Universidade Federal de Minas Gerais

Programa de Pós-Graduação em Engenharia de Produção

Departamento de Engenharia de Produção / Escola de Engenharia

Problema de Seqüenciamento em uma Máquina com Penalidades por Antecipação e Atraso: Modelagem e Resolução

Aloísio de Castro Gomes Júnior

Orientador: Prof. Dr. Carlos Roberto Venâncio de Carvalho

Co-Orientador: Prof. Dr. Marcone Jamilson Freitas Souza (UFOP)

Belo Horizonte

Março, 2007

Universidade Federal de Minas Gerais

Programa de Pós-Graduação em Engenharia de Produção

Departamento de Engenharia de Produção / Escola de Engenharia

Problema de Seqüenciamento em uma Máquina com Penalidades por Antecipação e Atraso: Modelagem e Resolução

Aloísio de Castro Gomes Júnior

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Univesidade Federal de Minas Gerais para a obtenção do título de Mestre em Engenharia de Produção.

Área de Concentração: Produção e Logística

Linha de Pesquisa: Otimização de Sistemas Produtivos

Orientador: Prof. Dr. Carlos Roberto Venâncio de Carvalho

Co-Orientador: Prof. Dr. Marcone Jamilson Freitas Souza (UFOP)

Belo Horizonte

Março, 2007

"Dedico este trabalho ao meu pai, homem sereno de várias virtudes, um exemplo em todas as fases da minha vida."

Resumo

O presente trabalho trata do problema de seqüenciamento em uma máquina com penalidades por antecipação e atraso da produção, com tempo de preparação da máquina dependente da seqüência de produção e janelas de entrega. Para a contextualização do problema estudado são apresentadas várias técnicas utilizadas por vários autores para a resolução do mesmo e de problemas afins. Primeiramente é proposto um modelo de programação linear inteira mista (PLIM) para representar o problema. Este modelo foi implementado usando a ferramenta de modelagem AMPL e resolvido pelo *software* de otimização CPLEX 9.1. Em seguida, são propostos métodos heurísticos de resolução baseados nas meta-heurísticas GRASP e ILS. Para cada seqüência de jobs gerada pelas heurísticas propostas, usa-se um algoritmo de tempo polinomial para determinar a data ótima de início de processamento dos jobs na seqüência dada. Experimentos computacionais realizados sobre um conjunto de problemas-teste gerados aleatoriamente indicam que a performance dos métodos heurísticos propostos é muito boa. Em problemas de pequenas dimensões (8 a 12 *jobs*) os métodos conseguiram quase sempre alcançar o valor ótimo e em problemas de dimensões maiores (15 a 75 *jobs*) obtiveram desvios baixos em relação à melhor solução obtida.

Palavras-Chave: Seqüenciamento em uma máquina, GRASP, ILS, Programação Linear Inteira Mista, Metaheurísticas.

Abstract

This dissertation deals with the problem of scheduling single machine with earliness and tardiness penalties, with sequence dependent setup and due windows. For contextualization of the problem studied, some techniques used by others authors to solve it and similar problems are showed. Firstly, a model of mixed integer linear programming is proposed to represent the problem. This model was implemented using the modeling tool AMPL and solved by optimization software CPLEX 9.1. Afterwards, heuristics methods based on GRASP, Iterated Local Search and Variable Neighborhood Descent to solve it are proposed. For each job sequence generated by the heuristics, an optimal timing algorithm is used to determine the starting time for each job in the job sequence. Computational experiments realized with instances randomly generated show that the methods proposed are able to reach the optimal solution in most of the small instances (8 to 12 jobs) and yield low gaps in instances with 15 to 75 jobs.

Keywords: Single Machine Scheduling, GRASP, ILS, Mixed Integer Linear Programming, Metaheuristics.

Agradecimentos

Gostaria de agradecer, primeiramente, à Deus pelo dom da vida.

À UFMG pela oportunidade.

Aos professores Carlos e Marcone pela orientação e pelas preciosas dicas, imprescindíveis para a realização deste trabalho.

À CAPES pelo investimento em minha formação.

Ao Pablo pela fundamental ajuda com a implementação dos códigos.

À minha família pelo apoio e incentivos constantes.

À minha namorada Kívia pelo incentivo, confiança, compreensão e carinho.

À minha amiga Valéria e seus pais por terem me acolhido e me aguentado durante 7 meses.

Aos meus amigos, colegas da faculdade, professores do DEP, colegas do Unileste e à todos que de uma forma ou de outra contribuíram para a realização deste trabalho.

Sumário

Lista de Figuras **x**

Lista de Tabelas **xii**

Lista de Siglas **xiii**

1 Introdução **1**

1.1 Relevância do Trabalho 3

1.2 Objetivos 3

1.2.1 Objetivo Geral 4

1.2.2 Objetivos Específicos 4

1.3 Estrutura do Trabalho 4

2 Revisão Bibliográfica **6**

2.1 O Problema de Programação da Produção 7

2.1.1 Os Principais Tipos de Problemas de Programação da Produção 8

2.1.1.1 O Problema de Seqüenciamento em uma Máquina . . . 8

2.1.1.2 Outros tipos de Problema de Programação da Produção 9

2.1.2 Os Medidores de Desempenho Não Regulares 10

2.1.3 O Tempo de Preparação das Máquinas 12

2.2 Modelagem Matemática 14

2.2.1 Modelos Matemáticos para o PPP 15

2.3 Métodos de Resolução 19
2.3.1 Métodos Exatos 20
2.3.2 Métodos Heurísticos 21
2.3.2.1 Heurísticas Construtivas 21
2.3.2.2 Heurísticas de Refinamento 22
2.3.3 Meta-heurísticas 23
2.3.3.1 Busca Tabu 24
2.3.3.2 Algoritmos Genéticos 25
2.3.3.3 ILS 25
2.3.3.4 GRASP 26
2.3.3.5 Outras Meta-heurísticas 26
2.3.3.6 Aplicações de Meta-heurísticas ao PPP 27

3 Metodologia **32**

3.1 O Problema Estudado 32
3.2 Modelagem Matemática 33
3.3 Métodos de Resolução 37
3.3.1 Determinação das Datas Ótimas de Início de Processamento . . 37
3.3.2 Determinação da Seqüência de *Jobs* 44
3.3.2.1 Representação de uma Solução 45
3.3.2.2 Vizinhança de uma Solução 45
3.3.2.3 Função de Avaliação 46
3.3.2.4 Geração da Solução Inicial 46
3.3.2.5 Métodos de Busca Local 47
3.3.2.6 ILS 48
3.3.2.7 GRASP 50

4 Resultados e Análise **52**

4.1 Geração dos Problemas-Teste 52
4.2 Resultados da Modelagem Matemática 53
4.3 Resultados dos Métodos Heurísticos 57
4.3.1 Definição dos Parâmetros 57
4.3.2 Resultados Finais dos Métodos GRASP e ILS 63

5 Conclusão e Trabalhos Futuros **67**

Referências Bibliográficas **70**

Lista de Figuras

2.1 Exemplo do Problema de Uma Máquina 8

2.2 Diagrama de Gantt para o Problema de Uma Máquina 9

2.3 Exemplo do Problema de Máquinas Paralelas 10

2.4 Exemplo do Problema de *Job Shop* 10

2.5 Exemplo do Problema de *Flow Shop* 11

2.6 Diagrama de Gantt para problemas que adotam TPMDSP 13

2.7 Estrutura de Vizinhança com Movimentos de troca adjacente 28

3.1 Procedimento Proposto. Fonte: Adaptado de Wan e Yen [34] 38

3.2 Algoritmo para Determinar as Datas Ótimas de Início de Processamento 40

3.3 Inserção do primeiro *job* na seqüência do exemplo dado 41

3.4 Inserção do segundo *job* na seqüência do exemplo dado 43

3.5 Inserção do terceiro *job* na seqüência do Exemplo dado 43

3.6 Bloco movido até o ponto de mínimo 44

3.7 Estrutura de vizinhança com troca da ordem de processamento de dois *jobs* 45

3.8 Estrutura de vizinhança com realocação de *jobs* 45

3.9 Algoritmo para Construir uma Solução via GRASP 47

3.10 Algoritmo para Gerar uma Solução Inicial 47

3.11 Método Randômico de Descida. 48

3.12 Método da Descida. 48

3.13 Método VND. 49
3.14 Algoritmo *ILS* 50
3.15 Algoritmo *GRASP* aplicado ao PSUMAA 51

4.1 Gráfico de Gantt para o Cenário 1 55
4.2 Gráfico de Gantt para o Cenário 2 55
4.3 Gráfico de Gantt para o Cenário 3 56
4.4 Gráfico - Valor Padronizado $\times$ γ 58
4.5 Gráfico - Percentual Médio de Melhora $\times$ Número Máximo de Iterações 59
4.6 Gráfico - Percentual Médio de Melhora $\times$ Variação de k 60
4.7 Gráfico - Percentual Médio de Melhora $\times$ Variação de k 60
4.8 Gráfico - GAP médio $\times$ Variação de k_1 62

Lista de Tabelas

3.1 Dados do Exemplo 41

4.1 Dados do Cenário 1 53

4.2 Dados do Cenário 2 54

4.3 Resultados do Cenário 1 54

4.4 Resultados do Cenário 2 55

4.5 Dados do Cenário 3 55

4.6 Resultados do Cenário 3 56

4.7 Resultados obtidos pelo CPLEX 9.1 57

4.8 Resultados dos Testes para o VND 61

4.9 Resultados dos Testes com ILS para Determinar $IterMax6$ 63

4.10 Tabela de Parâmetros Utilizados 64

4.11 Resultados do Primeiro Conjunto de Testes 64

4.12 Resultados do Método ILS-VND-RT 65

4.13 Resultados do Método GRASP-VND-RT 65

Lista de Siglas

ADDOIP: Algoritmo de Determinação das Datas Ótimas de Início de Processamento.

AG: Algoritmos Genéticos.

BT: Busca Tabu.

EDD: *Earliest Due Date* – data de entrega mais cedo.

GRASP: *Greedy Randomized Adaptive Search Procedure* – Procedimento de busca adaptativa gulosa e randomizada.

ILS: *Iterated Local Search* – Busca Local Iterativa.

JIT: *Just in Time* – Filosofia de produção adotada por certas empresas onde tudo tem o momento certo para acontecer.

LRC: Lista Restrita de Candidatos.

MD: Método de Descida.

MRD: Método Randômico de Descida.

PLIM: Programação Linear Inteira Mista.

PPP: Problema de Programação da Produção.

PR: *Path Relinking* – Reconexão por Caminhos.

PSUMAA: Problema de Seqüenciamento de uma Máquina com penalidades por Antecipação e Atraso da Produção.

SA: *Simulated Annealing* - Recozimento simulado.

VND: *Variable Neighborhood Descent* – Método de Descida em Vizinhança Variável.

Capítulo 1

Introdução

Como é bem conhecido, a produção de bens sob encomenda vem sendo adotada por um grande número de empresas. Estas empresas buscam adaptar sua linha de produção às necessidades do cliente (da quantidade de produtos à data de entrega), de modo a atendê-lo da melhor forma possível. Uma das mais importantes decisões relativas ao chão de fábrica que estas empresas têm que tomar diz respeito à programação da produção.

O problema de programação da produção consiste em, dado um período do horizonte de planejamento (tipicamente uma semana), fazer a alocação de operações a máquinas e a programação dessas operações em cada máquina, ou seja, decidir a seqüência nas quais as operações devem ser processadas e em que momento elas devem ser realizadas no período de planejamento.

Entre os principais tipos de problemas de programação da produção encontra-se o de seqüenciamento em uma máquina com penalidades por antecipação e atraso na produção. Resolver este tipo de problema pode implicar em uma sensível redução dos custos gerados com a antecipação e com o atraso na produção. Há antecipação quando a produção é finalizada antes da data desejada para a entrega do produto, gerando-se estoques. Por outro lado, há atraso quando a produção é finalizada após a data desejada para a entrega do produto, incorrendo-se em multas. As datas de entrega podem ser comuns ou distintas a todas as operações de produção. Uma formulação mais geral para este tipo de problema considera uma janela de tempo, denominada janela de entrega, onde as operações que forem finalizadas dentro desse intervalo de tempo não implicam em custos adicionais à empresa.

Geralmente outras restrições são impostas a este tipo de problema. Uma dessas

restrições adicionais é com relação ao tempo de preparação da máquina. Esse tempo, denominado tempo de *setup*, inclui o tempo de preparar a máquina, o processo ou a oficina para a fabricação de determinados tipos de produtos.

Vários trabalhos encontrados na literatura têm seu foco em técnicas para a resolução deste problema. Este fato se deve ao enorme número de aplicações práticas existentes nas indústrias metalúrgicas, têxteis, de tintas, entre outras. Por exemplo, a fabricação de fios metálicos em uma indústria siderúrgica, onde o conjunto de laminadores são considerados como uma única máquina.

Nesta dissertação é estudado o problema de seqüenciamento de uma máquina com penalidades por atraso e antecipação da produção, com tempo de preparação da máquina dependente da seqüência de produção e com janelas de entrega. Trata-se de um problema combinatório de difícil resolução na otimalidade.

O problema pode ser representado adequadamente por meio de um modelo de programação matemática, entretanto os métodos de programação matemática, ditos exatos, em geral são capazes de resolver na otimalidade apenas problemas de pequenas dimensões, aquém da realidade prática. Como para sua resolução requerem um tempo computacional proibitivo para a maioria dos casos reais, a abordagem mais comum é o uso de métodos heurísticos. A apresentação de um modelo matemático também tem sua importância, primeiramente, porque através dele pode-se, em alguns casos, se obter a solução ótima do problema e, segundo, porque é possível fornecer o limite inferior para a solução ótima. Estas informações normalmente são utilizadas para avaliar e validar os métodos heurísticos.

Os métodos heurísticos, também denominados aproximados, são métodos capazes de encontrar boas soluções para o problema, mas sem estarem capacitados para garantir se esta solução é ótima ou não. Devido à sua capacidade de produzir, em geral, boas soluções em um tempo computacional reduzido, estes métodos são normalmente aplicados a problemas de dimensões mais elevadas, próximos à realidade. Na maioria das situações reais, encontrar uma boa solução já é suficiente.

A meta de produzir boas soluções em tempos computacionais cada vez mais reduzidos tornou-se mais realista a partir da reunião de conceitos das áreas de Otimização Combinatória e Inteligência Artificial, tornando possível a construção das chamadas melhores estratégias ou dos métodos "inteligentemente flexíveis", normalmente conhecidos como meta-heurísticas. Estas técnicas procuram dar uma certa inteligência aos métodos heurísticos, desta forma podendo escapar de ótimos locais. Devido a esses

fatores, vários autores têm dado ênfase à aplicação destes métodos na resolução de diversos tipos de problemas, entre eles, o problema de seqüenciamento de uma máquina com penalidades por antecipação e atraso.

Baseados nos fatores descritos anteriormente, são propostos aqui métodos baseados nas meta-heurísticas GRASP (*Greedy Randomized Adaptive Search Procedure* – Procedimento de busca adaptativa gulosa e randomizada) e ILS (*Iterated Local Search* – Busca Local Iterativa) para resolver o problema estudado.

1.1 Relevância do Trabalho

Determinar a melhor programação da produção é um objetivo a ser alcançado por qualquer empresa, em vista da redução de custos de produção e/ou estocagem que ela pode proporcionar. A redução no custo de produção pode ocorrer pela diminuição das multas por atraso na entrega dos produtos e a diminuição do custo de estocagem pode ser viabilizada evitando-se, tanto quanto possível, produzir antes da hora.

Considerando que cada empresa possui um cenário diferente, a utilização de sistemas computacionais genéricos, nem sempre compatíveis com a realidade da empresa, se torna inadequada. Algumas empresas chegam a utilizar planilhas eletrônicas como ferramentas de suporte para definir a programação da produção, mas sem se preocupar com sua otimização. Então o desenvolvimento de sistemas específicos voltados para a realidade da empresa e utilizando técnicas adequadas se torna útil para determinar a melhor programação da produção.

Ademais, o problema objeto de estudo, apesar de ter enorme aplicação prática, ainda não foi exaustivamente estudado pela literatura através das novas metodologias heurísticas de resolução.

1.2 Objetivos

Nesta seção são apresentados os principais objetivos desta dissertação.

1.2.1 Objetivo Geral

O objetivo geral desta dissertação é desenvolver um modelo matemático para representar o Problema de Seqüenciamento de uma Máquina com Penalidades por Antecipação e Atraso da Produção, com tempo de preparação de máquina dependente da seqüência de produção e com janelas de entrega, bem como desenvolver metodologias eficientes para sua resolução.

1.2.2 Objetivos Específicos

Os objetivos específicos desta dissertação são os seguintes:

(i) Fazer um levantamento bibliográfico sobre o problema estudado e problemas afins, bem como as diversas metodologias utilizadas para resolvê-los;

(ii) Desenvolver um modelo matemático para representar o problema;

(iii) Testar o modelo matemático em diversos cenários utilizando um método exato de resolução;

(iv) Criar um conjunto de problemas-teste do problema em questão;

(v) Desenvolver uma ou mais metodologias para resolver o problema;

(vi) Implementar um sistema computacional em linguagem C, baseado nas metodologias desenvolvidas para resolver o problema;

(vii) Validar o sistema computacional através da comparação com problemas resolvidos por meio do modelo matemático em problemas de pequeno porte;

(viii) Resolver os problemas-teste por meio do sistema computacional desenvolvido.

1.3 Estrutura do Trabalho

Esta dissertação está estruturada em 5 capítulos, os quais são descritos a seguir.

No Capítulo 1 é apresentado o contexto no qual está inserido o problema a ser resolvido, bem como destacada a importância do estudo do mesmo. São apresentados, ainda, os objetivos gerais e específicos desta dissertação.

No Capítulo 2 são apresentados alguns dos trabalhos mais relevantes da literatura sobre o problema estudado e afins. Este capítulo contém uma breve descrição sobre alguns desses trabalhos.

O Capítulo 3 apresenta a descrição detalhada do modelo matemático para representar o problema e as metodologias heurísticas desenvolvidas para resolver o mesmo.

O Capítulo 4 mostra como os problemas-teste foram gerados, bem como apresenta e analisa os resultados obtidos tanto através da modelagem matemática quanto das metodologias heurísticas desenvolvidas.

O Capítulo 5 conclui o trabalho e aponta sugestões para trabalhos futuros sobre o problema estudado.

Capítulo 2

Revisão Bibliográfica

Nos últimos anos, a preocupação com a satisfação dos clientes vem crescendo vertiginosamente. Vários fatores têm influenciado este crescimento, tais como o surgimento de novas empresas e a globalização. Tais fatores têm feito com que as empresas invistam cada vez mais em pesquisas em busca de uma linha de produção mais eficiente, dentro dos modernos conceitos da economia globalizada.

Arenales *et al.* [3] ressaltam a importância dos problemas na área de produção e mostram que tais problemas em geral são decompostos hierarquicamente em três níveis: estratégico, tático e operacional. O nível estratégico, no qual as decisões são de longo prazo, trata da escolha e do projeto do processo. Já o nível tático trata do planejamento das atividades. E por fim, o nível operacional, o qual é tratado nesta dissertação, controla as atividades diárias das ordens de produção provenientes do nível tático.

Este capítulo tem por finalidade apresentar alguns dos principais trabalhos relacionados ao problema de programação da produção(PPP), mais especificamente, ao problema de seqüenciamento em uma máquina com penalidades por antecipação e atraso da produção (PSUMAA), o qual é objeto de estudo desta dissertação. Inicialmente são detalhados os principais conceitos relativos ao PPP. Em seguida, são apresentados alguns modelos matemáticos propostos para o PSUMAA. São apresentados, ainda, diversos métodos utilizados durante os últimos anos para resolvê-lo.

2.1 O Problema de Programação da Produção

O PPP, como dito anteriormente, consiste em designar um conjunto de operações a máquinas e determinar a seqüência de processamento destas operações em cada máquina, bem como o instante de início e término do processamento de cada operação.

Como existem diversos tipos de ambientes de produção, a literatura nesta área é bem ampla (ARENALES *et al.* [3]).

Cada tipo de problema possui uma característica que o diferencia dos demais. Várias modelagens foram propostas para estes problemas e vários métodos foram criados para resolvê-los. Os principais elementos desta classe de problema são:

- Recursos: Um bem ou serviço, cuja disponibilidade é limitada ou não. Eles podem ser classificados em recuperáveis e não-recuperáveis. Alguns exemplos mais comuns de recursos são: máquinas, matérias-primas, mão-de-obra, etc.
- Tarefas ou Operações: Trabalho elementar cuja realização necessita de um certo número de unidades de tempo e/ou recursos.
- *Job*: representa uma seqüência conhecida de uma ou mais operações (tarefas) que representam uma seqüência tecnológica de fabricação de um produto. Os tempos de processamento também são associados às tarefas que compõem um *job*. Assim, num contexto de manufatura de produtos, um job pode representar a fabricação de um produto ou de um lote de família de produtos que possuem a mesma seqüência tecnológica de fabricação. Em algumas situações práticas, o conceito de tarefa coincide com o conceito de *job*, que é o caso, por exemplo, do problema de uma máquina. (CARVALHO [7])

Segundo Baker [4], para classificar a maioria dos modelos de seqüenciamento é necessário caracterizar os dados sobre os recursos e o comportamento das operações. Um modelo pode conter um recurso ou vários tipos de recursos. Os recursos podem ser especializados (cada recurso executa um único tipo de operação) ou paralelos (quando diversos recursos podem executar uma mesma operação).

