCONI, D.; HENRIQUES, L. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking, **Omega**, v. 37, n. 2, p. 272–281, 2009.

SCRICH, C. **Busca Tabu para a programação de tarefas em job shop com datas de entrega**. 1997. 117 p. Tese (Doutorado em Engenharia Elétrica) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 1997.

SCRICH, C.; ARMENTANO, V.; LAGUNA, M. Tardiness minimization in a flexible job shop: a tabu search approach, **Journal of Intelligent Manufacturing**, v. 15, n. 1, p. 103–115, 2004.

SHAO, X.; LIU, W.; LIU, Q.; ZHANG, C. Hybrid discrete particle swarm optimization for multi-objective flexible job-shop scheduling problem, **International Journal of Advanced Manufacturing Technology**, v. 67, n. 9–13, p. 2885–2901, 2013.

SILVER, E. An overview of heuristic solution methods, **Journal of the Operational Research Society**, v. 55, n. 9, p. 936–956, 2004.

SUBRAMANIAN, A.; PENNA, P.; UCHOA, E.; OCHI, L. A hybrid algorithm for the heterogeneous fleet vehicle routing problem, **European Journal of Operational Research**, v. 221, n. 2, p. 285–295, 2012.

SZETO, W.; WU, Y.; HO, S. An artificial bee colony algorithm for the capacitated vehicle routing problem, **European Journal of Operational Research**, v. 215, n. 1, p. 126–135, 2011.

- TAILLARD, E. Benchmarks for basic scheduling problems, **European Journal of Operational Research**, v. 64, n. 2, p. 278–285, 1993.
- TAILLARD, E. Parallel taboo search techniques for job shop scheduling problem, **ORSA Journal of Computing**, v. 6, n. 2, p. 108–117, 1994.
- TASGETIREN, M.; PAN, Q.; SUGANTHAN, P.; CHEN, A. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops, **Information Sciences**, v. 181, n. 16, p. 3459–2475, 2011.
- TAY, J.; HO, N. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems, **Computers & Industrial Engineering**, v. 54, n. 3, p. 453–473, 2008.
- VAESSENS, R.; AARTS, E.; LENSTRA, J. Job shop scheduling by local search, **INFORMS Journal of Computing**, v. 8, n. 3, p. 302–317, 1996.
- VEPSALAINEN, A.; MORTON, T. Priority rules for job shop with weighted tardiness costs, **Management Science**, v. 33, n. 8, p. 1035–1047, 1987.
- VILCOT, G.; BILLAUT, J. A tabu search and a genetic algorithm for solving a bicriteria general job shop scheduling problem, **European Journal of Operational Research**, v. 190, n. 12, p. 398–411, 2008.
- WAGNER, H. An integer linear programming model for machine scheduling, **Naval Research Logistics Quarterly**, v. 6, n. 2, p. 131–140, 1959.
- WANG, L.; ZHOU, G.; XU, Y.; WANG, S.; LIU, M. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem, **International Journal of Advanced Manufacturing Technology**, v. 60, n. 1–4, p. 303–315, 2012.
- YUAN, Y.; XU, H. An integrated search heuristic for large-scale flexible job shop scheduling problems, **Computers & Operations Research**, v. 40, n. 12, p. 2864–2877, 2013.
- YUAN, Y.; XU, H. Flexible job shop scheduling using hybrid differential evolution algorithms. **Computers & Industrial Engineering**, v. 65, n. 2, p. 246–260, 2013.
- ZANAKIS, S.; EVANS, J.; VAZACOPOULOS, A. Heuristic methods and applications: a categorized survey. **European Journal of Operational Research**, v. 43, n. 1, p. 88–110, 1989.
- ZHANG, G.; SHAO, X.; LI, P.; GAO, L. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem, **Computers & Industrial Engineering**, v. 56, n. 4, p. 1309–1318, 2009.
- ZHANG, H.; GEN, M. Multistage-based genetic algorithm for flexible job-shop scheduling problem, **Complexity International**, v. 11, p. 223–232, 2005.