Os modelos de seqüenciamento podem ser classificados em relação aos dados de entrada da seguinte forma: (i) Dinâmicos (quando os dados variam em função do tempo, ou seja, uma vez iniciada a execução das operações, os dados iniciais podem mudar) e (ii) Estáticos (os dados são fixos, ou seja, uma vez iniciada a execução das operações, os dados iniciais se mantém os mesmos).

2.1.1 Os Principais Tipos de Problemas de Programação da Produção

Entre os diversos tipos de problemas de programação da produção, Baker [4] e Arenales *et al.* [3] classificam como principais: (*i*) o problema de máquinas paralelas, (*ii*) o problema de *job shop*, (*iii*) o problema de *flow shop* e (*iv*) o problema de uma máquina. As seções seguintes descrevem cada um destes problemas. Como o foco desta dissertação é o problema de seqüenciamento em uma máquina, dá-se um destaque maior ao mesmo.

2.1.1.1 O Problema de Seqüenciamento em uma Máquina

Neste tipo de problema, todos os produtos são processados em uma única máquina especializada. Um esquema deste tipo de problema é apresentado na Fig. 2.1. Neste exemplo, os *jobs* 1 a n são processados em uma única máquina dando origem aos produtos 1 a n.

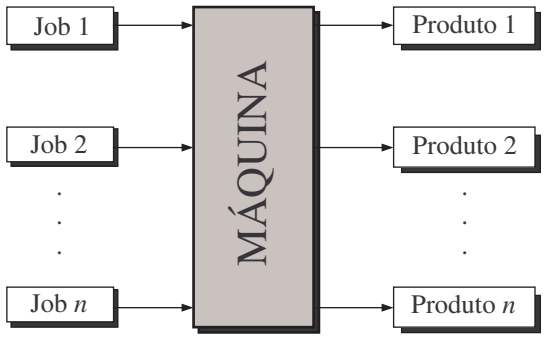

Figura 2.1: Exemplo do Problema de Uma Máquina

Além dos sistemas produtivos que operam com uma única máquina, os sistemas produtivos onde as estações de trabalho são dependentes umas das outras também podem ser representados por este tipo de problema.

Uma possível aplicação real deste problema é na fabricação de chapas ou fios metálicos, citado por Bustamante [6]. Neste caso, considera-se uma máquina como sendo uma seqüência de laminadores e cada *job* um conjunto de operações realizadas nesses laminadores que resulte em uma chapa com uma determinada espessura ou em um fio metálico com um determinado diâmetro. Para cada produto (chapa ou fio) fabricado com espessura ou diâmetro diferente, têm-se um conjunto de operações de

mesma natureza realizadas nos mesmos laminadores, mas que requerem tempos de processamento diferentes em cada um deles, configurando *jobs* diferentes.

Para representar graficamente as operações (*jobs*) em uma máquina ao longo do tempo normalmente é utilizado o Diagrama de Gantt, conhecido também como diagrama de barras. Este diagrama traz no eixo das ordenadas o recurso utilizado e no eixo das abscissas tem-se uma linha de tempo. Uma barra é então introduzida neste diagrama. As sub-divisões desta barra representam os instantes dos principais eventos da programação da produção. Geralmente, estes eventos são o início e o fim do processamento de cada operação e outras informações importantes na programação da produção, tais como: tempo ocioso da máquina, tempo de preparação da máquina, entre outros. Um exemplo é apresentado na Fig. 2.2.

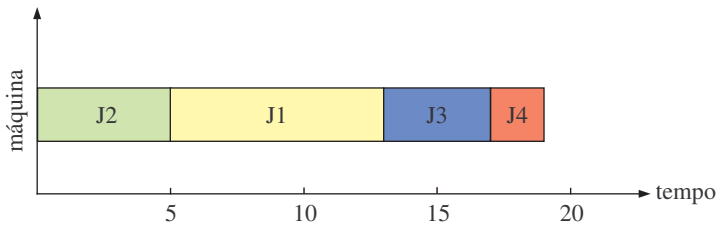

Figura 2.2: Diagrama de Gantt para o Problema de Uma Máquina

Na Fig. 2.2, o *job* 2 é o primeiro a ser processado e inicia seu processamento na data 0 e termina na data 5. O *job* 1, por sua vez, tem seu processamento iniciado na data 5 e finalizado na data 13.

2.1.1.2 Outros tipos de Problema de Programação da Produção

Os demais tipos de problemas de programação da produção conforme Baker [4] e Arenales *et al.* [3] são:

(a) *Máquinas Paralelas:* Alguns produtos passam por processos que podem ser realizados por uma máquina pertencente a um conjunto de máquinas idênticas ou não. As máquinas podem ser idênticas (possuem a mesma velocidade) ou diferentes (possuem diferentes velocidades). Um exemplo deste tipo de problema é apresentado na Fig. 2.3. Neste exemplo, o *job* 1 deve ser processado, primeiramente, na máquina 1 ou 2 (que formam o primeiro conjunto de máquinas) e posteriormente na máquina 3, 4 ou 5 (que formam o segundo conjunto de máquinas).

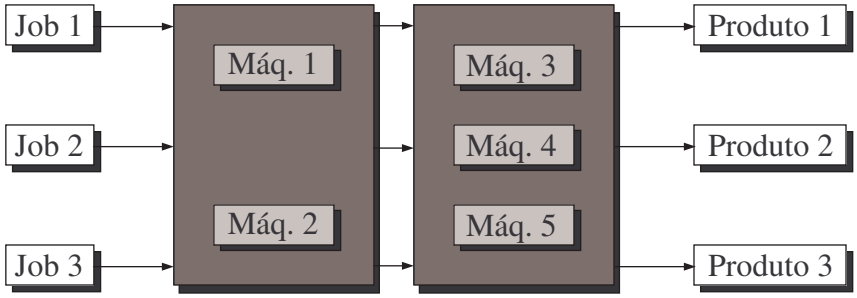

Figura 2.3: Exemplo do Problema de Máquinas Paralelas

(b) *Job Shop:* Um conjunto de máquinas executa um determinado número de operações sobre um conjunto de *jobs*, cada *job* possui sua própria seqüência tecnológica de produção. Um exemplo deste tipo de problema é apresentado na Fig. 2.4. Neste exemplo, o *job* 1 deve ser processado nas máquinas 2, 1 e 3, nesta ordem. Por sua vez, o *job* 2 deve ser processado nas máquinas 2 e 3, nesta ordem.

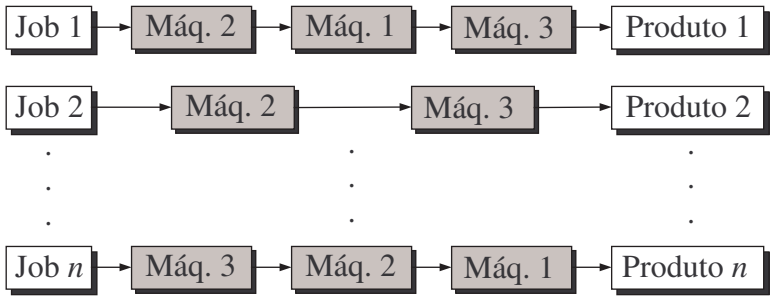

Figura 2.4: Exemplo do Problema de *Job Shop*

(c) *Flow shop:* É um caso particular do problema de *Job Shop*. Neste caso, todos os *jobs* possuem a mesma seqüência tecnológica de produção. Um exemplo deste tipo de problema é apresentado na Fig. 2.5. Neste exemplo, todos os *jobs* devem ser processados nas máquinas 2, 1 e 3, nesta ordem.

2.1.2 Os Medidores de Desempenho Não Regulares

Os critérios de otimização, chamados de medidores de desempenho, são classificados em regulares e não regulares. Um medidor de desempenho Z é dito regular se o objetivo do problema de seqüenciamento é minimizar Z e Z aumenta somente se no mínimo uma das datas de término de uma tarefa aumentar (BAKER [4]).

Os critérios de otimização regulares mais comuns são: minimizar o instante

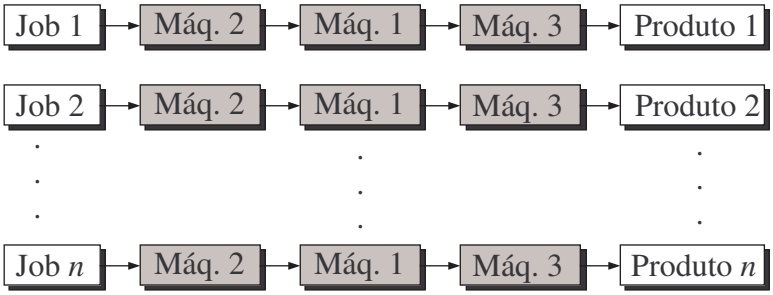

Figura 2.5: Exemplo do Problema de *Flow Shop*

de término da última operação processada (*Makespan*), minimizar a demora máxima, minimizar o tempo médio de fluxo, entre outros.

Os primeiros modelos de programação da produção utilizavam em sua maioria a minimização do *makespan* como critério de decisão (PINEDO [28]). Com o passar dos anos, novos critérios passaram a ser explorados, tendo em vista a crescente importância para o mercado de fatores ligados a determinação e ao cumprimento de datas de entrega (KIM [20]). Para acompanhar essa tendência, foram elaborados modelos cujos critérios de decisão relacionavam-se com o atraso, como por exemplo, minimizar o número de operações atrasadas, o somatório dos atrasos de todas as operações da programação ou minimizar o atraso médio de todas as operações.

Estes critérios, em particular, levam em consideração as datas de entrega; contudo ignoram a conseqüência dos *jobs* que são finalizados antes destas datas. Porém, a ênfase tem mudado com o corrente interesse em produções baseadas na filosofia *JIT*, que adere a noção de que antecipação, assim como atraso da produção, devem ser desencorajados. Em um ambiente de programação *JIT*, os produtos que são finalizados antecipadamente devem ser mantidos em estoque até a sua data de entrega, enquanto produtos que são finalizados tardiamente podem incorrer em penalidades contratuais, em perda de credibilidade da empresa e conseqüente perda de clientes (BAKER e SCUDDER [5]).

Baker e Scudder [5] apresentam uma revisão sobre os problemas de programação que adotam medidores de desempenho não regulares. Neste trabalho pode claramente ser visto que tais problemas diferenciam-se basicamente em relação a duas características: a data de entrega desejada e os custos de antecipação e atraso.

Em algumas formulações do problema de seqüenciamento com penalidades por atraso e antecipação da produção a data de entrega é conhecida a priori, enquanto

que em outros o problema é otimizar a data de entrega e a seqüência de operações simultaneamente. Uma formulação mais geral adota datas distintas de entrega.

Determinadas formulações adotam datas de entrega comuns a todos os *jobs* (quando todos os *jobs* devem ser finalizados em uma mesma data, por exemplo, para permitir a montagem do produto final). Uma formulação mais geral adota datas distintas de entrega (quando cada *job* deve ser finalizado em uma data específica).

Em outras formulações, adota-se uma janela de entrega ao invés de uma data fixa de entrega, ou seja, os produtos que forem finalizados dentro desta janela de tempo não terão penalidades por antecipação nem por atraso da produção.

Bustamante [6], de acordo com o trabalho de Baker e Scudder [5], classifica os problemas com medidores de performance não regulares, em relação aos custos unitários de antecipação e atraso, em:

- Problemas com custos dependentes: quando as penalidades por antecipação e atraso são dependentes das operações, assumindo valores diferentes para cada operação.
- Problemas com custos iguais: quando as penalidades por antecipação são iguais às penalidades por atraso.
- Problemas com custos diferentes: quando as penalidades por antecipação são diferentes das penalidades por atraso.

Esta dissertação adota como critério de otimização a minimização dos custos de antecipação e atraso da produção com janelas de entrega e custos dependentes das operações.

2.1.3 O Tempo de Preparação das Máquinas

O tempo de preparação das máquinas, ou tempo de *setup*, inclui o tempo de preparar a máquina, o processo ou a oficina para a fabricação de produtos. Isto inclui o tempo para obtenção das ferramentas, posicionamento dos materiais a serem usados no trabalho, processos de limpeza, preparação e ajuste das ferramentas e inspeção de materiais.

Muitas pesquisas em programação da produção desconsideram o tempo de preparação das máquinas ou então os incluem no tempo de processamento das ope-

rações. Isto simplifica a análise das aplicações, porém afeta a qualidade da solução quando tais tempos têm uma variabilidade relevante em função da ordenação das operações nas máquinas.

Existem dois tipos de problemas que requerem que o tempo de preparação das máquinas seja explícito ou separado do tempo de processamento das operações: tempos de preparação das máquinas independentes da seqüência de produção, os quais dependem somente da operação a ser processada, e tempos de preparação das máquinas dependentes da seqüência de produção, que dependem tanto da operação a ser processada quanto daquela que foi processada imediatamente antes na mesma máquina (ALLAHVERDI *et al.* [2]).

Tempos de preparação de máquinas dependentes da seqüência de produção são comumente encontrados em empresas que possuem um *mix* de produto muito alto ou onde uma determinada máquina realiza diferentes tipos de operações.

Este tipo de abordagem é bastante útil na indústria química. Neste tipo de indústria quando se deseja produzir diferentes componentes químicos em um sistema único, requer-se uma determinada purificação (limpeza) dos equipamentos, visando garantir níveis toleráveis de impureza(BAKER [4]). Outras aplicações são na indústria gráfica, têxtil, de celulose, farmacêutica, alimentícia e metalúrgica (ALLAHVERDI *et al.* [2]).

Nesta dissertação consideram-se tempos de preparação das máquinas dependentes da seqüência de produção (TPMDSP). A representação gráfica desta situação é mostrada na Fig. 2.6.

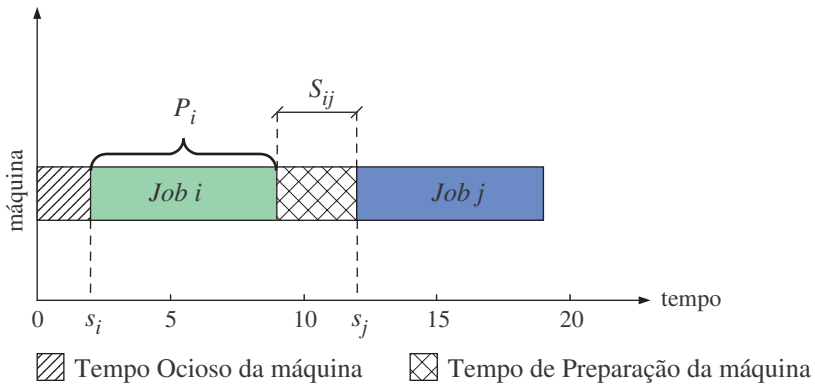

Figura 2.6: Diagrama de Gantt para problemas que adotam TPMDSP

Na Fig. 2.6, s_i é a data de início de processamento do *job* i, P_i é o tempo de processamento do *job* i e S_{ij} é o tempo de preparação de máquina necessário quando

o *job j* é processado imediatamente após o *job i*.

2.2 Modelagem Matemática

Segundo Golbarg e Luna [16], modelos são representações simplificadas da realidade que preservam, para determinadas situações e enfoques, uma equivalência adequada.

A partir da observação de fenômenos, processos ou sistemas, que podem ser físicos, químicos, biológicos, econômicos, buscam-se leis que os regem. Essas leis, se passíveis de serem descritas por relações matemáticas, dão origem aos modelos matemáticos. Em geral, para formular um modelo matemático, simplificações razoáveis do sistema ou problema real devem ser consideradas (em diferentes níveis) e a sua validação depende da solução obtida ser coerente com o contexto original. Com isso, o modelo matemático é uma representação simplificada (abstração) do problema real (ARENALES *et al.* [3]).

A programação matemática trata de problemas de decisão em espaços de dimensões fintas e faz uso dos modelos matemáticos. Variáveis são definidas e relações matemáticas entre essas variáveis são estabelecidas de forma a descrever o comportamento do sistema (função objetivo e restrições). (ARENALES *et al.* [3]).

Para uma melhor classificação dos problemas, Goldbarg e Luna [16] subdividem a programação matemática em algumas sub-áreas, que são: (*i*) Programação Linear (quando todas as variáveis são contínuas e a função objetivo e as restrições apresentam comportamento linear); (*ii*) Programação Não-Linear (quando ocorre algum tipo de não-linearidade, seja na função objetivo ou em qualquer uma de suas restrições); (*iii*) Programação Inteira (quando qualquer variável não puder assumir valores contínuos, ficando condicionada a assumir valores discretos) e (*iv*) Programação Linear Inteira Mista (neste caso há uma mistura da programação linear com a programação inteira, ou seja, algumas das variáveis assumem valores discretos, mas apresentam comportamento linear).

Geralmente, para cada uma das subdivisões da programação matemática são desenvolvidos métodos matemáticos e computacionais para resolução dos modelos (isto é, são determinados valores para as variáveis, produzindo soluções, que dependem dos dados do problema). Estes métodos são amplamente utilizados na resolução de problemas industriais reais, daí muitas empresas de informática propõem implementações

comerciais destes métodos. Alguns *softwares* comerciais de otimização disponíveis no mercado são o LINDO e o LINGO da LINDO SYSTEMS®, o GLPK da GNU®, o XPRESS da Dash Optimization®, o CPLEX da ILOG®, entre outros.

Após resolvido o modelo matemático, o passo seguinte é a validação do modelo, isto é, verificar se as soluções obtidas pela resolução do mesmo, para diversas situações alternativas (cenários), são compatíveis com a realidade.

2.2.1 Modelos Matemáticos para o PPP

Os primeiros modelos a tratarem do PPP datam das décadas de 50 e 60. Destes modelos, destacam-se os modelos propostos por Manne [25] e Wagner [33]. Ambos são modelos de programação linear inteira mista (PLIM), propostos inicialmente para o problema de *job shop*.

O modelo de Wagner [33] usa o problema clássico de alocação de tarefas para assinalar *jobs* à posições na seqüência de produção. Já o modelo de Manne [25] utiliza um par de restrições dicotômicas para controlar a ordem relativa dos jobs dentro da seqüência de produção.

Vários modelos baseados em Wagner [33] e Manne [25] são encontrados na literatura para os diversos problemas de programação da produção. Stafford-Jr. *et al.* [32] faz uma avaliação comparativa entre diversos modelos baseados nessas formulações para o problema de *flow shop* com o objetivo de minimizar a data de término da última operação a ser processada.

Bustamante [6] apresenta em seu trabalho dois modelos para o PSUMAA com tempo de preparação da máquina dependente da seqüência de produção, sendo um desses modelos baseados em Manne [25] e o outro baseado em Wagner [33].

O primeiro desses modelos, baseado em Manne [25], é descrito a seguir.

Os parâmetros de entrada do problema são:

- n: número de *jobs* a serem processados.
- P_i: tempo de processamento do *job* i.
- D_i: data desejada para o término do *job* i (data de entrega).
- S_{ij}: tempo de preparação da máquina para processar o *job* j depois do *job* i.

- α_i: custo de antecipação da produção do produto i por unidade de tempo (manutenção do estoque).
- β_i: custo de atraso da produção do produto i por unidade de tempo.
- M: um valor muito grande.

E as variáveis de decisão são:

- s_i: data de início do processamento do *job* i.
- y_{ij}: variável que determina a seqüência de produção, se y_{ij}=1 o *job* j é processado depois do *job* i e 0 caso contrário.
- e_i: tempo de antecipação do *job* i.
- t_i: tempo de atraso do *job* i.

Desta forma, tem-se o seguinte modelo de PLIM:

$$\text{minimizar} \quad Z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{2.1}$$

$$\text{sujeito a:} \quad s_j - s_i - (M + S_{ij}) y_{ij} \geq P_i - M \quad \forall\, i, j = 1, 2, \ldots, n \;\; e \;\; i \neq j \tag{2.2}$$

$$y_{ij} + y_{ji} \leq 1 \quad \forall\, i, j = 1, 2, \ldots, n \;\; e \;\; i \neq j \tag{2.3}$$

$$s_i + P_i + e_i - t_i = D_i \quad \forall\, i = 1, 2, \ldots, n \tag{2.4}$$

$$s_i \geq 0 \quad \forall\, i = 1, 2, \ldots, n \tag{2.5}$$

$$e_i \geq 0 \quad \forall\, i = 1, 2, \ldots, n \tag{2.6}$$

$$t_i \geq 0 \quad \forall\, i = 1, 2, \ldots, n \tag{2.7}$$

$$y_{ij} \in \{0, 1\} \quad \forall\, i, j = 1, 2, \ldots, n \tag{2.8}$$

A função objetivo, representada pela equação (2.1), tem como critério de otimização a minimização dos custos de antecipação e atraso. As restrições (2.2) definem a seqüência de operações sobre o recurso (máquina) utilizado. As restrições (2.3) garantem que se o *job* j é processado depois do *job* i (y_{ij}=1) o valor de y_{ji} será 0, e vice-versa. As restrições (2.4) definem os valores do atraso e da antecipação de acordo com a data desejada para o término do *job* i, caso estes existam. As restrições (2.5) a (2.8) definem o domínio das variáveis do problema.

Este modelo é válido somente se os dados do problema satisfizerem a desigualdade triangular, ou seja, se a condição (2.9) for satisfeita.

$$S_{ik} < S_{ij} + S_{jk} + P_j \;\; \forall (i,j,k) = 1,2,\ldots,n \tag{2.9}$$

O segundo modelo apresentado por Bustamante [6], baseado no modelo de Wagner [33], é descrito a seguir.

Os dados de entrada deste modelo, além dos já citados anteriormente para o primeiro modelo, são:

- I: conjunto de todos os n *jobs* a serem processados.
- K: conjunto das possíveis posições k na seqüência de produção.
- α: custo de antecipação da produção por unidade de tempo (manutenção do estoque).
- β: custo de atraso da produção por unidade de tempo.

As variáveis de decisão consideradas são:

- s_k: data de início do processamento da posição k da seqüência da produção.
- z_{ik}: determina se o *job* i foi alocado na posição k da seqüência de produção. Se $z_{ik} = 1$ o *job* foi alocado, e $z_{ik} = 0$ caso contrário.
- w_{ijk}: determina se o *job* i foi alocado em uma posição subseqüente ao *job* j. Se $w_{ijk} = 1$, o *job* i foi alocado na posição k e o *job* j na posição $k+1$ e 0 caso contrário.
- e_k: tempo de antecipação da produção da posição k da seqüência da produção;
- t_k: tempo de atraso da produção da posição k da seqüência da produção.

O segundo modelo proposto por Bustamante [6] é representado pelas equações (2.10) a (2.21).

$$\text{minimizar} \quad Z = \sum_{k \in K} (\alpha \; e_k + \beta \; t_k) \tag{2.10}$$

Sujeito às seguintes restrições:

$$s_{k+1} - s_k - \sum_{i \in I} P_i z_{ik} - \sum_{i \in I} \sum_{j \in I,\, i \neq j} S_{ij} w_{ijk} \geq 0, \qquad \forall\, k \in K,\ k < |K| \tag{2.11}$$

$$\sum_{i \in I} z_{ik} = 1, \qquad \forall\, k \in K \tag{2.12}$$

$$\sum_{k \in K} z_{ik} = 1, \qquad \forall\, i \in I \tag{2.13}$$

$$w_{ijk} - z_{ik} - z_{j,\,k+1} \geq -1, \qquad \forall\, i, j \in I,\ i \neq j,\ k \in K,\ k < |K| \tag{2.14}$$

$$\sum_{i \in I} \sum_{j \in I,\, i \neq j} \sum_{k \in K,\, k < |K|} w_{ijk} = |K| - 1, \tag{2.15}$$

$$s_k + \sum_{i \in I} P_i z_{ik} + e_k - t_k - \sum_{i \in I} D_i z_{ik} = 0, \qquad \forall\, k \in K \tag{2.16}$$

$$s_k \geq 0, \qquad \forall\, k \in K \tag{2.17}$$

$$e_k \geq 0, \qquad \forall\, k \in K \tag{2.18}$$

$$t_k \geq 0, \qquad \forall\, k \in K \tag{2.19}$$

$$z_{ik} \in \{0, 1\}, \qquad \forall\, i \in I, k \in K \tag{2.20}$$

$$w_{ijk} \in \{0, 1\}, \qquad \forall\, i, j \in I,\ k \in K \tag{2.21}$$

A função objetivo, representada pela equação (2.10), assim como no modelo anterior, tem como objetivo a minimização dos custos de antecipação e atraso da produção. As restrições (2.11) garantem que a operação na posição $k+1$ só poderá iniciar após o término da operação na posição anterior k e a preparação da máquina (S_{ij}) ter sido executada. As restrições (2.12) e (2.13) garantem que, respectivamente, cada posição k deve ser ocupada por exatamente um *job* i e cada *job* i só pode ocupar uma posição k na seqüência de produção. As restrições (2.14) e (2.15) determinam se o *job* i é alocado na posição k, imediatamente anterior à posição $k+1$ ocupada pelo *job* j. As restrições (2.15) garantem que a troca do *job* i para o *job* j só será considerada nas restrições (2.11) se o *job* i estiver alocado na posição k e o *job* j estiver alocado na posição $k+1$. As restrições (2.16) garantem a existência de antecipação e atraso em relação a data desejada para o término do *job* i. As restrições (2.17) a (2.21) definem o domínio das variáveis.

Neste segundo modelo, como os custos dependem da posição do *job* i na seqüên-

cia de produção, que é uma incógnita, torna-se necessária a criação de novas variáveis. Essas novas variáveis devem ser multiplicadas pelas variáveis e_k e t_k já existentes, tornando a função objetivo não linear. Desta forma, o autor considerou custos de atraso e antecipação diferentes, mas únicos para todos os *jobs*.

Os dois modelos propostos por Bustamante [6] foram implementados e resolvidos através do *software* de otimização GLPK 4.8, utilizando-se problemas-teste gerados pelo autor. Observou-se que os dois modelos propostos apresentaram desempenho semelhante e eram eficazes para problemas com até 10 *jobs*.

Stafford-Jr *et al.* [32], ao contrário de Bustamante [6], verificaram que os modelos baseados na formulação de Wagner [33], neste caso para o problema de *flow shop* com o objetivo de minimizar a data mais tarde de término de todas as operações, têm um melhor desempenho computacional que os modelos baseados na formulação de Manne [25].

2.3 Métodos de Resolução

Os problemas de otimização podem ser classificados, de acordo com a dificuldade de resolução do problema, em fáceis e difíceis (esses últimos também conhecidos como NP-Difíceis).

Os problemas NP-difíceis são aqueles para os quais não existem algoritmos que os resolvam em tempo polinomial.

Para a resolução dos problemas NP-difíceis são utilizadas diversas técnicas, tais como, relaxações, enumerações e heurísticas.

As enumerações são definidas como métodos exatos, pois tendem a garantir se a solução é ótima ou não. Esses métodos normalmente ficam limitados a problemas de pequeno porte, pois requerem um grande esforço computacional.

As técnicas heurísticas são técnicas que procuram boas soluções (próximas à solução ótima) a um custo computacional razoável, sem, no entanto, estarem capacitadas a garantir se a solução encontrada é ótima ou não, bem como garantir quão próximo uma determinada solução está da solução ótima.

Os problemas de programação da produção pertencem à classe NP-difícil (ARENALES *et al.* [3] e GAREY e JOHNSON [13]) e a complexidade para a resolução desses problemas cresce com o número de operações, de restrições e de variáveis discretas en-

volvidas no processo.

O PSUMAA com tempo de preparação de máquina dependente da seqüência de produção e janelas de entrega, tema desta dissertação, é classificado como fortemente NP-difícil, como observa Pinedo [28], Liaw [24], Hino *et al.* [19], Feldmann e Biskup [10], Baker e Scudder [5], Wan e Yen [34] e Sourd [30].

Nos últimos anos, vários métodos exatos e heurísticos têm sido propostos para a resolução dos problemas de programação da produção. Alguns destes métodos são apresentados nas seções seguintes.

2.3.1 Métodos Exatos

Como dito anteriormente, os métodos exatos são capazes de garantir se a solução encontrada é ótima ou não. Uma solução é dita ótima se, sob um certo critério de otimização, não existir nenhuma outra solução do problema com um valor melhor que ela.

Os principais métodos exatos são os métodos de enumeração. Estes métodos consistem em organizar uma busca no conjunto de soluções viáveis do problema (soluções que não desrespeitam nenhuma restrição do problema) de maneira que este conjunto é todo vasculhado e uma solução ótima é armazenada.

Entre os métodos de enumeração cita-se, por exemplo, o método *Branch-and-Bound*. O método *Branch-and-Bound* baseia-se na idéia de desenvolver uma enumeração inteligente de pontos candidatos à solução ótima inteira de um problema. O termo *branch* refere-se ao fato de que o método efetua partições do espaço de solução. O termo *bound* ressalta que a prova da otimalidade da solução utiliza-se de limites calculados ao longo da enumeração. (GOLDBARG e LUNA [16])

A utilização deste método permite resolver problemas com dimensões maiores do que aqueles resolvidos por enumeração completa, mas certamente haverá uma dimensão acima da qual o problema torna-se intratável computacionalmente.

Este método tem sido largamente utilizado na resolução de diversos tipos de problemas, entre eles o PSUMAA. Entre os métodos baseados nesta técnica, pode-se citar os trabalhos de Liaw [24] e Li [23](PSUMAA sem tempo ocioso de máquina), Chang [8] (PSUMAA com datas distintas de entrega) e Rabadi *et al.* [29] (PSUMAA com datas comuns de entrega e tempo de preparação da máquina dependente da seqüência de produção). Como estes problemas são NP-Difíceis a abordagem mais comum

para resolvê-los é através de métodos heurísticos, os quais são apresentados na seção seguinte.

2.3.2 Métodos Heurísticos

As técnicas heurísticas vêm sendo bastante utilizadas nos últimos anos, pois na maioria das situações reais encontrar uma boa solução é suficiente, ao invés da melhor solução, a qual somente poderá ser encontrada após um considerável esforço computacional. O desafio é produzir, em tempo reduzido, soluções cujo valor do critério de otimização esteja tão próximo quanto possível do valor da solução ótima.

Esta meta tornou-se mais realista a partir da reunião de conceitos das áreas de Otimização e Inteligência Artificial, viabilizando a construção das chamadas melhores estratégias ou dos métodos "inteligentemente flexíveis", comumente conhecidas como meta-heurísticas.

As técnicas meta-heurísticas que são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico. Esta técnica procura dar uma certa inteligência aos métodos heurísticos, desta forma podendo escapar de ótimos locais.

Dentre os procedimentos enquadrados como meta-heurísticas que surgiram ao longo das últimas décadas, destacam-se: Algoritmos Genéticos (GOLDBERG [17]), Busca Tabu (GLOVER [14]), *Simulated Annealing* (KIRKPATRICK *et al.* [21]), GRASP (FEO e RESENDE [11]), VND (MLADENOVIC e HANSEN [27]), ILS (GLOVER e KOCHENBERGER [15]), entre outros.

Nas seções seguintes são apresentados os conceitos dos principais métodos heurísticos e meta-heurísticos encontrados e algumas aplicações destes ao PPP.

2.3.2.1 Heurísticas Construtivas

As heurísticas construtivas tem por objetivo construir uma solução, elemento por elemento. A forma de escolha de cada elemento a ser inserido a cada passo varia de acordo com a avaliação adotada, a qual, por sua vez, depende do problema abordado. Nas heurísticas clássicas, os elementos candidatos são geralmente ordenados segundo uma função gulosa, que estima o benefício da inserção de cada elemento, e somente o

"melhor" elemento é inserido a cada passo. Estes tipos de heurísticas são geralmente utilizadas para a construção da solução inicial do problema.

Uma heurística construtiva muito comumente utilizada para o PSUMAA é a EDD (*Earliest Due Date*), que seqüencia as operações em ordem não decrescente das datas de entrega desejadas.

Mazzini e Armentano [26] apresentam uma heurística construtiva para o PSUMAA com datas de disponibilidade e entrega distintas. Esta heurística construtiva é dividida em três partes: Ordenação, Viabilidade e Atualização dos tempos ociosos. Na etapa de ordenação, os *jobs* são seqüenciados em ordem não decrescente do valor de u_i, definido como se segue.

$$u_i = \begin{cases} R_i, & \text{se } R_i + P_i > D_i; \\ D_i - P_i, & \text{se } R_i + P_i \leq D_i. \end{cases} \tag{2.22}$$

onde: R_i é a data de disponibilidade do *job* i; D_i e P_i são definidos na seção 2.2.1.

Depois de ordenados os *jobs* são inseridos, um a um, dentro de intervalo preferencial, que corresponde a janela de tempo $[u_i, u_i + P_i]$, onde incorre o menor custo ao *job* i, formando a seqüência de produção. O procedimento de viabilidade garante que não haverá sobreposições entre *jobs*, ou seja, dois *jobs* sendo processados ao mesmo tempo. Quando ocorre sobreposições, os *jobs* inseridos são movidos para a direita ou para a esquerda de acordo com o custo. Na última etapa são inseridos os tempos ociosos entre os *jobs*.

Outra forma de construir uma solução inicial é construí-la de maneira aleatória. A grande vantagem deste método é que ele é de fácil implementação. A desvantagem é baixa qualidade da solução final produzida, o que requerirá um esforço maior na fase de refinamento (SOUZA [31]).

2.3.2.2 Heurísticas de Refinamento

As heurísticas de refinamento em problemas de otimização, também chamadas de técnicas de busca local, constituem uma família de técnicas baseadas na noção de vizinhança (SOUZA [31]). A vizinhança de uma solução v é o conjunto de soluções v' que diferem de v por um movimento.

Em linhas gerais, essa classe de heurísticas parte de uma solução inicial qualquer (a qual pode ser obtida por uma heurística construtiva ou então gerada aleatoria-

mente) e caminha, a cada iteração, de vizinho para vizinho de acordo com a definição de vizinhança adotada. A definição de vizinhança é crucial em uma heurística de refinamento. De uma solução v do espaço de soluções deve ser sempre possível atingir qualquer outra solução em um número finito de passos, utilizando um determinado tipo ou tipos de movimentos.

Os métodos de busca local mais conhecidos são: Método da Descida/Subida, Método primeiro de Melhora e Método Randômico de Descida/Subida.

O Método de Descida/Subida parte de uma solução inicial qualquer e a cada passo são analisados todos os seus possíveis vizinhos, movendo-se somente para aquele que representar uma melhora no valor atual da função de avaliação. Desta forma, o método pára quando um ótimo local é encontrado.

O Método Primeiro de Melhora é uma simplificação do método anterior, neste caso a exploração da vizinhança é interrompida quando um vizinho melhor é encontrado.

O Método Randômico de Descida é outro método que evita a busca exaustiva na vizinhança de uma solução. Este método consiste em analisar um vizinho qualquer e o aceitar se ele for estritamente melhor que a solução corrente; não o sendo, a solução corrente permanece inalterada e um outro vizinho é gerado e analisado. O procedimento é interrompido após um número fixo de iterações sem melhora no valor da melhor solução obtida até então. Como neste método não é feita a exploração de toda a vizinhança da solução corrente, não há garantia de que a solução final seja um ótimo local (SOUZA [31]).

2.3.3 Meta-heurísticas

As meta-heurísticas se diferenciam entre si basicamente pelo mecanismo usado para escapar de ótimos locais. Elas se dividem em duas categorias, de acordo com o princípio usado para explorar o espaço de soluções: busca local e busca populacional.

Nas meta-heurísticas baseadas em busca local, a exploração do espaço de soluções é feita por meio de movimentos, os quais são aplicados a cada passo sobre a solução corrente, gerando outra solução promissora em sua vizinhança. Alguns exemplos de meta-heurísticas baseadas neste princípio são: Busca Tabu (BT), *Simulated Annealing* (SA), Método de Pesquisa em Vizinhança Variável e ILS (*Iterated Local Search* - Busca Local Iterativa).

Os métodos baseados em busca populacional, por sua vez, consistem em manter um conjunto de boas soluções e combiná-las de forma a tentar produzir soluções ainda melhores. Seus principais exemplos são: Algoritmos Genéticos (AG) e Algoritmos Meméticos.

Nas seções seguintes são apresentadas as principais metodologias meta-heurísticas e algumas aplicações destas na resolução dos problemas de programação da produção.

2.3.3.1 Busca Tabu

A Busca Tabu é um método de busca local que consiste em explorar o espaço de soluções movendo-se de uma solução para outra que seja seu melhor vizinho. Esta estratégia, juntamente com uma estrutura de memória para armazenar as soluções geradas (ou características dessas) possibilita escapar dos ótimos locais. (GLOVER e KOCHENBERGER [15])

As idéias por trás deste método podem ser resumidas da seguinte forma: partindo de uma solução inicial v_0, o algoritmo de BT move, a cada iteração, da solução corrente v para seu melhor vizinho v^*, mesmo que a solução v^* seja pior que a solução corrente v, até que um critério de parada seja satisfeito. (WAN e YEN [34]).

O critério de escolher o melhor vizinho é utilizado para escapar de ótimos locais. Esta estratégia, entretanto, pode fazer com que o algoritmo cicle, isto é, que retorne a uma solução já gerada anteriormente.

De forma a evitar que isto ocorra, existe uma lista tabu L, a qual é uma lista de movimentos proibidos. A lista tabu clássica contém os movimentos reversos aos últimos $|L|$ movimentos realizados (onde $|L|$ é um parâmetro do método) e funciona como uma fila de tamanho fixo, isto é, quando um novo movimento é adicionado à lista, o mais antigo sai. O tamanho da lista deve ser suficientemente grande para prevenir ciclos e suficientemente pequena para não proibir muitos movimentos, fazendo com que o método termine antecipadamente (WAN e YEN [34]).

Adicionalmente, um critério de aspiração é definido para tratar as situações onde os movimento são tabus. Se um movimento corrente é tabu, mas satisfaz o critério de aspiração, seu status tabu é cancelado e torna-se um movimento permitido.(WAN e YEN [34])

2.3.3.2 Algoritmos Genéticos

Algoritmos Genéticos (AG) é uma meta-heurística que se fundamenta em uma analogia com processos naturais de evolução, nos quais, dada uma população, os indivíduos com características genéticas melhores têm maiores chances de sobrevivência e de produzirem filhos cada vez mais aptos, enquanto indivíduos menos aptos tendem a desaparecer.

Nos métodos baseados em AG, cada cromossomo (indivíduo da população) está associado a uma solução do problema e cada gene está associado a uma componente da solução. Um mecanismo de reprodução, baseado em processos evolutivos, é aplicado sobre a população com o objetivo de explorar o espaço de busca e encontrar melhores soluções para o problema. Cada indivíduo é avaliado por uma certa função de aptidão, a qual mensura seu grau de adaptação ao meio. Quanto maior o valor da função de aptidão, melhor o indivíduo está adaptado ao meio.

Um AG inicia sua busca com uma população $\{v_1^0, v_2^0, \ldots, v_n^0\}$, normalmente aleatoriamente escolhida, a qual é chamada de população no tempo 0.

O procedimento principal é um processo de repetição que cria uma população $\{v_1^{t+1}, v_2^{t+1}, \ldots, v_n^{t+1}\}$ no tempo $t+1$ a partir de uma população do tempo t. Para atingir esse objetivo, os indivíduos da população do tempo t passam por uma fase de reprodução, a qual consiste em selecionar indivíduos para operações de recombinação e/ou mutação.

Gerada a nova população do tempo $t+1$, define-se a população sobrevivente, isto é, as n soluções que integrarão a nova população. A população sobrevivente é definida pela aptidão dos indivíduos.

O método termina, em geral, quando um certo número de populações é gerado ou quando a melhor solução encontrada atinge um certo nível de aptidão ou ainda, quando não há melhora após um certo número de iterações. (SOUZA [31])

2.3.3.3 ILS

O método ILS (*Iterated Local Search* – Busca Local Iterativa) é baseado na simples idéia de que um procedimento de busca local pode ser melhorado gerando-se novas soluções de partida, as quais são obtidas por meio de perturbações na solução ótima local. A perturbação precisa ser suficientemente forte para permitir que a busca

local explore diferentes soluções, mas também fraca o suficiente para evitar um reinício aleatório.

O método ILS é, portanto, um método de busca local que procura focar a busca não no espaço completo de soluções, mas em um pequeno subespaço definido por soluções que são ótimas locais de determinado procedimento de otimização.

2.3.3.4 GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure* – Procedimento de busca adaptativa gulosa e randomizada) é um método iterativo que consiste de duas fases: uma fase de construção, na qual uma solução é gerada, elemento a elemento, de forma parcialmente gulosa, e de uma fase de busca local, na qual um ótimo local na vizinhança da solução construída é pesquisado. A melhor solução encontrada ao longo de todas as iterações GRASP realizadas é retornada como resultado.

2.3.3.5 Outras Meta-heurísticas

Simulated Annealing (SA) é uma técnica de busca local probabilística que se fundamenta em uma analogia com a termodinâmica, ao simular o resfriamento de um conjunto de átomos aquecidos, operação conhecida como recozimento. Esta técnica começa sua busca a partir de uma solução inicial qualquer. O procedimento principal consiste em um procedimento de repetição que gera aleatoriamente, em cada iteração, um único vizinho v_0 da solução corrente v. Para poder escapar de ótimos locais este método aceita, dentro de uma certa probabilidade, soluções de piora.

O método Algoritmos Meméticos é uma variação de Algoritmos Genéticos, consistindo no refinamento dos indivíduos antes de se submeterem às operações de recombinação e mutação.

O Método de Descida em Vizinhança Variável (*Variable Neighborhood Descent* – VND) é um método de refinamento que consiste em explorar o espaço de soluções através de trocas sistemáticas de estruturas de vizinhança, aceitando somente soluções de melhora da solução corrente e retornando à primeira estrutura quando uma solução melhor é encontrada.

Segundo Souza [31], o método VND baseia-se em três princípios básicos:

- Um ótimo local com relação a uma dada estrutura de vizinhança não corres-

ponde necessariamente a um ótimo local com relação a uma outra estrutura de vizinhança;

- Um ótimo global corresponde a um ótimo local para todas as estruturas de vizinhança;
- Para muitos problemas, ótimos locais com relação a uma ou mais estruturas de vizinhança são relativamente próximos.

2.3.3.6 Aplicações de Meta-heurísticas ao PPP

Wan e Yen [34] apresentam uma metodologia baseada na meta-heurística Busca Tabu para resolver o PSUMAA com janelas de entrega distintas. Os autores dividem o problema em dois subproblemas: determinar a seqüência dos *jobs* e determinar a data de conclusão ótima do processamento de cada *job* dada uma certa seqüência. Para resolver o primeiro subproblema, os autores desenvolvem um método baseado na meta-heurística Busca Tabu. Já para resolver o segundo, os autores propõem um algoritmo para determinar a data de conclusão ótima, explorando as características do problema.

A metodologia proposta por Wan e Yen [34] pode ser resumida da seguinte forma:

1. Gere uma seqüência inicial e vá para o passo 3;
2. Gere a seqüência dos *jobs* através da BT;
3. Gere o seqüenciamento pelo algoritmo de determinação da data de conclusão ótima. Se o critério de parada for satisfeito, então pare; caso contrário, vá para o passo 2.