APÊNDICE A – Número de soluções do JSF em função de n e m

O gráfico da Figura 44 compara o número de soluções do *Job Shop* Flexível (JSF) com o do *Job Shop* (JS) à medida que n varia de 1 a 4 e m é mantido igual a 2. Para se ter outra referência de comparação, também são plotadas as quantidades de soluções, em função de n , para o Problema do Caixeiro Viajante, ou *Travelling Salesman Problem* (TSP), um dos problemas combinatórios mais conhecidos na literatura. No caso do TSP o valor n se refere ao número de locais a serem visitados.

Figura 44 - Número de soluções de TSP, JS e JSF em função de n .

Fonte: elaborada pelo autor.

Com $n=1$, tanto o TSP quanto o JS têm somente uma solução, enquanto o JSF com flexibilidade total e m operações por *job* tem 4 soluções. À medida que n cresce as três curvas se distanciam e fica nítido o descolamento da curva referente ao JSF em relação à curva do JS. Mantido $m=2$ e tendo $n=4$ o JSF possui 256 vezes o número de soluções do JS e 6144 vezes o número de soluções do TSP. Na Figura 45 se observa a evolução do número de soluções dos três problemas com valores mais elevados de n . Para tanto se utiliza escala logarítmica.

Figura 45 - Número de soluções de TSP, JS e JSF em escala logarítmica.

Fonte: elaborada pelo autor.

Em escala logarítmica é possível notar com maior clareza o quanto o número de soluções do JSF se distancia do número de soluções do JS com n aumentando de 1 a 50. Como esperado, fica evidente que a curva do JSF cresce mais rapidamente que a curva do JS, indicando que o JSF é mais complexo. Com $n=25$ o JSF tem cerca de 10^{15} vezes mais soluções que o JS. Com $n=50$ o JSF passa a ter em torno de 10^{31} vezes mais possibilidades de solução que o JS tradicional. Mais que isso, ambos os problemas apresentam um crescimento de número de soluções bem mais acentuado que o terceiro problema usado como referência, o TSP. Esses gráficos deixam claro o quão difícil é o JSF.

APÊNDICE B – Resultados adicionais da ILS

A execução da ILS com BL-ARC-AI foi realizada com limite de tempo igual ao utilizado pela versão proposta da ILS com BL-INS-AI e $60000/(n \cdot m)$ iterações. As versões ILS-Elite e ILS-PV foram executadas com $60000/(n \cdot m)$ iterações. Todos esses resultados estão condensados na Tabela 41 juntamente com os resultados da ILS proposta.

Tabela 41 - Resultados de diferentes versões da ILS.

Dimensão ($n \times m$)	ILS proposta	ILS com BL-ARC-AI	ILS-Elite	ILS-PV
5×5	126,3	143,9	124,7	125,5
10×5	333,6	384,0	332,8	340,5
10×10	250,1	370,2	254,1	266,5
15×10	412,3	582,2	418,0	442,6
15×15	516,2	770,1	521,1	566,9
30×15	1835,1	2400,6	1845,9	1975,6
50×5	15706,4	16369,5	15791,2	16124,7
50×10	10076,6	10820,3	10077,4	10317,6
50×15	7730,1	8487,1	7823,8	8009,1
100×10	63156,5	64251,8	63245,9	63551,3
Média	10014,3	10458,0	10043,5	10172,0

O intuito da apresentação desses resultados é permitir que as principais versões implementadas da ILS possam ser comparadas entre si e também com a ILS proposta.

APÊNDICE C – Resultados adicionais da GRASP

A GRASP com BL-ARC-AI foi executada com um limite de tempo igual ao tempo consumido pela versão clássica da GRASP com BL-INS-AI e $60000/(n \cdot m)$ iterações. Esse também foi o número de iterações da GRASP Reativa, identificada como GRASP proposta. Os resultados desses métodos estão presentes na Tabela 42.

Tabela 42 - Resultados de diferentes versões da GRASP.