A estrutura de vizinhança adotada por Wan e Yen [34] para o algoritmo BT considera o movimento de troca adjacente de *jobs*. Este tipo de estrutura agiliza o método BT, pois apresenta uma quantidade pequena de vizinhos (nesta estrutura, cada solução tem somente $n-1$ vizinhos, como pode ser visto na Fig. 2.7).

Para superar o ponto fraco desta estrutura de vizinhança, que é encontrar facilmente um ótimo local devido ao pequeno número de vizinhos, Wan e Yen [34] adotaram a estratégia *multi-start*, na qual o procedimento de busca inicia de várias soluções iniciais diferentes, explorarando assim diferentes áreas do espaço de solução.

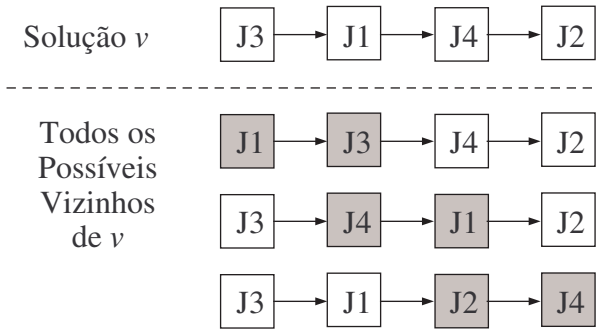

Figura 2.7: Estrutura de Vizinhança com Movimentos de troca adjacente

Wan e Yen [34] utilizaram problemas-teste com número de *jobs* igual a 15, 20, 30, 50 e 80. Os autores geraram 100 problemas-teste para cada número de *jobs*, totalizando-se 500 problemas-teste. O algoritmo proposto pelos autores apresentou um desempenho computacional excelente em termos de tempos computacionais. O algoritmo gastava em torno de 4,29 segundos para apresentar a solução final nos problemas com 80 *jobs*. Já em termos de qualidade da solução este desempenho não foi tão bom, para problemas com 80 *jobs* somente em 2% destes os autores encontraram a solução ótima (determinada por um algoritmo de *branch-and-bound*), para problemas pequenos (15 e 20 *jobs*) este desempenho foi melhor, pois em aproximadamente 20% dos problemas-teste encontrou-se a solução ótima.

Hino *et al.* [19] apresentam métodos baseados nas meta-heurísticas BT e AG para resolver o PSUMAA com datas comuns de entrega. Os autores não consideram tempos ociosos de máquina. Os autores propõem, ainda, estratégias híbridas incorporando as meta-heurísticas BT e AG. A estrutura de vizinhança adotada pelos autores para o método BT, consiste na troca aleatória entre um *job* J_1 completado antes ou na data de entrega desejada por um *job* J_2 finalizado depois da data de entrega desejada. Os autores utilizam problemas-testes com número de *jobs* igual a 10, 20, 50, 100, 200, 500 e 1000. Os autores obtiveram bons resultados com a implantação destes métodos, tiveram um tempo computacional médio de 7,8 segundos e o método AG mostrou-se ser o mais eficiente, alcançando ou ultrapassando em 92% os valores das soluções dos problemas testados. Os valores das soluções desses problemas-teste eram comparados com valores encontrados na literatura para os mesmos. O mesmo problema foi resolvido por Feldmann e Biskup [10], mas aplicando algoritmos baseados nas meta-heurísticas BT e SA. Estes autores conseguiram com a metodologia proposta uma melhora de 5% em relação aos melhores resultados conhecidos na literatura.

Lee e Choi [22] aplicaram a meta-heurística AG ao PSUMAA com datas de entrega distintas. Assim como Wan e Yen [34], Lee e Choi [22] dividiram o problema em dois sub-problemas: encontrar a seqüência de *jobs* por meio da meta-heurística AG e determinar a data de início ótima do processamento de cada *job* através de um algoritmo específico. Lee e Choi [22] resolvem problemas-teste com 15, 30, 50 e 80 jobs, gerados pelos mesmos. Os autores obtiveram resultados satisfatórios, tendo o algoritmo se mostrado eficiente para o problema abordado, superando os resultados obtidos com métodos encontrados na literatura.

França *et al.* [12] apresentaram uma metodologia baseada na meta-heurística Algoritmos Meméticos aplicada ao problema de seqüenciamento em uma máquina com minimização do tempo total de atraso e tempo de preparação de máquina dependente da seqüência de produção. Os autores aplicaram esta metodologia a problemas-teste com 20, 30, 40, 60, 80 e 100 *jobs*. Os resultados são comparados com aqueles obtidos utilizando-se métodos baseados em AG. Os métodos baseados em algoritmos meméticos se mostraram superiores aos obtidos pelos métodos baseados em AG.

Gupta e Smith [18] resolvem o mesmo problema resolvido por França *et al.* [12]. Para resolver o problema os autores propõem dois métodos. O primeiro método é uma busca local baseada no espaço de soluções e o segundo método é a meta-heurística GRASP. O método GRASP, proposto pelos autores, é dividido em três fases: Construção, Melhoramento e Reconexão por Caminhos.

Na fase de construção, as soluções são construídas por um procedimento "guloso" aleatório. Este procedimento insere os *jobs*, um a um, na seqüência de produção levando em consideração uma função custo "gulosa". Os custos para todos os jobs são calculados usando esta função, e uma lista restrita de candidatos (LRC) é formada. A LRC é formada com os *jobs* que apresentam os melhores custos. Seja CT_{max} e CT_{min} os custos máximo e mínimo avaliados pela função custo. Os *jobs* com custo menor ou igual a $(CT_{max} - CT_{min})\gamma + CT_{min}$ são inseridos na LRC e escolhidos aleatoriamente a serem inseridos na seqüência. O parâmetro γ controla o tamanho da LRC, sendo que $\gamma \in [0, 1]$. Se $\gamma = 0$ o procedimento torna-se totalmente guloso e se $\gamma = 1$ o procedimento é totalmente aleatório. O procedimento termina quando todos os *jobs* forem seqüenciados. Gupta e Smith [18] adotam γ com valores iguais a 0; 0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9 e 1,0 de acordo com uma distribuição de probabilidade uniforme.

A fase de melhora adota um procedimento de busca local baseada nos princípios do VND. São adotadas três estruturas de vizinhança: (i) movimentos de troca entre

jobs – a posição de todos os pares jobs são trocadas, e a solução corrente é alterada se uma melhora ocorre; o ótimo local é encontrado quando nenhuma troca entre pares de *jobs* resultar em uma melhor solução; (*ii*) realocação de *jobs* para frente – um *job* é realocado para posições posteriores à sua na seqüência, o método continua até nenhuma melhora ser encontrada. (*iii*) realocação de *jobs* para trás – um *job* é realocado para posições anteriores à sua na seqüência, o método continua até nenhuma melhora ser encontrada.

A fase de melhora proposta por Gupta e Smith [18] pode ser descrita como se segue.

Passo 1 : A solução corrente é obtida como descrito na fase de construção.

Passo 2 : Faça VAL igual ao valor da solução corrente. Realize a busca usando movimentos de troca entre *jobs* até um ótimo local ser encontrado. Faça o ótimo local como a solução corrente.

Passo 3 : Realize a busca usando o método de realocação para frente até um ótimo local ser obtido. Faça o ótimo local como a solução corrente.

Passo 4 : Realize a busca usando o método de realocação para trás até um ótimo local ser obtido. Faça o ótimo local como a solução corrente.

Passo 5 : Se o valor da solução corrente é melhor que VAL, então vá para o Passo 2. Senão, pare.

A terceira fase, denominada Reconexão por caminhos (PR – *Path Relinking*) é um método de intensificação de algoritmos GRASP, usualmente resultando em melhoras nas soluções obtidas.

A idéia da Reconexão por Caminhos é gerar e explorar caminhos no espaço de soluções partindo de uma ou mais soluções e levando a outras soluções. Nesse método, um conjunto de boas soluções, chamado de conjunto elite, é armazenado durante a aplicação do método de busca local. Uma variante deste método consiste em considerar todos os pares de soluções elite e para cada par aplicar a técnica de reconexão. Em cada par considera-se uma solução como guia e a outra como solução de partida. O método inicia, então, de uma solução elite e movimentos são feitos até que a solução elite guia seja alcançada. Estes movimentos podem ser quaisquer movimentos de busca na vizinhança, tais como os descritos na fase de melhora. Estes movimentos devem

preservar a viabilidade da solução e devem sistematicamente introduzir os atributos da solução guia. O processo é repetido até a solução guia ser obtida da solução inicial. Deste modo, a Reconexão por Caminhos busca introduzir os atributos da solução elite guia dentro das soluções bases a fim de encontrar soluções ainda melhores.

No método proposto por Gupta e Smith [18], o tamanho do conjunto elite é igual a 100. Este conjunto inicia-se vazio e as primeiras 100 soluções encontradas nas primeiras 100 iterações são armazenadas inicialmente neste conjunto. No final de cada iteração, se um ótimo local encontrado é melhor que a pior solução no conjunto, ela substitui a pior solução no conjunto. O método PR é aplicado a cada 100 iterações e como método de pós-otimização.

Gupta e Smith [18] resolveram problemas-teste com 15, 25, 35, 45, 55, 65, 75 e 85 *jobs*, gerados pelos próprios. Os resultados obtidos para o GRASP com Reconexão por caminhos foram muito bons, na maioria das vezes os resultados encontrados, inclusive os piores resultados, eram melhores, iguais ou bem próximos da melhor solução encontrada para os problemas-teste. Nos problemas menores (15 a 45 *jobs*), os resultados eram comparados com resultados obtidos por um método Branch-and-bound e por meio de alguns métodos meta-heurísticos. Já para os problemas maiores (55 a 85 *jobs*), utilizou-se apenas um dos métodos meta-heurísticos para a comparação. Estes métodos meta-heurísticos foram reproduzidos como encontrados na literatura.

Capítulo 3

Metodologia

Este trabalho tem seu foco no PSUMAA com tempo de preparação da máquina dependente da seqüência de produção e janelas de entrega, o qual é definido na Seção 3.1. Um modelo de programação linear inteira mista do problema é apresentado na Seção 3.2. Na Seção 3.3 são apresentados os métodos heurísticos para resolução do PSUMAA. São propostos métodos baseados nas meta-heurísticas GRASP e ILS para resolvê-lo. Esses métodos, por sua vez, fazem uso de procedimentos de refinamento baseados no método randômico de descida usando duas diferentes estruturas de vizinhança: realocação de *jobs* e troca de *jobs*.

Nessa Seção são descritos, ainda, os métodos para determinação de uma solução inicial, os métodos de busca local adotados, a função de avaliação, as estruturas de vizinhança e a forma de representar uma solução do problema. Baseado nos trabalhos de Wan e Yen [34] e Lee e Choi [22], é descrito, também, um algoritmo para determinar as datas ótimas de início do processamento dos *jobs*.

3.1 O Problema Estudado

O problema estudado nesta dissertação é o PSUMAA com tempo de preparação da máquina dependente da seqüência de produção e janelas de entrega e possui as seguintes características:

(*a*) Uma máquina deve processar um conjunto de n *jobs*.

(*b*) Cada *job* possui um tempo de processamento P_i, uma data inicial E_i e uma data final T_i desejadas para o término do processamento.

(*c*) A máquina executa no máximo um *job* por vez e uma vez iniciado o processamento de um *job*, o mesmo deve ser finalizado, ou seja, não é permitido a interrupção do processamento.

(*d*) Todos os *jobs* estão disponíveis para processamento na data 0.

(*e*) Quando um *job* j é seqüenciado imediatamente após um *job* i, sendo estes pertencentes a diferentes famílias de produtos, é necessário um tempo S_{ij} para a preparação da máquina. Tempos de preparação de máquina nulos ($S_{ij} = 0$) implicam em produtos da mesma família. Assume-se, ainda, que a máquina não necessita de tempo de preparação inicial, ou seja, o tempo de preparação da máquina para o processamento do primeiro *job* na seqüência é igual a 0.

(*f*) É permitido tempo ocioso entre a execução de dois *jobs* consecutivos.

(*g*) Os *jobs* devem ser finalizados dentro da janela de tempo $[E_i, T_i]$, denominada janela de entrega. Se o *job* i for finalizado antes de E_i então há um custo de manutenção de estoque. Caso o *job* seja finalizado após T_i então há associado um custo por atraso. Os *jobs* que forem finalizados dentro da janela de entrega não proporcionarão nenhum custo adicional para a empresa.

(*h*) Os custos unitários por antecipação e atraso da produção são dependentes dos *jobs*, ou seja, cada *job* possui um custo de antecipação α_i e um custo de atraso β_i.

(*i*) O objetivo a ser alcançado com a resolução deste problema é a minimização do somatório dos custos de antecipação e atraso da produção.

Nas seções seguintes são apresentados o modelo matemático para representar o problema e os métodos de resolução do mesmo.

3.2 Modelagem Matemática

O modelo matemático desenvolvido foi baseado no trabalho de Bustamante [6], apresentado na Seção 2.2.1, e Manne [25].

Sejam n o número de *jobs* a serem processados, P_i o tempo de processamento do *job* i, s_i a data de início do processamento do *job* i ($s_i \geq 0$) e S_{ij} o tempo de preparação da máquina necessário para processar o *job* j depois do *job* i.

Diferentemente de Bustamante [6], foram utilizados dois *jobs* fictícios, 0 (zero) e $n+1$, de tal forma que 0 antecede imediatamente a primeira tarefa e $n+1$ sucede imediatamente o último *job* na seqüência de produção. Admite-se que P_0 e P_{n+1} são iguais a zero e que $S_{0i} = 0$ e $S_{i,n+1} = 0, \quad \forall\, i \;=\; 1, \ldots, n$.

Para garantir que haja um tempo suficiente para completar um *job* i antes de começar um *job* j, caso um *job* j seja processado imediatamente após um *job* i, sem nenhum *job* intermediário, é necessário impor as restrições (3.1) e (3.2). As restrições (3.1) são válidas quando o *job* j sucede imediatamente o *job* i na seqüência de produção. Quando o *job* j antecede o *job* i, as restrições (3.2) tornam-se válidas.

$$s_j \geq s_i + P_i + S_{ij} \quad \forall\, i = 0, 1, \ldots, n\ ,\ \forall\, j = 1, 2, \ldots, n+1 \ \ e \ \ i \neq j \tag{3.1}$$

ou

$$s_i \geq s_j + P_j + S_{ji} \quad \forall\, i = 1, 2, \ldots, n+1\ ,\ \forall\, j = 0, 1, \ldots, n \ \ e \ \ i \neq j \tag{3.2}$$

Como estas restrições são disjuntas, faz-se necessário a introdução de uma variável $y_{ij} \in [0, 1]$ para que esta disjunção não apareça no modelo matemático. A variável y_{ij} é definida da seguinte forma:

$$y_{ij} = \begin{cases} 1, & \text{se o } job\ j \text{ é sequenciado imediatamente após o } job\ i; \\ 0, & \text{caso contrário.} \end{cases}$$

Desta maneira, as restrições (3.1) e (3.2) podem ser substituídas pelas restrições (3.3). Neste novo conjunto de restrições, M é um valor muito grande.

$$s_j - s_i - y_{ij}(M + S_{ij}) \geq P_i - M \quad \forall\, i = 0, 1, \ldots, n\ ,\ \forall\, j = 1, 2, \ldots, n+1 \ \ e \ \ i \neq j \tag{3.3}$$

Desta forma, quando $y_{ij} = 1$, a restrição (3.3) torna-se:

$$s_j \geq s_i + P_i + S_{ij} \tag{3.4}$$

E quando $y_{ij} = 0$, tem-se:

$$s_j - s_i - P_i \geq -M \tag{3.5}$$

A restrição (3.3) é, desta forma, desativada, pois a equação (3.5) é redundante, ou seja, a parcela $(s_j - s_i - P_i)$ será sempre maior que $-M$.

É proposição deste trabalho, a introdução das restrições (3.6) e (3.7) inspiradas

em modelos de fluxo em redes (AHUJA et al. [1]). Estas restrições garantem que cada *job* tenha somente um *job* imediatamente antecessor e um *job* imediatamente sucessor, respectivamente. Além disso, estas restrições eliminam o problema da desigualdade triangular.

$$\sum_{i=0,\, i\neq j}^{n} y_{ij} = 1 \quad \forall\, j = 1, 2, \ldots, n+1 \tag{3.6}$$

$$\sum_{j=1,\, j\neq i}^{n+1} y_{ij} = 1 \quad \forall\, i = 0, 1, \ldots, n \tag{3.7}$$

Sejam E_i a data de início do período de entrega do *job* i, e_i o tempo de antecipação do *job* i, T_i a data de término do período de entrega do *job* i e t_i o tempo de atraso do *job* i. As restrições (3.8) a (3.11) garantem que o tempo de antecipação e_i seja o máximo entre 0 e $E_i - P_i - s_i$ e que o tempo de atraso t_i seja o máximo entre 0 e $s_i + P_i - T_i$.

$$s_i + P_i + e_i \geq E_i \quad \forall\, i = 1, 2, \ldots, n \tag{3.8}$$

$$s_i + P_i - t_i \leq T_i \quad \forall\, i = 1, 2, \ldots, n \tag{3.9}$$

$$e_i \geq 0 \quad \forall\, i = 1, 2, \ldots, n \tag{3.10}$$

$$t_i \geq 0 \quad \forall\, i = 1, 2, \ldots, n \tag{3.11}$$

Sejam α_i e β_i os custos de antecipação e atraso da produção do *job* i por unidade de tempo, respectivamente. O custo total por antecipação é dado por $\sum_{i=1}^{n} \alpha_i e_i$ e o custo total por atraso é dado por $\sum_{i=1}^{n} \beta_i t_i$. A função objetivo, que consiste em minimizar o somatório dos custos totais de antecipação e atraso da produção, é dada pela equação (3.12).

$$\min \quad Z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{3.12}$$

Resumindo, as variáveis de decisão do modelo proposto são:

- s_i: data de início do processamento do *job* i;
- y_{ij}: variável que determina a seqüência de produção, se y_{ij}=1 o *job* j é processado imediatamente depois do *job* i e 0 caso contrário;
- e_i: tempo de antecipação do *job* i;
- t_i: tempo de atraso do *job* i;

E o modelo correspondente de Programação Linear Inteira Mista (PLIM) é:

$$\text{minimizar} \quad Z = \sum_{i=1}^{n} (\alpha_i e_i + \beta_i t_i) \tag{3.13}$$

$$\text{sujeito a:} \quad s_j - s_i - y_{ij}(M + S_{ij}) \geq P_i - M \quad \forall\, i = 0, 1, \ldots, n; \quad \forall\, j = 1, 2, \ldots, n+1 \; e \; i \neq j \tag{3.14}$$

$$\sum_{j=1,\, j\neq i}^{n+1} y_{ij} = 1 \qquad \forall\, i = 0, 1, \ldots, n \tag{3.15}$$

$$\sum_{i=0,\, i\neq j}^{n} y_{ij} = 1 \qquad \forall\, j = 1, 2, \ldots, n+1 \tag{3.16}$$

$$s_i + P_i + e_i \geq E_i \qquad \forall\, i = 1, 2, \ldots, n \tag{3.17}$$

$$s_i + P_i - t_i \leq T_i \qquad \forall\, i = 1, 2, \ldots, n \tag{3.18}$$

$$s_i \geq 0 \qquad \forall\, i = 0, 1, \ldots, n+1 \tag{3.19}$$

$$e_i \geq 0 \qquad \forall\, i = 1, 2, \ldots, n \tag{3.20}$$

$$t_i \geq 0 \qquad \forall\, i = 1, 2, \ldots, n \tag{3.21}$$

$$y_{ij} \in \{0, 1\} \qquad \forall\, i, j = 0, 1, 2, \ldots, n+1 \tag{3.22}$$

A função objetivo, representada pela equação (3.13), tem como critério de otimização a minimização dos custos de antecipação e atraso. As restrições (3.14) definem a seqüência de operações sobre o recurso (máquina) utilizado. As restrições (3.15) e (3.16) garantem que cada *job* tenha somente um *job* imediatamente antecessor e um *job* imediatamente sucessor, respectivamente. As restrições (3.17) e (3.18) definem os valores do atraso e da antecipação de acordo com a janela de entrega desejada para o término do processamento do *job* i, caso estes existam. As restrições (3.19) a (3.22) definem o domínio das variáveis do problema.

3.3 Métodos de Resolução

O PSUMAA com tempo de preparação da máquina dependente da seqüência de produção e com janelas de entrega é um problema fortemente NP-difícil (Pinedo [28], Liaw [24], Hino *et al.* [19], Feldmann e Biskup [10], Baker e Scudder [5], Wan e Yen [34] e Sourd [30]). O modelo matemático proposto na Seção 3.2 é geralmente capaz de resolver problemas de tamanho reduzido se comparados à problemas reais, ou seja, com um pequeno número de *jobs*. Para a resolução de problemas de maiores dimensões é proposto nesta seção métodos baseados nas meta-heurísticas GRASP e ILS.

A resolução do problema é decomposta em dois sub-problemas: (i) determinação de uma seqüência de *jobs* e (ii) determinação das datas ótimas de início do processamento dos *jobs*. O primeiro sub-problema é resolvido através dos métodos heurísticos propostos. Já o segundo sub-problema parte de uma seqüência de *jobs* e aplica um algoritmo específico para determinar qual a data ótima de início do processamento de cada *job* na seqüência de produção.

O procedimento proposto pode ser resumido nas seguintes etapas e ilustrado pela Fig. 3.1:

(1) **Inicialização:** Gere uma solução inicial e vá para o passo (3);

(2) Gere uma seqüência de *jobs* por algum dos métodos propostos;

(3) Gere a programação da produção pelo algoritmo de determinação das datas ótimas. Se o critério de parada for satisfeito, então pare; caso contrário, vá para o passo (2).