Dimensão ($n \times m$)	GRASP proposta	GRASP com BL-ARC-AI	GRASP clássica
5×5	122,2	123,8	121,7
10×5	351,5	358,7	346,8
10×10	291,4	357,5	282,3
15×10	454,3	522,2	462,1
15×15	633,9	774,3	650,5
30×15	1967,0	2158,2	1982,4
50×5	15937,8	16273,8	15977,5
50×10	9913,7	10005,5	9928,9
50×15	7539,8	7712,9	7529,5
100×10	62041,2	62134,3	62065,2
Média	9925,3	10042,1	9934,7

Esses resultados são disponibilizados para que possam ser feitas comparações entre as diferentes versões implementadas da meta-heurística construtiva.

APÊNDICE D – Resultados adicionais da ABC

A ABC proposta foi executada por $30000/(n \cdot m)$ iterações, assim como a ABC com população inicial que utiliza as melhores regras e como a ABC com população inicial aleatória. A ABC sem o procedimento EV-INS-AI, a ABC com a BL-INS-AI e a MABC foram executadas com os mesmos tempos de execução da ABC proposta. A ABC cuja população inicial é gerada utilizando uma LRC baseada na porcentagem θ é aqui denominada ABC proposta. Os resultados desses seis métodos constam na Tabela 43.

Tabela 43 - Resultados de diferentes versões da ABC.

Dimensão ($n \times m$)	ABC proposta	ABC pop. inicial melhores regras	ABC pop. inicial aleatória	ABC sem EV-INS-AI	ABC com BL-INS-AI	MABC
5×5	124,9	123,5	125,1	124,9	124,6	125,1
10×5	328,8	328,0	329,7	328,8	328,7	333,8
10×10	240,7	236,9	242,9	240,5	240,5	246,0
15×10	402,8	410,1	419,4	402,4	393,4	414,3
15×15	508,8	522,0	562,5	508,3	508,2	545,1
30×15	1876,4	1925,2	2212,2	1874,1	1838,0	1988,0
50×5	15545,0	15654,9	17151,7	15551,6	15569,8	15746,4
50×10	9916,0	10121,1	12332,7	9919,5	9922,0	10093,8
50×15	7603,3	7812,6	10869,5	7597,3	7592,1	7901,9
100×10	62348,9	63125,9	86615,5	62341,6	62324,0	62867,6
Média	9889,6	10026,0	13086,1	9888,9	9884,1	10026,2

Esses resultados são apresentados para que possam ser realizadas comparações entre as principais versões implementadas da ABC.

Os atrasos médios, por dimensão, de todas as soluções das populações iniciais da ABC podem ser consultados na Tabela 44. Essa tabela traz na, segunda coluna, as médias da heurística construtiva determinística aplicando a regra EDD-AAF. A terceira coluna apresenta as médias dos atrasos de todas as soluções que formaram as populações iniciais. A quarta coluna traz a média relativa às melhores soluções iniciais utilizadas, uma de cada instância. Na última coluna consta a redução que houve no atraso médio, por dimensão, quando se utilizou a geração de população inicial proposta, em relação à regra EDD-AAF determinística. Para determinar tais reduções se utilizou os valores da quarta e da segunda colunas da Tabela 44.

Tabela 44 - Atrasos médios da população inicial da ABC e da melhor solução inicial.

Dimensão ($n \times m$)	EDD-AAF	Pop. inicial da ABC proposta	Solução incumbente inicial da ABC proposta	%Dif
5×5	187,7	199,3	160,3	-14,62
10×5	554,7	569,1	471,6	-14,99
10×10	587,2	606,9	495,6	-15,61
15×10	866,1	879,1	703,6	-18,77
15×15	1130,1	1155,5	996,3	-11,84
30×15	2750,6	2790,2	2506,5	-8,87
50×5	17355,3	17379,4	16929,9	-2,45
50×10	11242,6	11299,0	10754,8	-4,34
50×15	8746,2	8836,8	8344,0	-4,60
100×10	64681,6	64734,9	63516,8	-1,80
Média	10810,2	10845,0	10487,9	-2,98

Tais resultados mostram como a geração de população inicial proposta para a ABC auxiliou na redução do atraso total. Os ganhos obtidos com construções envolvendo aleatoriedade são o tema tratado na Seção 7.9.