A Seção 3.3.1 apresenta o algoritmo de determinação das datas ótimas de início de processamento. Na Seção 3.3.2 são apresentados os métodos heurísticos destinados à resolução do problema.

3.3.1 Determinação das Datas Ótimas de Início de Processamento

Dada uma seqüência de *jobs*, o algoritmo de determinação das datas ótimas de início de processamento (ADDOIP) determina a melhor data para se iniciar o processamento de um *job*, de acordo com a janela de entrega do mesmo.

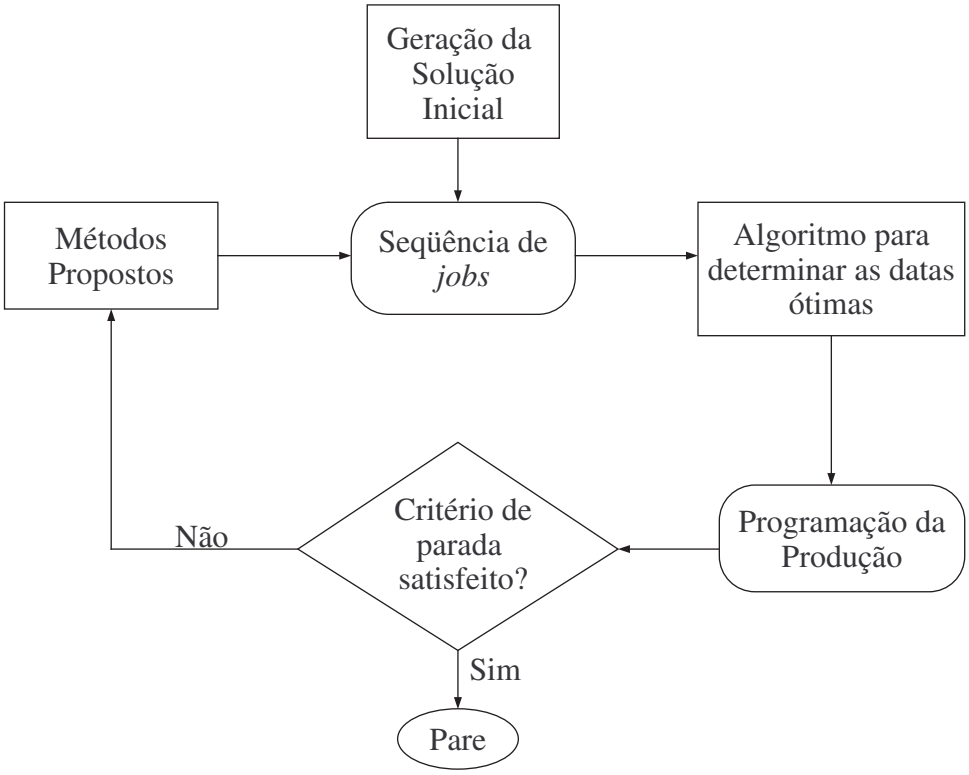

Figura 3.1: Procedimento Proposto. Fonte: Adaptado de Wan e Yen [34]

O ADDOIP aqui proposto é baseado nos trabalhos de Wan e Yen [34], Lee e Choi [22] e Chrétienne e Sourd [9].

Sejam V a seqüência de *jobs* dada e C_k a data de conclusão do processamento do *job* k. Suponha que um sub-conjunto de *jobs* $B_j = \{J_0, J_1, \ldots, J_m\} \subseteq V$ forme um bloco no seqüenciamento, ou seja, os *jobs* em B_j são seqüenciados consecutivamente sem tempo ocioso entre eles. Além disso, assume-se que há l blocos em V e sejam $prim(j)$ e $ult(j)$ o primeiro e o último *job* no bloco B_j, respectivamente.

O processamento do primeiro *job* (J_1) da seqüência V é programado para ser finalizado na data E_{J_1} se $P_{J_1} \leq E_{J_1}$ ou iniciado na data 0 (finalizando na data P_{J_1}) se $P_{J_1} > E_{J_1}$.

Os demais *jobs* são programados da seguinte forma:

- Se $C_{J_k} + S_{(J_k)(J_{k+1})} + P_{J_{k+1}} < E_{J_{k+1}}$, então o *job* J_{k+1} tem seu processamento programado para ser finalizado na data $E_{J_{k+1}}$, isto é, o *job* J_{k+1} tem seu processamento programado para ser finalizado no início de sua janela de entrega, desta forma, iniciando um novo bloco.

- Por outro lado, se $C_{J_k} + S_{(J_k)(J_{k+1})} + P_{J_{k+1}} \geq E_{J_{k+1}}$, então o *job* J_{k+1} tem seu processamento programado para ser finalizado na data $C_{J_k} + S_{(J_k)(J_{k+1})} + P_{J_{k+1}}$ e é incluído como último elemento do bloco corrente.

Após a determinação da data de conclusão do processamento de cada *job*, necessita-se verificar o posicionamento dos blocos. A função custo para o bloco B_j da mudança de x unidades de tempo da data $E_{prim(j)}$ para a esquerda pode ser expressa pela equação (3.23).

$$Custo_j(x) = \sum_{i \in B_j} g_i(C_i - x) \tag{3.23}$$

onde $C_i - x$ é a nova data de conclusão do processamento do *job* i.

Lema 1(Wan e Yen [34]): *O somatório de duas funções lineares convexas por partes é também uma função linear convexa por partes.*

Baseado no Lema 1, Wan e Yen [34] fazem a seguinte proposição.

Proposição 1(Wan e Yen [34]): *$Custo_j(x)$ é uma função linear convexa por partes em relação a x.*

O custo mínimo do bloco B_j ocorre nos pontos extremos de sua função custo, isto é, no início ou no final da janela de entrega de um dos *jobs* no bloco. Por causa da natureza linear convexa por partes da função custo, o custo mínimo pode ser facilmente obtido pela comparação dos somatórios das penalidades dos *jobs* no bloco, no início e no final de cada janela de entrega no bloco (penalidades por antecipação são consideradas negativas enquanto penalidades por atraso são consideradas positivas). Estes somatórios fornecem as inclinações das retas (m) pertencentes à função custo. O custo mínimo do bloco B_j se dá no ponto extremo onde a inclinação m torna-se maior ou igual a zero.

Quando o ponto mínimo do bloco B_j é encontrado, todo o bloco é movido em direção ao ponto mínimo até que um dos três casos seguintes aconteça:

(i) $s_{prim(j)} = 0$.

(ii) O ponto mínimo é alcançado.

(iii) $s_{prim(j)}$ torna-se igual a $C_{ult(j-1)} + S_{(ult(j-1))(prim(j))}$.

No caso (iii), o bloco B_j é unido ao bloco B_{j-1}, resultando em um novo bloco B_{j-1}. Então, o ponto mínimo do novo B_{j-1} deve ser obtido. O procedimento anterior deve ser realizado até que o caso (i) e (ii) ocorram para cada bloco.

Proposição 2(Wan e Yen [34]): *O custo total de uma dada seqüência de jobs alcança seu valor ótimo se cada bloco B_j na seqüência alcançar seu ponto mínimo, com exceção de $s_{prim(1)}$ que pode ser igual a zero para o primeiro bloco.*

O ADDOIP para uma seqüência de *jobs* dadas é descrito na Fig. 3.2, onde o procedimento *MudaBloco* movimenta o bloco J até seu ponto de mínimo e, caso seja necessário, combina-o com o bloco anterior.

```
procedimento Determina_Datas_Otimas_Inicio(n, v, f(v))
1  {Inicialização}
2  B ← 1; prim(B) ← ult(B) ← 1;
3  s_1 ← max{0, E_1 − P_1};
4  C_1 ← max{E_1, P_1};
5  {Demais jobs}
6  para i = 2 até n (passo= 1) faça
7      se (C_{i−1} + P_i + S_{(i−1)(i)} < E_i)
8          então
9              B ← B + 1;
10             prim(B) ← ult(B) ← i;
11             s_i ← E_i − P_i + S_{(i−1)(i)};
12             C_i ← E_i;
13         senão
14             se (C_{i−1} + P_i + S_{(i−1)(i)} = E_i)
15                 então
16                     ult(B) ← i;
17                     s_i ← C_{i−1} + S_{(i−1)(i)};
18                     C_i ← E_i;
19                 senão
20                     ult(B) ← i;
21                     s_i ← C_{i−1} + S_{(i−1)(i)};
22                     C_i ← s_i + P_i;
23             fim-se;
24     fim-se;
25     repita (até todos os blocos estiverem no ponto mínimo) ou (s_{prim(1)} = 0)
26         MudaBloco(B);
27     fim-repita;
28 fim-para
29 Determina f(v);
fim Determina_Datas_Otimas_Inicio;
```

Figura 3.2: Algoritmo para Determinar as Datas Ótimas de Início de Processamento

Para exemplificar o método, seja a Tab. 3.1, relativa a 4 jobs, em que se

apresenta o tempo de processamento (P_i), o custo por antecipação (α_i), o custo por atraso (β_i), os tempos de preparação da máquina (S_{ij}), a data inicial (E_i) e a data final (T_i) desejadas para entrega. Seja também a seqüência $V = \{3, 4, 1, 2\}$.

Tabela 3.1: Dados do Exemplo

Jobs	**Dados do Problema**					**Tempo de Preparação da Máquina**			
	P_i	E_i	T_i	α_i	β_i	**1**	**2**	**3**	**4**
1	3	3	7	9	18	0	1	1	2
2	5	11	16	10	20	1	0	3	2
3	4	5	9	7	14	1	3	0	2
4	5	7	13	8	16	2	2	2	0

A seguir, são descritas a inicialização e as primeiras iterações para o ADDOIP aplicado ao exemplo dado.

Inicialização: Determinando as datas de início e término do processamento do primeiro job na seqüência: $J_1 = 3$; $s_{J_1} = s_3 = \max\{0, E_3 - P_3\} = \max\{0, 5 - 4\} = 1$; $C_{J_1} = C_3 = \max\{E_3, P_3\} = E_3 = 5$. A inserção do primeiro *job* na seqüência pode ser vista no gráfico da Fig. 3.3.

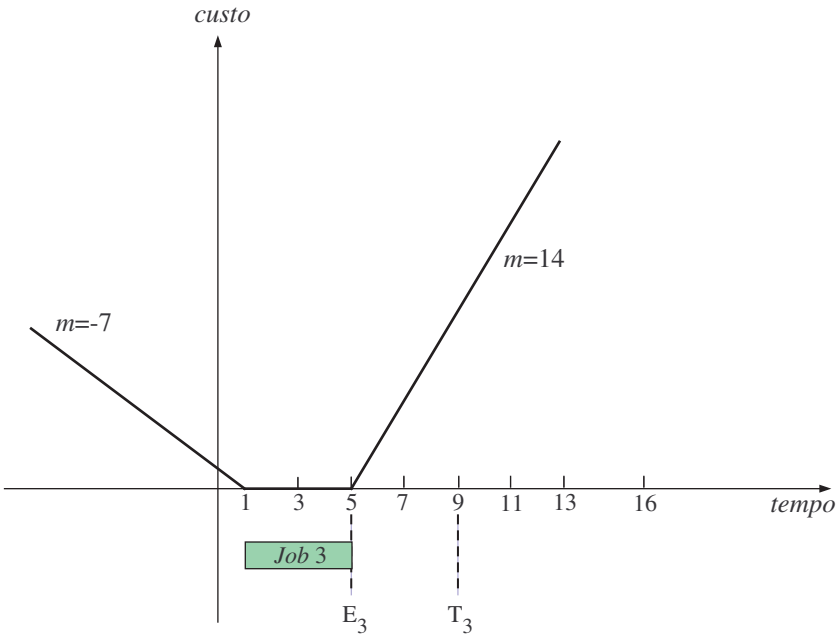

Figura 3.3: Inserção do primeiro *job* na seqüência do exemplo dado

Primeira Iteração: Determinando as datas de início e término do processamento do segundo job na seqüência: $J_2 = 4$; $C_3 + S_{34} + P_4 = 5 + 2 + 5 = 12$. Como

$C_3 + S_{34} + P_4 > E_4$ $(12 > 7)$, o job 4 é inserido como último elemento do bloco corrente. E $s_{J_2} = s_4 = C_3 + S_{34} = 5 + 2 = 7$; $C_{J_2} = C_4 = s_4 + P_4 = 7 + 5 = 12$.

É necessário verificar neste momento a posição do bloco para verificar se ele está em seu ponto mínimo. Para isto, primeiramente, determina-se as possíveis datas de início do processamento do bloco. Essas datas são obtidas posicionando cada *job* do bloco no início e no final de sua janela de entrega, pois como foi visto o custo mínimo do bloco se dá no início ou no final da janela de entrega de um dos *jobs* no bloco. No exemplo mostrado, as possíveis datas de início do processamento do bloco são: -4, 1, 2 e 5. Determinadas as possíveis datas de início de processamento, calculam-se então as inclinações m de acordo com essas datas. Para calcular essas inclinações, as penalidades por antecipação assumem valores negativos e as penalidades por atraso assumem valores positivos. O ponto mínimo, o qual deseja-se encontrar, se dá no ponto aonde a inclinação m se torna maior ou igual a zero. No exemplo apresentado, se o processamento do bloco iniciar antes da data 4, ou seja, com todos os jobs sendo antecipados, então a inclinação m é dada pelo somatório das penalidades por antecipação dos *jobs* 3 e 4, portanto $m = -7 - 8 = -15$. Agora, se o processamento do bloco iniciar entre a data -4 e 1, o *job* 4 deixa de estar antecipado, e a inclinação m passa a ser $m = -7$ (penalidade por antecipação do *job* 3). Agora, se o processamento do bloco iniciar entre a data 1 e 2, o *job* 3 deixa de estar antecipado, e a inclinação m passa a ser $m = 0$. Como m tornou-se maior igual a zero este ponto é o ponto mínimo do bloco, ou seja, o custo mínimo do bloco se dá quando o processamento do bloco se inicia na data 1. Como o início do processamento bloco já é na data 1 então não é necessário movimentar o bloco até esta data. Esta situação é ilustrada na Fig. 3.4. Nesta figura são apresentadas, ainda, as demais inclinações, mas no momento em que a inclinação torna-se maior ou igual a zero, o procedimento é interrompido, pois este é o ponto mínimo do bloco.

Segunda Iteração: Determinando as datas de início e término do processamento do terceiro job na seqüência: $J_3 = 1$; $C_4 + S_{41} + P_1 = 12 + 2 + 3 = 17$. Como $C_4 + S_{41} + P_1 > E_1$ $(17 > 3)$, o job 1 é inserido como último elemento do bloco corrente. E $s_{J_3} = s_1 = C_4 + S_{41} = 12 + 2 = 14$; $C_{J_3} = C_1 = s_1 + P_1 = 14 + 3 = 17$.

Verificando a posição do bloco: As possíveis datas de início do processamento do bloco são: -13, -11, -4, 1, 2 e 5. As inclinações m calculadas de acordo com estas datas são mostradas na Fig. 3.5.

Na Fig. 3.5 verifica-se que na data -11 a inclinação m torna-se maior ou igual

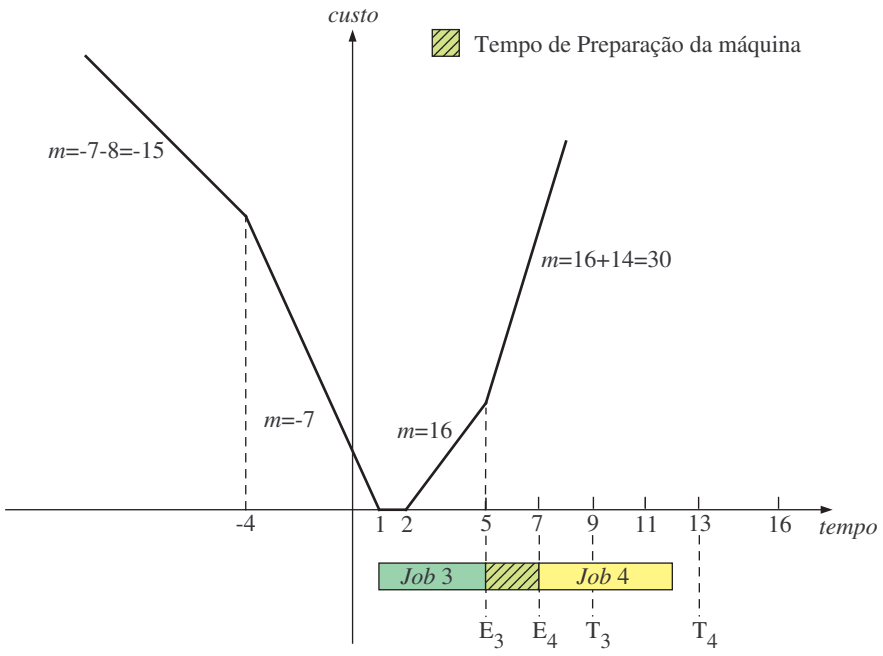

Figura 3.4: Inserção do segundo *job* na seqüência do exemplo dado

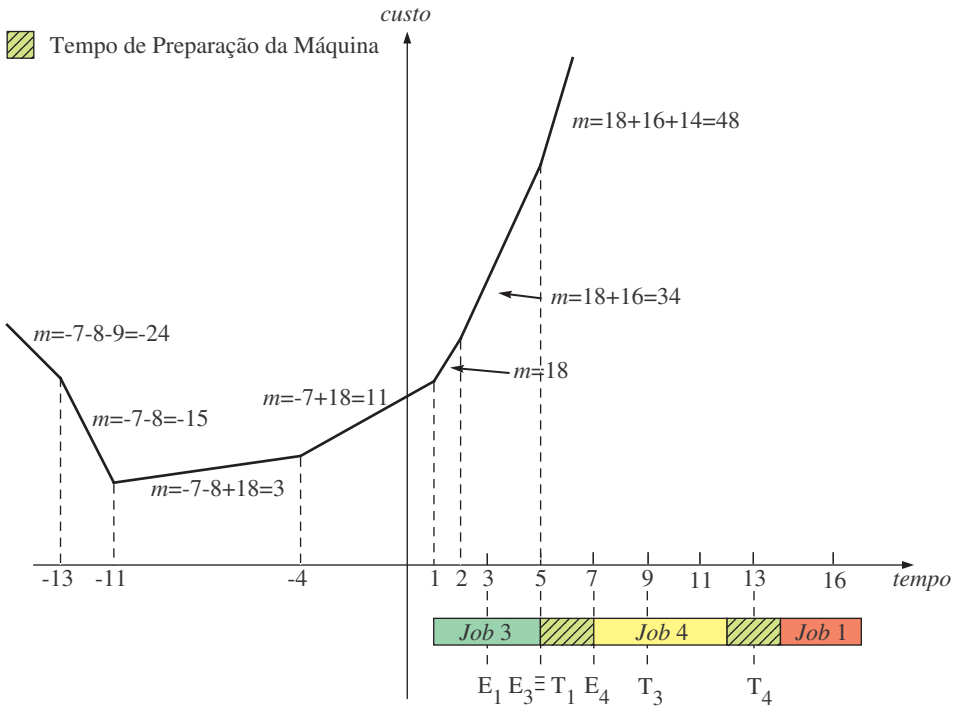

Figura 3.5: Inserção do terceiro *job* na seqüência do Exemplo dado

a zero, sendo este, então, o ponto mínimo do bloco. Como pode ser visto também nesta figura o bloco se encontra no instante 1, então o bloco deve ser movido até um

dos três casos citados anteriormente acontecerem. No exemplo considerado, o bloco é empurrado até o caso (i) ocorrer, ou seja, $s_{prim(1)} = s_3 = 0$. Daí todas as datas são atualizadas. Desta forma, tem-se: $s_3 = 0$ e $C_3 = 4$; $s_4 = 6$ e $C_4 = 11$; $s_1 = 13$ e $C_1 = 16$. A Fig. 3.6 apresenta o bloco já movido até o instante 0.

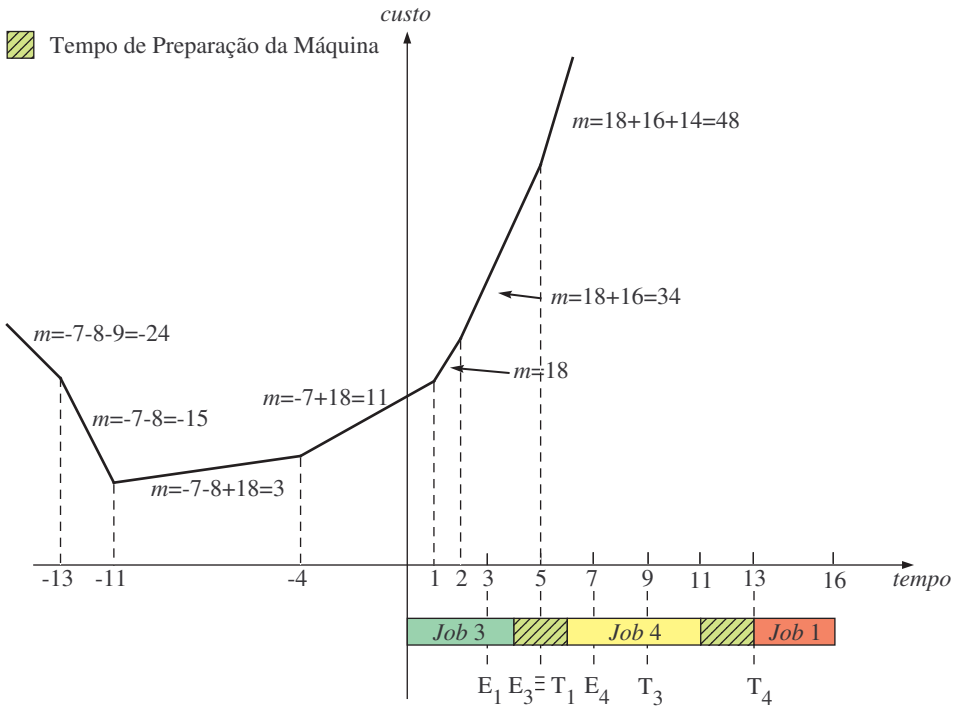

Figura 3.6: Bloco movido até o ponto de mínimo

Este procedimento se repete até que todos os *jobs* estejam seqüenciados.