APÊNDICE E – Exemplo de instância do JSF e grafos do JSFMD

A seguir é exemplificada e descrita uma instância gerada para o JSF.

5	□	Número de <i>jobs</i> do problema.
5	□	Número de máquinas do problema.
152 3		Informações sobre os <i>jobs</i> . Cada <i>job</i> ocupa duas linhas. Na primeira linha consta a data de entrega do <i>job</i> e seu número de operações. Na segunda linha são listadas suas operações. Por exemplo, o primeiro <i>job</i> tem data de entrega 152 e possui 3 operações, sendo elas 1, 2 e 3.
1 2 3		
285 5		
4 5 6 7 8		
243 4		
9 10 11 12		
222 4		
13 14 15 16		
326 5		
17 18 19 20 21		
16	□	Número de arcos de precedência do problema
1 1 2		Informações sobre os arcos de precedência de cada <i>job</i> . Em cada linha é detalhado um arco de precedência. Primeiro o <i>job</i> ao qual o arco pertence e, depois, a operação que deve preceder e a operação que deve ser precedida. Por exemplo, o primeiro arco pertence ao <i>job</i> 1 e determina que a operação 1 deve preceder a operação 2.
1 2 3		
2 4 5		
2 5 6		
2 6 7		
2 7 8		
3 9 10		
3 10 11		
3 11 12		
4 13 14		
4 14 15		
4 15 16		
5 17 18		
5 18 19		
5 19 20		
5 20 21		
1 3 3 20 1 35 4 56		Informações sobre as máquinas alternativas de cada operação. Em cada linha são detalhadas informações sobre uma operação. Primeiramente é identificada a operação e, em seguida, seu número de máquinas alternativas. Depois cada máquina alternativa é listada juntamente com seu tempo de processamento para a operação correspondente. Por exemplo, a operação 1 possui 3 máquinas alternativas: máquina 3, com tempo de processamento 20, máquina 1, com tempo 35 e máquina 4 com tempo 56.
2 4 1 80 2 85 3 99 4 99		
3 2 4 44 5 82		
4 4 1 79 3 99 5 95 2 98		
5 4 2 44 5 53 4 87 1 74		
6 3 2 63 5 97 1 68		
7 2 3 68 2 79		
8 4 3 25 5 56 2 60 4 57		
9 4 1 58 5 67 3 70 4 90		
10 2 4 50 3 96		
11 2 3 70 2 79		
12 4 1 79 5 81 3 97 2 84		
13 2 1 93 4 98		
14 4 5 85 2 94 3 90 4 98		
15 2 3 14 5 21		
16 3 4 69 5 74 2 78		
17 2 5 81 4 87		
18 2 1 63 3 83		
19 4 1 50 3 86 5 92 4 80		
20 4 5 90 1 98 3 93 2 92		
21 2 2 68 1 93		

As instâncias JSFMD possuem o mesmo formato de dados das instâncias JSF. Adicionalmente, cada instância JSFMD apresenta os grafos dos *jobs* envolvidos. Os grafos presentes em uma instância JSFMD com 10 *jobs* e 10 máquinas são apresentados a seguir.

Job 1:

Job 2:

Job 3:

Job 4:

Job 5:

Job 6:

Job 7:

Job 8:

Job 9:

$$\begin{array}{ccccccc}
 & & & & 116 & - & 117 & - & 118 \\
 & & & & / & & & & \\
 112 & - & 113 & - & 114 & - & 115 & - & 119 & - & 120 & - & 121 \\
 & & & & \backslash & & & & \\
 & & & & 122 & - & 123 & - & 124
 \end{array}$$

Job 10:

$$\begin{array}{ccccccccccccccc}
 & & & & & & 128 & - & 129 & - & 130 & - & 131 & - & 132 & - & 133 \\
 & & & & & & / & & & & & & & & & & \backslash \\
 125 & - & 126 & - & 127 & - & 134 & - & 135 & - & 136 & - & 137 & - & 138 & - & 139 & - & 146 \\
 & & & & & & \backslash & & & & & & & & & & / \\
 & & & & & & 140 & - & 141 & - & 142 & - & 143 & - & 144 & - & 145
 \end{array}$$