3.3.2 Determinação da Seqüência de *Jobs*

Com a finalidade de determinar a melhor seqüência de processamento dos *jobs* são propostos métodos baseados nas meta-heurísticas ILS e GRASP. Para ambos métodos foram utilizados as mesmas estruturas de vizinhança, descrita na Seção 3.3.2.2, os mesmos métodos de busca local, descritos na Seção 3.3.2.5, e o mesmo método de geração da solução inicial, descrito na Seção 3.3.2.4. São apresentadas, ainda, a função de avaliação do problema e a forma de representar uma solução do mesmo.

3.3.2.1 Representação de uma Solução

Uma solução para o problema é representada por um vetor v de n posições, com cada posição v_i indicando a ordem de produção do i-ésimo *job*.

Por exemplo, para seis *jobs*, uma possível solução é a seqüência $v = \{3, 2, 6, 4, 1, 5\}$. Nesta solução, o *job* 3 é o primeiro a ser realizado e o *job* 5, o último.

3.3.2.2 Vizinhança de uma Solução

Para explorar o espaço de soluções são utilizados dois tipos de movimentos: troca da ordem de processamento de dois *jobs* da seqüência de produção e realocação de um *job* para outra posição na seqüência de produção. Esses movimentos definem, respectivamente, as estruturas de vizinhança $N^{(T)}$ e $N^{(R)}$.

Na primeira estrutura de vizinhança, $N^{(T)}$, para um conjunto com n *jobs*, há $n(n-1)/2$ movimentos possíveis, ou equivalentemente, esse mesmo número de vizinhos. Por exemplo, na Fig. 3.7, a solução v' é um vizinho da solução v, pois é obtido pela troca do primeiro com o quarto *job*.

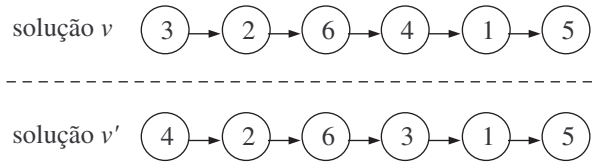

Figura 3.7: Estrutura de vizinhança com troca da ordem de processamento de dois *jobs*

Já na segunda estrutura de vizinhança, $N^{(R)}$, há $(n-1)^2$ movimentos possíveis. A Fig. 3.8 apresenta um exemplo deste tipo de estrutura. A solução v', vizinha a v, é obtida pela realocação do *job* da primeira posição para a quarta posição na seqüência de produção. Nesta estrutura são permitidos movimentos para posições sucessoras ou antecessoras à posição em que o *job* se encontra na seqüência de produção.

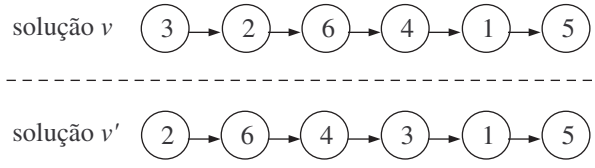

Figura 3.8: Estrutura de vizinhança com realocação de *jobs*

3.3.2.3 Função de Avaliação

Como os movimentos propostos não produzem soluções inviáveis, uma solução do problema é avaliada pela própria função objetivo, dada pela expressão (3.13) do modelo de programação matemática.

3.3.2.4 Geração da Solução Inicial

O método proposto para construir uma solução inicial baseia-se no método de construção da meta-heurística GRASP, onde os elementos são inseridos um a um na solução de acordo com uma função gulosa, a qual estima o benefício da seleção de cada elemento.

A cada iteração do método de construção GRASP, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista de candidatos, seguindo um critério de ordenação pré-determinado. Os melhores candidatos, de acordo com uma função gulosa, são colocados em uma lista restrita de candidatos (LRC). A seguir, um desses candidatos é escolhido aleatoriamente e inserido na solução corrente. Um parâmetro γ é utilizado para controlar o tamanho da LRC.

Apresenta-se, a seguir, as adaptações do método GRASP para construir uma solução para o PSUMAA.

Para estimar o benefício da inserção de cada *job*, utiliza-se a data de início de entrega desejada como função de avaliação. O procedimento proposto é, portanto, uma variante da heurística construtiva EDD descrita na seção 2.3.2.1.

Na Fig. 3.9 apresenta-se o pseudo-código da construção GRASP aplicada ao problema de seqüenciamento em questão. Nesta figura, E_{min} representa cronologicamente a primeira data de início de entrega e E_{max}, a última.

O método proposto para geração da solução inicial utiliza o método de construção do GRASP para gerar um conjunto de soluções iniciais e selecionar a melhor delas. Este método inicia com uma solução gulosa (isto é, faz-se $\gamma = 0$) e as demais iterações selecionam, aleatoriamente, um valor para γ dentro de um conjunto de valores (Γ). Este método é repetido por um certo número de iterações ($MaxIter$). A melhor solução encontrada por esse procedimento é então retornada. Este método de geração de uma solução inicial é esquematizado na Fig. 3.10.

procedimento $Constroi\_Solucao\_GRASP(n, v, \gamma)$
1 $v \leftarrow \emptyset$;
2 Inicialize o conjunto LC de candidatos;
3 enquanto $(LC \neq \emptyset)$ faça
4 $E_{min} = \min_{i \in LC}\{E_i\}$;
5 $E_{max} = \max_{i \in LC}\{E_i\}$;
6 $LRC = \{i \in LC \mid E_i \leq E_{min} + \gamma(E_{max} - E_{min})\}$;
7 Selecione, aleatoriamente, um elemento $i \in LRC$;
8 $v \leftarrow v \cup \{i\}$;
9 Atualize o conjunto LC de candidatos;
10 fim-enquanto;
11 Retorne v;
fim $Constroi\_Solucao\_GRASP$;

Figura 3.9: Algoritmo para Construir uma Solução via GRASP

procedimento $Constroi\_Solucao\_Inicial(n, v, MaxIter)$
1 $Constroi\_Solucao\_GRASP(n, v', \gamma = 0)$; {Constrói uma solução gulosa}
2 $Determina\_Datas\_Otimas\_Inicio(n, v', f(v'))$;
3 $v^* \leftarrow v'$;
4 $f^* \leftarrow f(v')$;
5 para $i = 1$ até $MaxIter$ (passo= 1) faça
6 Selecione, aleatoriamente, um valor $\gamma \in \Gamma$;
7 $Constroi\_Solucao\_GRASP(n, v', \gamma)$;
8 $Determina\_Datas\_Otimas\_Inicio(n, v', f(v'))$;
9 se $(f(v') < f^*)$ então
10 $v^* \leftarrow v'$;
11 $f^* \leftarrow f(v')$;
12 fim-se;
13 fim-para;
14 Retorne v^*;
fim $Constroi\_Solucao\_Inicial$;

Figura 3.10: Algoritmo para Gerar uma Solução Inicial

3.3.2.5 Métodos de Busca Local

Foram utilizados três métodos de busca local. Todos eles usam as duas estruturas de vizinhança descritas na Seção 3.3.2.2.

O primeiro método é baseado no método randômico de descida (MRD), descrito na Fig. 3.11. Neste método, o vizinho v' é gerado aleatoriamente de acordo com a estrutura de vizinhança adotada ($N^{(T)}(v)$ ou $N^{(R)}(v)$).

O segundo método de busca local é o método de descida (MD). Ele é utilizado para poder garantir que uma solução é um ótimo local, segundo uma determinada

```
procedimento Metodo_Randomico_Descida(n, v, IterMax)
1  iter ← 0; {Contador de Iterações sem melhora}
2  enquanto (iter < IterMax) faça
3      iter ← iter + 1;
4      Selecione, aleatoriamente, v' ∈ N(v);
5      se (f(v') < f(v)) então
6          v ← v';
7          iter ← 0;
8      fim-se;
9  fim-enquanto;
10 Retorne v;
fim Metodo_Randomico_Descida;
```

Figura 3.11: Método Randômico de Descida.

estrutura de vizinhança. Neste método, é feita uma varredura completa na vizinhança ($N(v)$) de uma solução v por melhores soluções. Este método é descrito na Fig. 3.12.

```
procedimento Metodo_Descida(n, v)
1  V = {v' ∈ N(v) | f(v') < f(v)};
2  enquanto (|V| > 0) faça
3      Selecione v' ∈ V, onde v' =argmin{f(v') | v' ∈ V};
4      v ← v';
5      V = {v' ∈ N(v) | f(v') < f(v)};
6  fim-enquanto;
7  Retorne v;
fim Metodo_Descida;
```

Figura 3.12: Método da Descida.

O terceiro método é baseado na meta-heurística VND, e trabalha com as estruturas de vizinhança $N^{(T)}$ e $N^{(R)}$. O método utiliza um dos métodos de busca local descritos anteriormente para gerar um vizinho v' da solução corrente v. Este método é descrito na Fig. 3.13.

3.3.2.6 ILS

Para aplicar um algoritmo ILS, quatro componentes têm que ser especificados: (i) um procedimento que gera uma solução inicial v_0 para o problema; (ii) um procedimento de busca local, que retorna uma solução melhorada v''; (iii) um procedimento de perturbação, que modifica a solução corrente v guiando a uma solução intermediária v' e (iv) um procedimento que decide de qual solução a próxima perturbação será aplicada.

```
procedimento Metodo_VND(n, v, IterMax)
1  Seja r o número de estruturas diferentes de vizinhanças;
2  k ← 1; {Tipo de estrutura de vizinhança corrente}
3  enquanto(k ≤ r)faça
4      Encontre o melhor vizinho v' usando o MRD ou MD
           com a estrutura de vizinhança N^(k);
5      se (f(v') < f(v)) então
6          v ← v';
7          k ← 1;
8      senão
9          k ← k + 1;
10     fim-se;
11 fim-enquanto;
12 Retorne v;
fim Metodo_VND;
```

Figura 3.13: Método VND.

A solução inicial é gerada de acordo com o método descrito na Seção 3.3.2.4, Fig. 3.10. Os métodos de busca local utilizados são os descritos na Seção 3.3.2.5.

As perturbações são feitas por meio de trocas aleatórias entre dois *jobs* quaisquer da solução. Caso o nível de perturbação seja igual a $Nperturb$, então são feitas $Nperturb + 1$ trocas aleatórias. O nível de perturbação é aumentado em uma unidade quando a busca local não resultar em melhoria da solução corrente e quando no mínimo 10% da vizinhança desta solução tenha sido explorada. Quando há melhora, o nível de perturbação volta para seu nível mínimo, isto é, 1, voltando a se repetir o processo. O método pára após um número máximo de iterações sem melhora no valor da melhor solução encontrada até então.

O método move de uma solução para outra somente se o ótimo local encontrado for melhor que o ótimo local corrente, ou seja, se $f(v'') < f(v)$.

O método ILS proposto é esquematizado na Fig. 3.14.

Foram desenvolvidos três métodos baseados na meta-heurística ILS:

(*i*) O primeiro, denominado ILS-MRD-R, utiliza como método de busca local do ILS o MRD com a estrutura de vizinhança $N^{(R)}$.

(*ii*) Já o segundo, denominado ILS-MRD-T, utiliza como método de busca local do ILS o MRD com a estrutura de vizinhança $N^{(T)}$.

(*iii*) E, finalmente, o terceiro método, denominado ILS-VND-RT, utiliza o método

VND. Esta metodologia adota como método de busca local o MRD e as as estruturas de vizinhança $N^{(R)}$ e $N^{(T)}$, nesta ordem. Na Seção 4.3.1 é justificado porque as estruturas de vizinhança são utilizadas nesta ordem.

```
procedimento ILS
1  v_0 ← Constroi_Solucao_Inicial(n, v_0, MaxIter);
2  v ← MetodoBuscaLocal(n, v_0);
3  nivel ← 1;
4  enquanto (os critérios de parada não forem satisfeitos) faça
5      v' ← Perturbacao(nivel, v);
6      v'' ← MetodoBuscaLocal(n, v');
7      se (f(v'') < f(v)) então
8          v ← v'';
9          f(v) ← f(v'');
10         nivel ← 1;
11     senão
12         se (pelo menos 10% da vizinhança no nível corrente tenha sido explorada)
13             então nivel ← nivel + 1;
14         fim-se
15     fim-se
16 fim-enquanto;
17 Retorne v;
fim ILS;
```

Figura 3.14: Algoritmo *ILS*

Como o MRD não garante que a solução encontrada é um ótimo local, aplicou-se ao final dos três métodos o procedimento VND, adotando como método de busca local o MD e $N^{(R)}$ como primeira estrutura de vizinhança e $N^{(T)}$ como segunda estrutura de vizinhança.

3.3.2.7 GRASP

Como dito anteriormente, o método GRASP é composto de duas fases: (i) fase de construção e (ii) fase de melhoramento (busca local). Para resolver o PSUMAA foram feitas as seguintes adaptações:

a) Uma fase preliminar de geração de solução inicial utilizando o método descrito na Seção 3.3.2.4, Fig. 3.10

b) Uma fase de construção parcialmente gulosa também utilizando o mesmo método anterior

c) Uma fase de refinamento aplicado à solução construída, utilizando os métodos de busca local descritos na Seção 3.3.2.5

A Fig. 3.15 ilustra o pseudo-código do método GRASP proposto para resolver o problema de seqüenciamento em questão.

```
procedimento GRASP(n, v, IterMax)
1   v_0 ← Constroi_Solucao_Inicial(n, v, MAX_ITER);
2   v ← MetodoBuscaLocal(n, v_0);
3   f* ← f(v);
4   Iter ← 0;
5   enquanto (Iter < IterMax) faça
6       v' ← Constroi_Solucao_Inicial(n, v, MAX_ITER);
7       v' ← MetodoBuscaLocal(n, v');
8       se (f(v') < f*) então
9           v ← v';
10          f* ← f(v');
11      fim-se
12      Iter ← Iter + 1;
13  fim-enquanto;
14  Retorne v;
fim GRASP;
```

Figura 3.15: Algoritmo *GRASP* aplicado ao PSUMAA

O método GRASP é executo por um certo número máximo de iterações ($IterMax$).

De forma análoga ao método ILS, desenvolveu-se três métodos baseados na meta-heurística GRASP:

(*i*) O primeiro método, denominado GRASP-MRD-R, utiliza como método de busca local do GRASP o MRD com a estrutura de vizinhança $N^{(R)}$.

(*ii*) O segundo método, denominado GRASP-MRD-T, utiliza como método de busca local do GRASP o MRD com a estrutura de vizinhança $N^{(T)}$.

(*iii*) E o terceiro método, denominado GRASP-VND-RT, utiliza o método VND. O VND utiliza o MRD como método de busca local e e as estruturas de vizinhança $N^{(R)}$ e $N^{(T)}$, nesta ordem.

Assim como no método ILS, aplicou-se ao final destes três métodos o procedimento VND, utilizando o MD e adotando as estruturas de vizinhança $N^{(R)}$ e $N^{(T)}$, nesta ordem.

Capítulo 4

Resultados e Análise

Neste capítulo são apresentados e analisados os resultados obtidos pela modelagem matemática do problema e pelos métodos heurísticos propostos. São apresentados, também, os procedimentos usados para determinar os principais parâmetros dos métodos utilizados e como foram gerados os problemas-teste para o problema. Uma comparação entre os resultados obtidos pelas diversas metodologias é feita objetivando-se analisar a eficiência das metodologias propostas.

4.1 Geração dos Problemas-Teste

Os problemas-teste foram gerados através de métodos pseudo-aleatórios, baseados nos trabalhos de Wan e Yen [34], Liaw [24] e Mazzini e Armentano [26], com número de *jobs* n igual a 6, 7, 8, 9, 10, 11, 12, 15, 20, 25, 30, 40, 50 e 75.

Os tempos de processamento (P_i) dos *jobs* são uniformemente e discretamente distribuídos entre 1 e 100.

Os centros das janelas de entrega são uniformemente e discretamente distribuídos dentro do intervalo $[(1 - TF - RDD/2) \times MS, (1 - TF + RDD/2) \times MS]$, onde MS é o tempo total de processamento de todos os *jobs*, TF é o fator de atraso e RDD é a variação relativa da janela de entrega. Os valores considerados para TF foram 0,1; 0,2; 0,3 e 0,4 e os valores considerados para RDD foram 0,8; 1,0 e 1,2. Os tamanhos das janelas de entrega são uniformemente e discretamente distribuídas no intervalo $[1, MS/n]$.

Os custos por atraso da produção (β_i) são uniformemente e discretamente

distribuídos dentro do intervalo $[20, 100]$. Na maioria dos casos reais, o atraso da produção é menos desejável do que a sua antecipação. Desta forma, os custos por antecipação da produção (α_i) são gerados como k vezes o custo por atraso do mesmo *job*, sendo k uma variável aleatória uniforme entre 0 e 1.

Os tempos de preparação da máquina (S_{ij}) são gerados uniformemente e discretamente dentro do intervalo $[0, 50]$. Utilizou-se tempos de preparação simétricos, ou seja, $S_{ij} = S_{ji}$.

Como TF possui quatro valores diferentes e RDD três valores diferentes, foram gerados um total de 12 problemas-teste para cada número de *jobs*, totalizando 180 problemas-teste.

4.2 Resultados da Modelagem Matemática

O modelo matemático apresentado na Seção 3.2 foi implementado usando a ferramenta de modelagem AMPL e resolvido pelo *software* de otimização CPLEX, versão 9.1. Os testes com o modelo foram realizados em um computador Atlon XP 3000+ 64 bits (aproximadamente 2 GHz), com 1 GB de memória RAM, sob plataforma Windows XP.

Primeiramente, o modelo matemático foi testado em três cenários diferentes.

O primeiro cenário criado possui 6 *jobs* a serem seqüenciados. A Tab. 4.1 apresenta as principais informações sobre cada um deste *jobs*. São apresentados os tempos de processamento (P_i), a janela de entrega $[E_i, T_i]$ considerada e as penalidades por atraso (α_i) e antecipação (β_i) da produção. Nesta tabela é apresentada, ainda, a matriz com os tempos de preparação da máquina.

Tabela 4.1: Dados do Cenário 1

Jobs	**Dados do Problema**					**Tempo de Preparação da Máquina**					
	P_i	E_i	T_i	α_i	β_i	**1**	**2**	**3**	**4**	**5**	**6**
1	5	9	13	10	100	0	1	1	2	4	3
2	6	12	18	15	150	1	0	3	2	1	2
3	5	9	12	10	100	1	3	0	2	4	3
4	4	15	21	20	200	2	2	2	0	2	2
5	3	20	26	30	300	4	1	4	2	0	5
6	4	12	16	25	250	3	2	3	2	5	0

O segundo cenário é idêntico ao primeiro, diferenciando-se apenas em relação à janela de entrega. Esta mudança na janela de entrega teve como objetivo forçar o custo total a ser igual a zero. A janela de entrega utilizada neste cenário é apresentada na Tab. 4.2.

Tabela 4.2: Dados do Cenário 2

Jobs	**Janelas de Entrega**	
	E_i	T_i
1	19	23
2	7	11
3	15	19
4	39	43
5	43	47
6	27	31

Os resultados para o primeiro e segundo cenários se mostraram satisfatórios conforme pode ser observado nas Tabelas 4.3 e 4.4 e nos gráficos de Gantt mostrados nas Figuras 4.1 e 4.2. No Cenário 2, observou-se que o custo total foi igual a 0 (zero), como era esperado.

Tabela 4.3: Resultados do Cenário 1

Seqüência de *jobs*	s_i	C_i	e_i	t_i	**Custo de Estoque**	**Custo por Atraso**
3	0	5	4	0	40	0
6	8	12	0	0	0	0
4	14	18	0	0	0	0
5	20	23	0	0	0	0
2	24	30	0	12	0	1800
1	31	36	0	23	0	2300
Total					40	4100
Custo Total						4140

O terceiro cenário foi criado para verificar se o modelo é capaz de resolver problemas-teste onde a desigualdade triangular não é satisfeita. Os dados deste cenário são mostrados na Tab. 4.5.

No terceiro cenário a desigualdade triangular não é satisfeita, por exemplo, tem-se que $S_{32} < S_{36} + S_{62} + P_6$ não é verificada, pois $50 > 45$. Como pode ser observado na Tab. 4.6 e na Fig. 4.3, mesmo esta condição não sendo satisfeita, o modelo conseguiu resolver o problema. Este mesmo cenário foi resolvido com o modelo

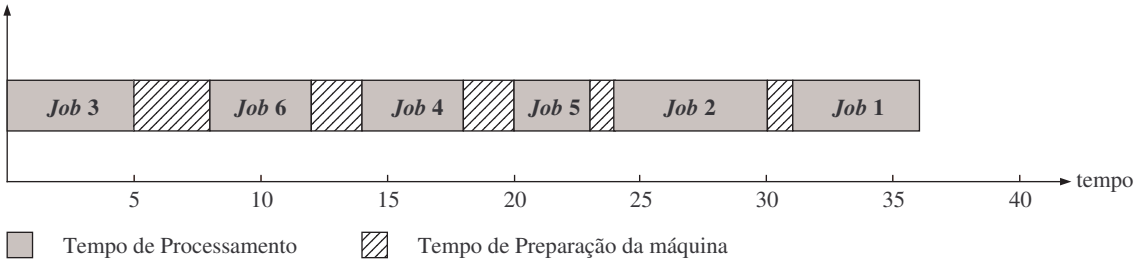

Figura 4.1: Gráfico de Gantt para o Cenário 1

Tabela 4.4: Resultados do Cenário 2

Seqüência de *jobs*	s_i	C_i	e_i	t_i	**Custo de Estoque**	**Custo por Atraso**
2	1	7	0	0	0	0
3	10	15	0	0	0	0
1	16	21	0	0	0	0
6	24	28	0	0	0	0
4	35	39	0	0	0	0
5	41	44	0	0	0	0
Total					0	0
Custo Total						0

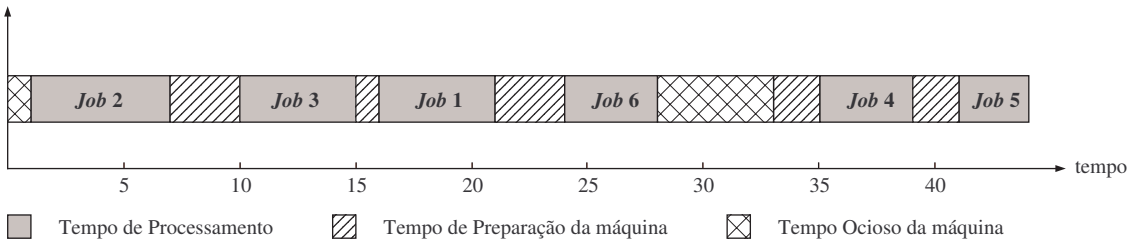

Figura 4.2: Gráfico de Gantt para o Cenário 2

Tabela 4.5: Dados do Cenário 3

Jobs	**Dados do Problema**					**Tempo de Preparação da Máquina**					
	P_i	E_i	T_i	α_i	β_i	**1**	**2**	**3**	**4**	**5**	**6**
1	60	120	140	10	100	0	40	50	20	30	10
2	50	230	250	15	150	40	0	50	5	30	5
3	100	40	80	10	100	50	50	0	20	10	10
4	80	290	310	20	200	20	5	20	0	50	15
5	10	50	70	30	300	30	30	10	50	0	30
6	20	220	260	25	250	10	5	20	15	30	0

proposto por Bustamante [6] apresentando a mesma seqüência de *jobs*, só que o *job* 6 inicia seu processamento na data 145 ao invés da data 140, desta forma gerando um custo total maior (31225).

Tabela 4.6: Resultados do Cenário 3

Seqüência de *jobs*	s_i	C_i	e_i	t_i	**Custo de Estoque**	**Custo por Atraso**
5	0	10	40	0	1200	0
3	20	120	0	40	0	4000
6	140	160	60	0	1500	0
2	165	215	15	0	225	0
4	220	300	0	0	0	0
1	320	380	0	240	0	24000
Total					2925	28000
Custo Total						30925

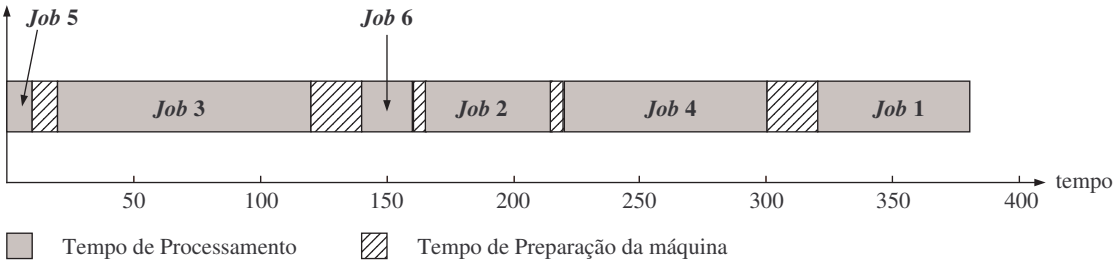

Figura 4.3: Gráfico de Gantt para o Cenário 3

Com estes resultados viu-se que o modelo é capaz de representar o PSUMAA com tempo de preparação da máquina dependente da seqüência de produção e com janelas de entrega.

O próximo passo é a aplicação deste modelo aos problemas-teste de pequeno porte gerados. Foram resolvidos os problemas-teste de 6 a 12 *jobs* e os resultados são apresentados na Tab. 4.7. Nesta tabela a primeira coluna apresenta o número de *jobs* do conjunto de problemas-teste, a segunda coluna apresenta o tempo computacional médio gasto para se chegar à solução ótima e a última coluna apresenta o percentual de soluções ótimas encontradas dentro dos 12 problemas-teste para cada número de *jobs* dentro do tempo limite estabelecido. Considerou-se um tempo de 3600 segundos como tempo limite para obtenção de uma solução ótima.

Como pode ser observado na Tab. 4.7, o modelo proposto se mostra eficiente para resolver problemas-teste com até 12 *jobs*.

Tabela 4.7: Resultados obtidos pelo CPLEX 9.1

Número de *jobs*	**Tempo Computacional Médio (s)**	**Percentual de soluções ótimas encontradas (%)**
06	0,08	100
07	0,28	100
08	2,22	100
09	43,38	100
10	93,39	100
11	1931,25	75
12	2022,72	50

4.3 Resultados dos Métodos Heurísticos

Os primeiros testes com os métodos propostos têm o objetivo de definir os valores dos parâmetros, os quais são utilizados nos testes finais. A Seção 4.3.1 apresenta como foram obtidos os principais parâmetros dos métodos de geração da solução inicial, dos métodos de busca local e dos métodos GRASP e ILS utilizados.

Definidos os parâmetros de entrada, os métodos GRASP e ILS foram aplicados ao conjunto de problemas-teste gerados. Os resultados destes testes são apresentados ao final desta seção.

Todos os testes foram realizados em um computador Atlon XP 64 bits 3000+ (aproximadamente 2 GHz), com 1 GB de memória RAM, sob plataforma Windows XP. Os métodos foram desenvolvidos em linguagem C, utilizando o ambiente Borland C++ Builder 5.0.

4.3.1 Definição dos Parâmetros

O primeiro parâmetro a ser determinado foi o parâmetro γ do procedimento de construção GRASP apresentado na Seção 3.3.2.4. Os testes foram realizados com 4 problemas-teste de 10, 15, 20 e 25 *jobs*, totalizando 16 problemas. O gráfico apresentado na Fig. 4.4 mostra os resultados dos testes realizados para o parâmetro γ. Para a confecção deste gráfico os resultados obtidos foram padronizados de acordo com a equação (4.1). Nesta equação, VP é o valor padronizado, Res é o resultado obtido, $Minimo$ é o menor valor obtido e $Maximo$ é o maior valor obtido. Para cada valor de γ e para cada problema-teste, o procedimento de construção GRASP foi aplicado 20 vezes. Os valores de γ variaram de 0 a 0,30.

$$VP = 100 * \left(\frac{Res - Minimo}{Maximo - Minimo} \right) \tag{4.1}$$

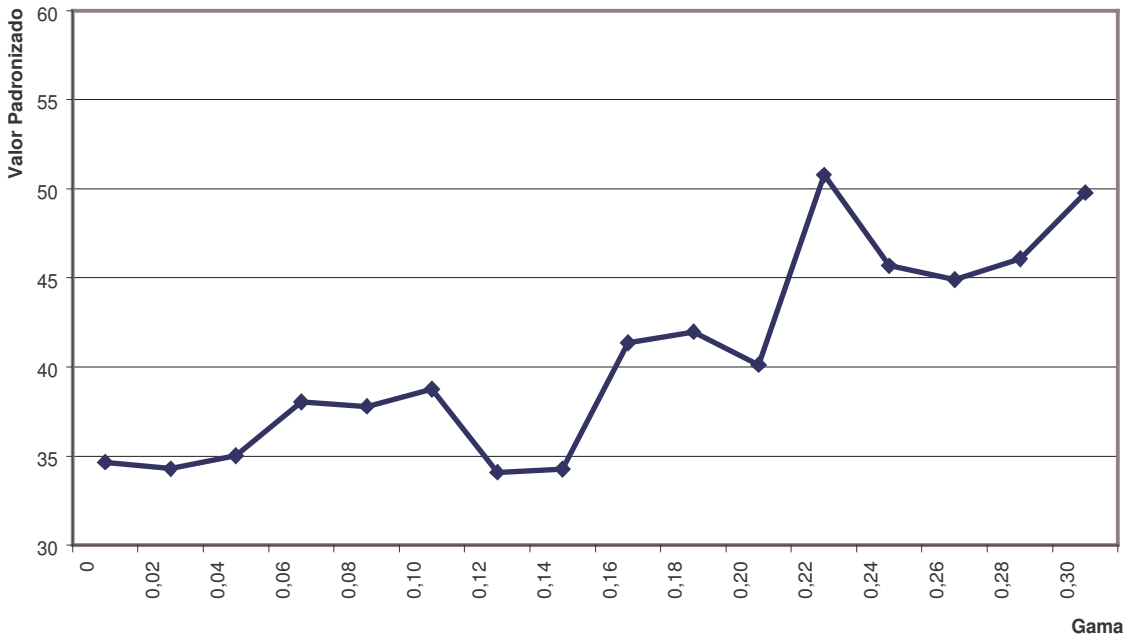

Figura 4.4: Gráfico - Valor Padronizado $\times$ γ

Como pode ser observado no gráfico da Fig. 4.4, os valores de γ que propiciaram os melhores resultados foram: 0,02; 0,04; 0,12 e 0,14 (neste caso, quanto menor o valor melhor o resultado). Desta forma, estes foram os valores de γ adotados para o procedimento de construção da solução inicial.

Definidos os valores de γ, o próximo passo foi determinar o número máximo de iterações ($IterMax1$) do procedimento de construção da solução inicial. Para isto, foram realizados testes com valores variando de 4 a 100 iterações em um determinado problema-teste de 15 *jobs*. Para cada valor de $IterMax1$, aplicou-se o procedimento de construção da solução inicial 20 vezes. Os resultados obtidos foram avaliados através do percentual de melhora obtido em relação aos resultados obtidos pelo método guloso ($\gamma = 0$). O gráfico da Fig. 4.5 apresenta o percentual médio de melhora obtido com cada número de iterações. O percentual médio é obtido pela média simples dos percentuais de melhora em relação à função gulosa.

Com base no gráfico da Fig. 4.5, adotou-se o valor de $IterMax1$ igual a 52, pois a partir deste valor a diferença de percentual não é muito significativa, não

compensando o tempo computacional despendido.

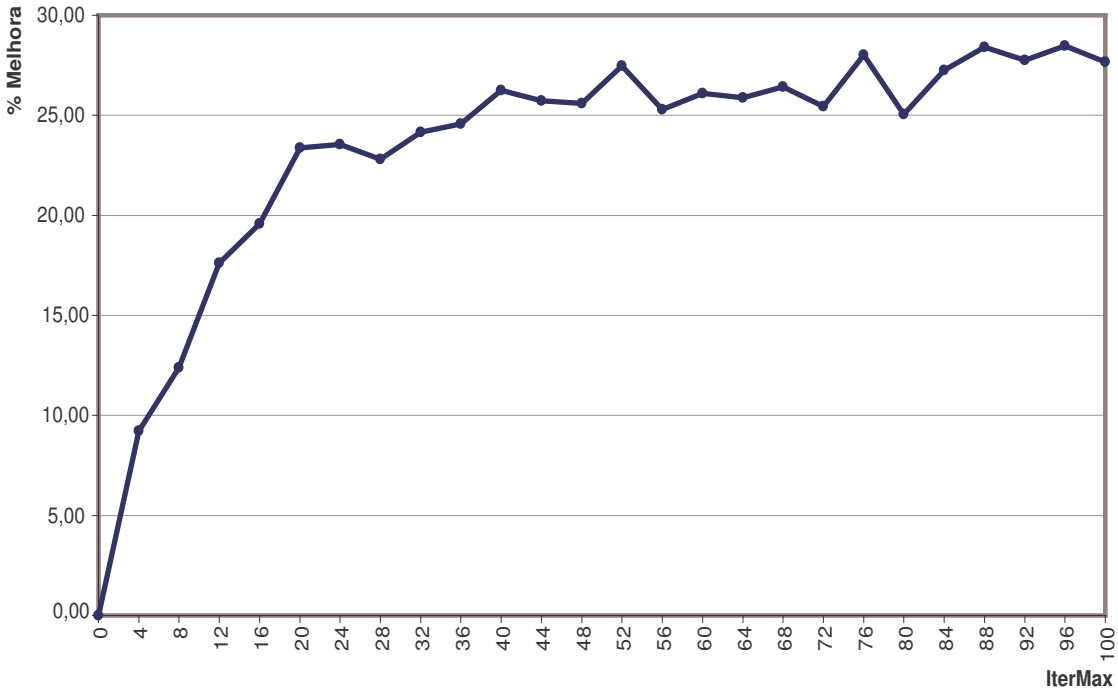

Figura 4.5: Gráfico - Percentual Médio de Melhora × Número Máximo de Iterações

Definidos os parâmetros para a construção da solução inicial, necessita-se determinar o número máximo de iterações dos métodos de busca local utilizados. Para isto, tomou-se um conjunto de problemas-teste, gerou-se uma solução inicial para cada problema-teste e aplicou-se o Método Randômico de Descida (MRD) desejado. Foram utilizadas 12 problemas-teste de 40 *jobs* para a realização dos testes. O MRD foi executado 20 vezes para cada problema-teste. O número máximo de iterações é dado por $(k \times n)$ iterações. Os gráficos das Fig. 4.6 e Fig. 4.7 apresentam os resultados para o MRD usando a estrutura de vizinhança $N^{(T)}$, nomeado método MRD-T, e $N^{(R)}$, nomeado método MRD-R, respectivamente. Estes gráficos trazem o percentual médio de melhora em relação à variável k. Em ambos os gráficos o percentual médio de melhora é obtido pela média dos percentuais de melhora obtidos pela aplicação do MRD à solução inicial gerada. O percentual de melhora é a diferença percentual entre o valor da solução inicial obtida e o valor da solução obtida com a aplicação do MRD correspondente.

No gráfico da Fig. 4.6 pode-se observar que do ponto $k = 8$ para o ponto $k = 9$ a variação no percentual médio de melhora é muito pequeno ($< 0,5\%$), então, adotou-se o número máximo de iterações ($IterMax2$) igual a $8 \times n$ para o MRD-T.

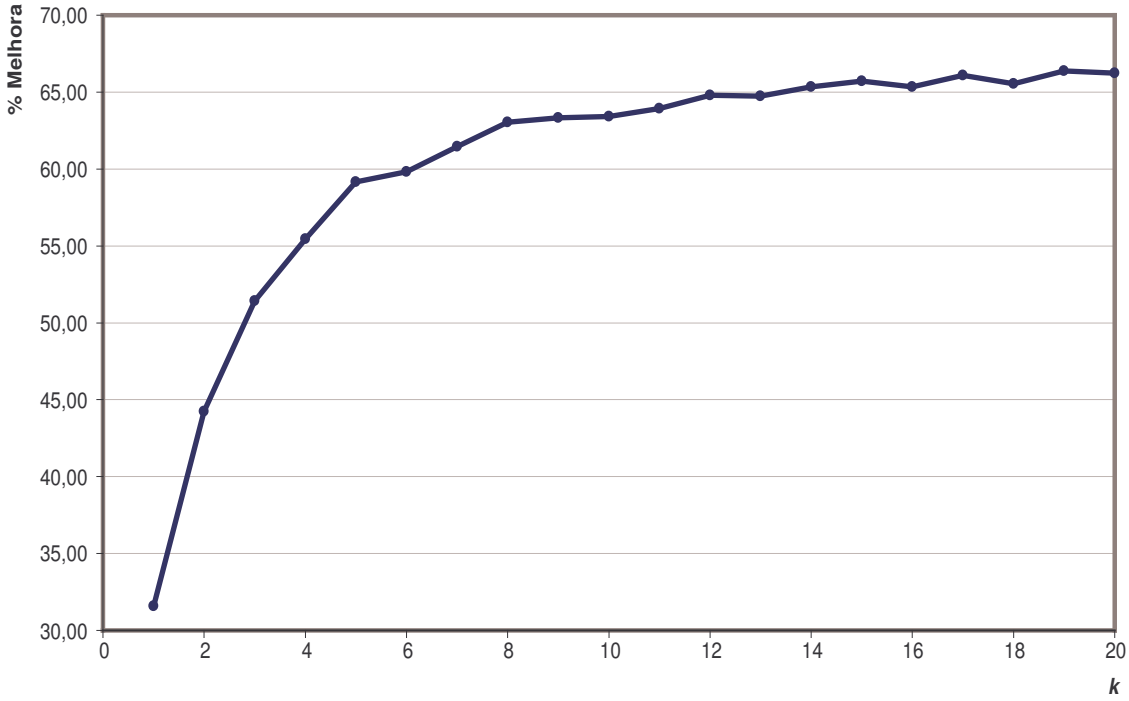

Figura 4.6: Gráfico - Percentual Médio de Melhora × Variação de k

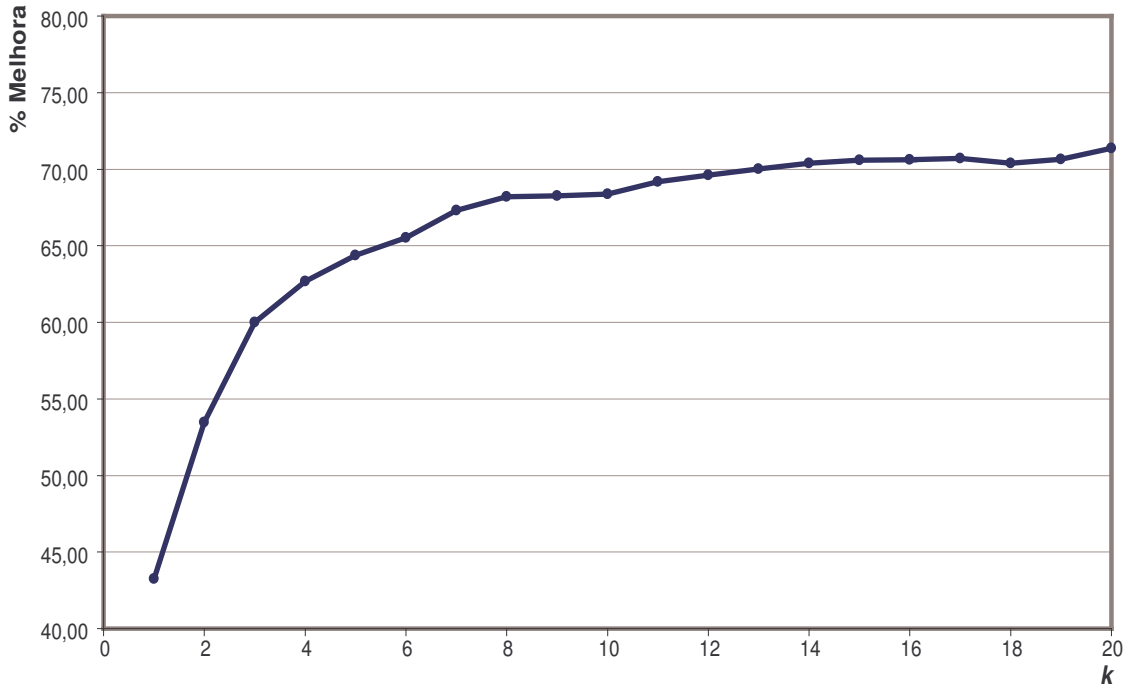

Figura 4.7: Gráfico - Percentual Médio de Melhora × Variação de k

De forma análoga, no gráfico da Fig. 4.7, a variação do percentual médio de melhora do ponto $k = 8$ para o $k = 9$ é muito pequena ($< 0,5\%$). Então, também foi adotado o número máximo de iterações ($IterMax3$) igual a $8 \times n$ para o MRD-R.

Pode-se observar, ainda, nestes gráficos, que existem valores de k que apresentam um percentual médio de melhora maior que o ponto $k = 8$, mas, nestes casos, os métodos despendem um tempo computacional muito maior, não sendo vantajoso utilizá-los.

Antes de definir o número máximo de iterações sem melhora ($IterMax4$) do MRD usado pelo método VND, fez-se necessário determinar qual seria a primeira estrutura de vizinhança: MRD-T ou MRD-R.

Tabela 4.8: Resultados dos Testes para o VND

k	**MRD-R → MRD-T % Médio de Melhora**	**MRD-T → MRD-R % Médio de Melhora**
1	46,01	44,23
2	56,99	55,97
3	62,08	61,58
4	65,54	63,89
5	66,92	66,80
6	68,41	67,38
7	69,24	68,41
8	69,63	69,11
9	70,90	69,58
10	70,62	70,25

Tal como nos testes do MRD, foram utilizados 12 problemas-teste de 40 *jobs*. Para cada valor de k, aplicou-se o método VND 20 vezes. Os resultados para o método VND são apresentados na Tab. 4.8. Como pode ser observado nesta tabela, o VND usando o MRD com $N^{(R)}$ como primeira estrutura de vizinhança conseguiu melhor desempenho do que o VND usando o MRD com $N^{(T)}$ como primeira estrutura de vizinhança, para qualquer valor de k. Então decidiu-se pela utilização do método VND utilizando o MRD com $N^{(R)}$ como primeira estrutura de vizinhança.

Observando-se, ainda, a Tab. 4.8, conclui-se que o valor de k para o MRD do método é 7, pois do ponto $k = 7$ para o ponto $k = 8$ tem-se uma diferença no percentual médio de melhora menor do que 0,5%. Portanto, o número máximo de iterações sem melhora ($IterMax4$) do MRD usado pelo método VND é igual a $7 \times n$.

Finalmente, os últimos parâmetros a serem definidos foram o número máximo de iterações do método GRASP e o número máximo de iterações sem melhora do método ILS ($IterMax5$ e $IterMax6$, respectivamente). Estes parâmetros são dados por $k_1 \times n$ e para determinação do valor de k_1 foram utilizados 8 problemas-teste de 25 *jobs*. Para cada valor de k_1, foram aplicados os métodos GRASP e ILS 20 vezes.

O método GRASP utilizado para definição de $IterMax5$ consta do método de construção da solução inicial e utiliza o MRD-T como método de busca local. O método utiliza os parâmetros definidos anteriormente.

Para cada valor de k_1 e para cada problema-teste, determinou-se o resultado médio obtido pelo método ($RMed$). Este resultado é então comparado com o melhor resultado obtido para cada problema (MR), fornecendo o GAP, sendo este definido pela equação 4.2.

$$GAP = 100 * \left(\frac{RMed - MR}{MR} \right) \tag{4.2}$$

Então, para cada valor de k_1, determinou-se o GAP médio, que é a média dos valores do GAP de cada problema-teste.

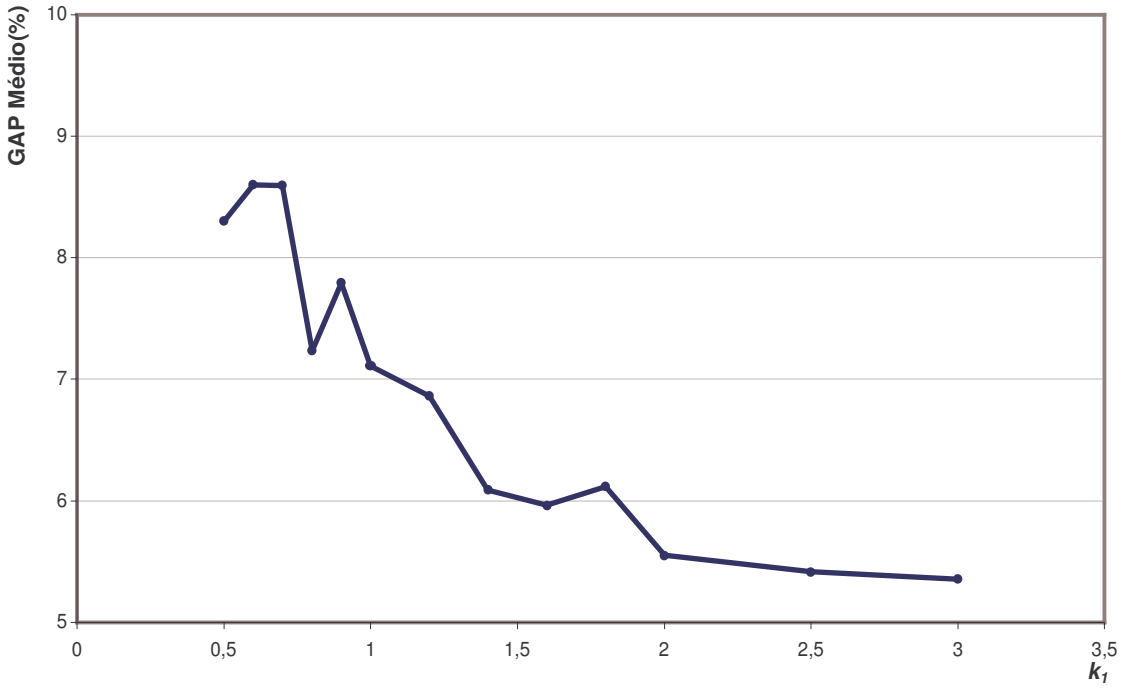

Figura 4.8: Gráfico - GAP médio × Variação de k_1

O gráfico da Fig. 4.8 apresenta os resultados dos GAP médios para o método GRASP. Como pode ser observado, a partir do ponto $k_1 = 2$ não se tem uma melhora significativa do valor do GAP médio. Sendo assim, adotou-se $k_1 = 2$. Desta forma, $IterMax5 = 2 \times n$.

Para definição de $IterMax6$, utilizou-se o MRD-T como método de busca local

do método ILS. Antes da aplicação das primeiras perturbações do método ILS, gera-se uma solução inicial, a qual é refinada pelo método MRD-T. O percentual de melhora é obtido pela diferença percentual entre o valor da solução obtida com o MRD-T e o valor da solução final obtida com a aplicação das perturbações. O percentual médio de melhora é a média dos percentuais de melhora. Para cada valor de k_1 determinou-se a média dos percentuais médios de melhora. Como estes valores estavam muito próximos, resolveu-se adotar como forma de determinar o número de iterações do método a divisão da média dos Percentuais Médios de Melhora pelo Tempo computacional médio. Estes resultados são apresentados na Tab. 4.9.

Tabela 4.9: Resultados dos Testes com ILS para Determinar $IterMax6$

k_1	**Média do Percentuais Médios de Melhora ($MPMM$)**	**Tempo Computacional Médio (TCM) (s)**	$\frac{MPMM}{TCM}$
0,5	67,49	5,09	13,26
0,6	67,28	5,59	12,04
0,7	67,23	6,70	10,03
0,8	67,96	8,02	8,47
0,9	67,76	8,52	7,96
1,0	68,40	9,95	6,87
1,2	67,58	10,73	6,30
1,4	68,44	12,65	5,41
1,6	68,50	14,73	4,65
1,8	68,60	15,68	4,37
2,0	68,74	16,93	4,06
2,5	69,19	21,59	3,21
3,0	69,46	23,37	2,97

Observando-se a Tab. 4.9 e levando-se em consideração que o percentual de melhora por segundo não deve ser inferior a 5, adotou-se $k_1 = 1,4$. Portanto, $IterMax6 = 1,4 \times n$.

Resumindo, os parâmetros utilizados para os testes com os métodos propostos são apresentados na Tab. 4.10.

4.3.2 Resultados Finais dos Métodos GRASP e ILS

Antes dos testes finais, realizou-se um primeiro conjunto de testes com 12 problemas-teste envolvendo 25 *jobs*. Este primeiro conjunto de testes teve como finalidade determinar quais seriam os métodos GRASP e ILS a serem utilizados nos testes

Tabela 4.10: Tabela de Parâmetros Utilizados

Parâmetro	**Método**	**Valor(es)**
γ	Método de Construção GRASP	0; 0,02; 0,04; 0,12 e 0,14
$IterMax1$	Método de Construção da Solução Inicial	52
$IterMax2$	MRD-T	$8 \times n$
$IterMax3$	MRD-R	$8 \times n$
$IterMax4$	Método VND	$7 \times n$
$IterMax5$	Método GRASP	$2 \times n$
$IterMax6$	Método ILS	$1,4 \times n$

finais.

Neste primeiro conjunto de testes, cada problema-teste foi resolvido 30 vezes com cada um dos métodos propostos. Os resultados deste primeiro conjunto de testes é apresentado na Tab. 4.11. Nesta tabela, o *GAP* é obtido pela Equação 4.2. O *GAP* médio é obtido pela média dos *GAP*. O valor do GAP-MS é obtido pela Equação 4.2, substituindo-se o resultado médio pelo melhor resultado obtido pelo método. O valor de MR corresponde ao melhor resultado obtido para o problema-teste com a aplicação de todos os métodos.

Tabela 4.11: Resultados do Primeiro Conjunto de Testes

Método	**Média dos GAP-MS (%)**	**Média dos GAP médios (%)**	**Tempo Computacional médio (s)**
ILS-MRD-T	0,57	5,97	11,91
ILS-MRD-R	0,00	2,76	10,94
ILS-VND-RT	0,00	2,10	13,11
GRASP-MRD-T	0,79	7,77	11,30
GRASP-MRD-R	0,00	3,94	10,89
GRASP-VND-RT	0,84	3,35	13,56

Como pode ser observado na Tab. 4.11, o melhor desempenho em relação ao *GAP* médio foram os métodos ILS-VND-RT e GRASP-VND-RT. Estes dois métodos foram utilizados para resolver o conjunto de problemas-teste gerado.

Foram resolvidos os problemas-teste com 8 a 75 *jobs*, totalizando 144 problemas. Cada qual foi resolvido 30 vezes pelos dois métodos escolhidos, com exceção dos problemas-teste de 75 *jobs*, os quais foram resolvidos apenas 20 vezes, pois demandavam um tempo computacional bastante elevado. Ainda, para os problemas-teste de 75 *jobs* adotou-se o valor $2 \times n$ como número máximo de iterações sem melhora do método VND. Este fato se deve, também, ao alto tempo computacional demandado.

As Tabelas 4.12 e 4.13 apresentam os resultados para os métodos ILS-VND-RT e GRASP-VND-RT, respectivamente. Nestas tabelas, os resultados dos métodos nos problemas-testes com 8 a 12 *jobs* foram comparados com os resultados ótimos obtidos pelo modelo matemático. Para os demais problemas-teste, os valores foram comparados com o melhor resultado obtido pela aplicação das diversas metodologias heurísticas.

Tabela 4.12: Resultados do Método ILS-VND-RT

Número de Jobs	**Média dos GAP médios (%)**	**Média dos GAP-MS (%)**	**Tempo Computacional médio (s)**
8	0,03	0,00	0,04
9	0,06	0,00	0,07
10	0,02	0,00	0,11
11	0,12	0,00	0,20
12	0,21	0,00	0,29
15	1,47	0,00	0,94
20	1,65	0,00	4,35
25	2,32	0,00	13,29
30	3,34	0,20	40,07
40	4,38	0,18	155,79
50	6,13	0,28	492,28
75	10,89	0,56	1.368,08
Média	2,55	0,10	-

Tabela 4.13: Resultados do Método GRASP-VND-RT

Número de Jobs	**Média dos GAP médios (%)**	**Média dos GAP-MS (%)**	**Tempo Computacional médio (s)**
8	0,04	0,00	0,06
9	0,66	0,00	0,10
10	0,37	0,00	0,17
11	0,55	0,00	0,26
12	0,29	0,27	0,38
15	1,23	0,00	1,12
20	2,24	0,00	4,55
25	3,17	0,00	13,48
30	5,05	0,59	39,08
40	7,33	1,39	156,98
50	10,67	1,91	496,21
75	13,69	1,47	1.856,27
Média	3,77	0,47	-

Conforme se observa nas Tabelas 4.12 e 4.13, ambos os métodos mostraram desempenhos parecidos com uma pequena vantagem (pouco mais de 1%) para o método

ILS-VND-RT. Este obteve um desempenho melhor tanto no desvio da melhor solução (GAP-MS) quanto no desvio médio (*GAP* médio). Os tempos computacionais estão bem próximos, com exceção dos problemas-teste envolvendo 75 *jobs*, onde o método GRASP-VND-RT gastou, em média, 488 segundos a mais que o método ILS-VND-RT para produzir uma solução. Observa-se, ainda, que nos problemas-teste de 8 a 12 jobs a maioria dos resultados alcançou a solução ótima. Este fato pode ser observado pelo pequeno desvio médio produzido e pelos desvios da melhor solução (quase todos iguais a 0%). Estes dados servem também para validar o método, pois os resultados obtidos com o modelo matemático foram os mesmos obtidos com os métodos propostos.

Capítulo 5

Conclusão e Trabalhos Futuros

Este trabalho teve seu foco no desenvolvimento de métodos de otimização para a resolução do problema de seqüenciamento em uma máquina com penalidades por antecipação e atraso da produção, com tempo de preparação da máquina dependente da seqüência de produção e janelas de entrega. Este problema tem diversas aplicações práticas, como por exemplo, na indústria metalúrgica (BUSTAMANTE [6]).

Propôs-se, inicialmente, um modelo de programação linear inteira mista para representar o problema estudado. Desejava-se definir um modelo de programação linear inteira mista, o qual pudesse ser resolvido por métodos exatos. Os modelos até então desenvolvidos necessitavam de adaptações nos dados de entrada para que pudessem ser utilizados. O modelo proposto tem como principal objetivo fechar estas pequenas lacunas deixadas pelos modelos existentes na literatura.

O modelo de programação linear inteira mista foi resolvido através do CPLEX 9.1 e foi capaz de resolver na otimalidade problemas com até 12 *jobs*.

Mesmo limitado a problemas de pequenas dimensões, os modelos matemáticos são importantes pois a resolução destes modelos fornecem dados para a comparação e a validação de resultados obtidos por métodos aproximados (mesmo quando a solução ótima não é encontrada é fornecido o limite inferior para a solução ótima).

Propôs-se, também, métodos heurísticos baseados nas meta-heurísticas GRASP e ILS. Estes métodos foram divididos em duas fases: determinar a seqüência de *jobs* e a partir desta seqüência determinar a data ótima de início de processamento de cada *job*. Em todos os métodos, a solução inicial é gerada por um procedimento baseado na fase de construção GRASP e a busca local feita pelo método randômico de descida.

Ao final de cada método é aplicado o método de descida.

Quanto à precisão dos métodos heurísticos, estes apresentaram resultados bem satisfatórios. Em problemas de pequenas dimensões (8 a 12 *jobs*), estes métodos, na maioria das vezes, conseguiram alcançar a otimalidade. Os resultados encontrados pelas heurísticas propostas desviam dos resultados ótimos em menos de 0,5%. Em problemas com dimensões maiores (15 a 75 *jobs*), os desvios da melhor solução também foram bem pequenos, em média 5,3%. Os métodos baseados na meta-heurística ILS mostraram uma pequena vantagem (um pouco mais de 1%) sobre o método baseado na meta-heurística GRASP.

Os tempos computacionais obtidos pelos métodos heurísticos foram razoavelmente bons se comparados ao horizonte de planejamento, que, tipicamente, é de uma semana. Os problemas de pequenas dimensões foram resolvidos, em média, em 0,17 segundos enquanto pela programação matemática gastou-se em torno de 818 segundos. Já os problemas maiores exigiram um esforço computacional maior, em torno de 332 segundos. Em termos de tempo computacional, os métodos baseado em GRASP e ILS se mostraram quase idênticos, com exceção dos problemas-teste com 75 jobs, onde o GRASP gastou, em média, 488 segundos a mais que o ILS.

Os dois métodos heurísticos desenvolvidos são bem simples de serem implementados, pois o único parâmetro a ser definido é o número máximo de iterações sem melhora. Além disso, são bem flexíveis, ou seja, é fácil incorporar novas restrições ao problema.

Como trabalho futuro, sugere-se a utilização de técnicas de intensificação, como a aplicação da reconexão por caminhos (*Path Relinking*), com a finalidade de obter resultados ainda melhores. Por esta técnica, um conjunto de soluções de qualidade, conhecido como conjunto elite, seria formado durante a exploração do espaço de busca pelos métodos heurísticos propostos. Periodicamente, pares de elementos desse conjunto são isolados para proceder a uma busca. De cada par de soluções elite, uma solução, dita corrente, é utilizada como solução inicial e a outra solução como guia. Atributos da solução guia são, então, adicionados passo a passo à solução corrente, a qual é submetida a um processo de refinamento. Na trajetória da solução inicial à solução guia espera-se que soluções de melhor qualidade sejam geradas. A literatura relata várias implementações bem sucedidas dessa técnica.

Aponta-se, ainda, a incorporação aos modelos de novas restrições, como por exemplo, a de que um *job* só estará disponível para processamento em uma data R_i.

Do ponto de vista prático, os métodos propostos poderiam ser aplicados em uma situação real, como por exemplo na indústria siderúrgica, têxtil, de tintas, entre outras.

Referências Bibliográficas

[1] AHUJA, R. K.; CUNHA, C. B.; ORLIN, J. B. *Network Flows.* Prentice Hall, New Jersey, 1993.

[2] ALLAHVERDI, A.; GUPTA, J. N. D.; ALDOWAISAN, T. "A Review of Scheduling Research Involving Setup Considerations". *Omega - The International Journal of Management Science*, v. 27, p. 219–239, 1999.

[3] ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. *Pesquisa Operacional para Cursos de Engenharia.* 1ª Edição, Editora Campus - Elsevier, Rio de Janeiro-RJ, 2006.

[4] BAKER, K. R. *Introduction to Sequecing and Scheduling.* John Wiley, New York-NY, 1974.

[5] BAKER, K. R.; SCUDDER, G. D. "Sequecing with Earliness and Tardiness Penalties: A Review". *Operations Research*, v. 38, p. 22–36, 1990.

[6] BUSTAMANTE, L. M. *Minimização do Custo de Antecipação e Atraso para o Problema de Seqüenciamento de uma Máquina com Tempo de Preparação Dependente da Seqüência: Aplicação em uma Usina Siderúrgica.* Dissertação de mestrado, Programa de Pós Graduação em Engenharia de Produção, UFMG, Belo Horizonte, 2006.

[7] CARVALHO, C. R. V. *Seqüenciamento - (Scheduling).* Notas de aula, Departamento de Engenharia de Produção, UFMG, Belo Horizonte-MG, 2004.

[8] CHANG, P. C. "A Branch and Bound Approach for Single Machine Scheduling with Earliness and Tardiness Penalties". *Computers & Mathematics with Applications*, v. 37, p. 133–144, 1999.

[9] CHRÉTIENNE, P.; SOURD, F. "PERT Scheduling with Convex Cost Functions". *Theorical Computer Science*, n. 292, p. 145–164, 2003.

[10] FELDMANN, M.; BISKUP, D. "Single-Machine Scheduling for Minimizing Earliness and Tardiness Penalties by Meta-heuristic Approaches". *Computers & Industrial Engineering*, v. 44, p. 307–323, 2003.

[11] FEO, T. A.; RESENDE, M. G. C. "Greedy randomized adaptive search procedures". *Journal of Global Optimization*, v. 6, p. 109–133, 1995.

[12] FRANÇA, P. M.; MENDES, A.; MOSCATO, P. "A Memetic Algorithm for the Total Tardiness Single Machine Scheduling Problem". *European Journal of Operational Research*, v. 132, p. 224–242, 2001.

[13] GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to Theory of NP-Completeness*. W. H. Freeman and Company, New York-NY, 1979.

[14] GLOVER, F. "Future paths for Integer Programming and links to Artificial Intelligence". *Computers & Operations Research*, v. 5, p. 553–549, 1986.

[15] GLOVER, F.; KOCHENBERGER, G. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Boston, 2003.

[16] GOLDBARG, M. C.; LUNA, H. P. L. *Otimização Combinatória e Programação Linear: modelos e algoritmos*. 2^a Edição, Editora Campus, Rio de Janeiro-RJ, 2005.

[17] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Berkeley, 1989.

[18] GUPTA, S. R.; SMITH, J. S. "Algorithms for single machine total tardiness scheduling with sequence dependent setups". *European Journal of Operational Research*, v. 75, p. 722–739, 2006.

[19] HINO, C. M.; RONCONI, D.P.; MENDES, A. B. "Minimizing Earliness and Tardiness Penalties in a Single-Machine Problem with a Common Due Date". *European Journal of Operational Research*, v. 160, p. 190–201, 2005.

[20] KIM, Y. D. "Minimizing Total Tardiness in Permutation Flowshops". *European Journal of Operational Research*, v. 85, p. 541–555, 1995.

[21] KIRKPATRICK, S.; GELLAT, D. C.; VECCHI, M. P. "Optimization by Simulated Annealing". *Science*, v. 220, p. 671–680, 1983.

[22] LEE, C. Y.; CHOI, J. Y. "A Genetic Algorithm for Job Sequencing Problems with Distinct Due Dates and General Early-Tardy Penalty Weights". *Computers & Operations Research*, v. 22, n. 8, p. 857–869, 1995.

[23] LI, G. "Single Machine Earliness and Tardiness Scheduling". *European Journal of Operational Research*, v. 96, p. 546–558, 1997.

[24] LIAW, C.F. "A Branch-and-Bound Algorithm for the Single Machine Earliness and Tardiness Scheduling Problem". *Computers & Operations Research*, v. 26, p. 679–693, 1999.

[25] MANNE, A. S. "On the Job-shop Scheduling Problem". *Operations Research*, v. 8, p. 219–223, 1960.

[26] MAZZINI, R.; ARMENTANO, V. A. "A Heuristic for Single Machine Scheduling with Early and Tardy Costs". *European Journal of Operational Research*, v. 128, p. 129–146, 2001.

[27] MLADENOVIC, N.; HANSEN, P. "Variable Neighborhood Search". *Computers & Operations Research*, v. 24, p. 1097–1100, 1997.

[28] PINEDO, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. *Sheduling: Theory, Algorithms and Systems.* Prentice-Hall, New Jersey, 1995.

[29] RABADI, G.; MOLLAGHASEMI, M.; ANAGNOSTOPOULOS, G. C. "A Branch-and-Bound Algorithm for the Early/Tardy Machine Scheduling Problem with a Common due-date and Sequence-Dependent Setup Time". *Computers & Operations Research*, v. 31, p. 1727–1751, 2004.

[30] SOURD, F. "Earliness-Tardiness Scheduling with Setup Considerations". *Computers & Operations Research*, v. 32, p. 1849–1865, 2005.

[31] SOUZA, M. J. F. *Inteligência Computacional para Otimização.* Notas de aula, Departamento de Computação, UFOP, Ouro Preto-MG, 2006.

[32] STAFFORD-JR, E. F.; TSENG, F. F.; GUPTA, J. N. D. "Comparative Evaluation of MILP Flowshop Models". *Journal of Operational Research Society*, v. 56, p. 88–101, 2005.

[33] WAGNER, H. M. "An Intenger Programming Model for Machine Scheduling". *Naval Research Logistics Quarterly*, v. 6, p. 131–140, 1959.

[34] WAN, G.; YEN, B. P. C. "Tabu Search for Single Machine Scheduling with Distinct Due Windows and Weighted Earliness/Tardiness Penalties". *European Journal of Operational Research*, v. 142, p. 129–146, 2002.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo