EDUARDO CORDEIRO DE SOUZA

PROGRAMAÇÃO DE TAREFAS EM UM FLOWSHOP

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

**São Paulo
2009**

PROGRAMAÇÃO DE TAREFAS EM UM FLOWSHOP

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

Área de Concentração:
Engenharia Naval

Orientador:
Prof. Dr. Oscar Brito Augusto

São Paulo
2009

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

São Paulo, de junho de 2009.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Souza, Eduardo Cordeiro
Programação de tarefas em um flow shop / E.C. de Souza
-- ed.rev. -- São Paulo, 2009
124 p.

Tese (Doutorado) - Escola Politécnica da Universidade de
São Paulo. Departamento de Engenharia Naval e Oceânica.

1. Pesquisa operacional (Otimização) 2. Heurística 3. Mode -
los matemáticos I. Universidade de São Paulo. Escola Politéc-
nica. Departamento de Engenharia Naval e Oceânica II. t.

Determinação, coragem e autoconfiança são fatores decisivos para o sucesso. Se estivermos possuídos por uma inabalável determinação conseguiremos superá-los. Independentemente das circunstâncias, devemos ser sempre humildes, recatados e despidos de orgulho.

Dalai Lama

À Sônia, minha mãe e ao meu falecido
pai.

AGRADECIMENTOS

Ao meu orientador Professor Doutor Oscar Brito Augusto pelos seus incentivos e direcionamento no andamento deste trabalho.

Aos Professores Doutores Marco Brinati e André Bergsten pela valiosa assistência no andamento deste trabalho.

Ao pessoal da secretaria do departamento de Engenharia Naval na pessoa das sras. Lânia, Sandra e Damaris e srs. César e Denílson pela colaboração.

À CAPES pelo suporte financeiro concedido.

Finalmente a todos que direta ou indiretamente colaboraram na execução deste trabalho.

Resumo

Este trabalho trata de um problema de programação de tarefas em ambiente *flow shop* com algumas características específicas que, juntas, o diferenciam dos problemas usuais. Há N tarefas a serem processadas por M máquinas independentes e cada tarefa tem seu roteiro particular ao longo da oficina (shop), não passando necessariamente por todas as máquinas; cada tarefa deve ser concluída dentro de um respectivo intervalo de tempo, designado de janela de tempo, e há punições por adiantamento e atraso na conclusão da tarefa. O desempenho da programação é medido pela soma das punições por adiantamento e atraso. Trata-se de um problema de natureza combinatória, pertencente à classe NP-Difícil, para o qual, no limite, há $(N!)^M$ alternativas.

Neste trabalho, propõe-se um modelo matemático para representação do problema; para sua resolução é utilizado o pacote de programação linear mista inteira CPLEX; dada a dificuldade da obtenção de solução exata para as instâncias maiores, são propostas heurísticas para resolução do problema. São apresentados também procedimentos combinados, utilizando uma solução inicial gerada por heurística e o modelo matemático, quer usando a estrutura geral de ramificação do CPLEX, quer usando a técnica de ramificação local (*Local Branching*).

Palavras-chaves: Pesquisa operacional. Heurísticas. Modelos matemáticos.

Abstract

This study focuses a job scheduling problem in a flow shop with some specific features, which, all together, make it different from the usual flow shop scheduling problems. There are N jobs to be processed in M different machines and each job has a particular route, skipping, eventually, one or more machines; each job should be finished within a time interval, called time window, and there are penalties for earliness and tardiness. This is a combinatorial problem for which, in the extreme case, there are $(N!)^M$ solutions, belonging to NP-Hard class.

In this study, a mathematical model is proposed for representing the problem; the CPLEX solver is used for solving the mixed integer linear problem obtained. Given the computational complexity of the model, heuristic procedures are proposed in order to solve large-scale instances of this problem. Combined procedures, using an initial solution obtained by a proposed heuristic and the mathematical model, either using the general branching procedure of CPLEX or a specific local branching procedure, are also shown.

Keywords: Operations research. Heuristics. Mathematical model

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

CAPÍTULO 1 - INTRODUÇÃO

1.1 - Sistema da produção.....	1
1.2 – Programação.....	2
1.3 - Problemas de Adiantamentos e Atrasos (Earliness/Tardiness).....	5
1.4 - Problema Proposto.....	6
1.5 - Objetivo e Relevância do Estudo.....	7
1.6 – Delineamento do Trabalho.....	9

CAPÍTULO 2 – REVISÃO BIBLIOGRÁFICA

2.1 - Introdução.....	10
2.2 - O Problema de Programação Intermitente.....	12
2.3 - <i>Flow Shops</i>	13
2.4 - Problemas do Tipo Adiantamento/Atraso (<i>Earliness/Tardiness</i>).....	17
2.5 - Métodos de Resolução de Problemas de Programação	19
2.6 - Método de Enumeração Implícita.....	22
2.7 - Método de Enumeração Explícita.....	24
2.8 - Método Heurístico - “Meta-heurísticas”	26
2.9 - “Meta-heurística” Escolhida - Busca Tabu	27
2.10 - <i>Local Branching</i>	35
2.11 - Caracterização do Problema Objeto deste Estudo	37
2.12 - Modelagens Matemáticas em Problemas em Ambiente <i>Flow shop</i>	41

CAPÍTULO 3 - DESCRIÇÃO E MODELAGEM DO PROBLEMA

3.1 - Descrição do Problema.....	46
3.2 - Formulação Matemática do Problema Geral.....	47
3.3 - Procedimentos Alternativos para Resolução do Problema.....	52

CAPÍTULO 4 - HEURÍSTICAS ALTERNATIVAS PROPOSTAS

4.1 - Introdução.....	55
4.2 - Heurística Propostas.....	57
4.3 - Busca Tabu.....	59
4.4 - Busca Exaustiva.....	68
4.5 - Heurística Construtiva.....	72

CAPÍTULO 5 – RESULTADOS

5.1 - Introdução.....	78
5.2 - Resultados de Dois Problemas Exemplos com 10 e 30 Tarefas.....	80
5.2.1 - Resultados do pacote CPLEX – E/TCPLEX.....	82
5.2.2 - Resultados utilizando a “meta-heurística” Tabu Search – E/TTabu.....	86
5.2.3 - Resultados utilizando a heurística de busca exaustiva – E/TBE.....	90
5.2.4 - Resultados utilizando a heurística construtiva – E/THC.....	92
5.2.5 - Comparação – E/TCPLEX, E/TTabu, E/TBE e E/THC - problemas com 10 e 30 tarefas.....	93
5.3 - Resultados do Algoritmo e Heurísticas - Problemas Diversos.....	95
5.3.1 - Problemas do tipo máquina única e data comum de entrega – E/TCPLEX E/TTabu e E/TBE e Heurística de Feldmann e Biskup.....	95

5.3.2 - Problemas com 10 tarefas e datas independentes – E/TCPLEX, E/TTabu e E/THC.....	98
5.3.3 - Problemas Diversos - E/TCPLEX, E/TTabu e E/THC.....	101
5.4 - Propostas de Melhoria.....	106
5.4.1 - Interação do E/TCPLEX com o E/TTabu e E/THC.....	106
5.4.2 - Heurística <i>Local Branching</i>	108
5.5 - Disposições Finais do Capítulo.....	110
 CAPÍTULO 6 - CONCLUSÕES E RECOMENDAÇÕES	
6.1 - Considerações Sobre o Tema.....	114
6.2 - Conclusões.....	116
6.3 – Sugestões.....	118
BIBLIOGRAFIA.....	120
 APÊNDICE A - Códigos computacionais do algoritmo e heurísticas.....	Em CD
APÊNDICE B - Dados de entrada do algoritmo e heurísticas.....	Em CD

LISTA DE FIGURAS

FIGURA 2.1 – Fluxo de processamento em um <i>flow shop</i> generalizado.....	14
FIGURA 2.2 - Matriz de transferência de um <i>flow shop</i> com 5 máquinas extraído de Hax (1984).....	14
FIGURA 2.3 – Pseudocódigo de um algoritmo <i>Branch and Cut</i> - Baseado em Pochet and Wolsey (2006).....	23
FIGURA 2.4 - Exemplo de árvore com grau 3.....	25
FIGURA 2.5 - Quatro dimensões da Busca Tabu.....	30
FIGURA 2.6 - Diagrama de blocos do Busca Tabu.....	34
FIGURA 3.1 - Exemplo de um ciclo.....	52
FIGURA 3.2 - Comportamento das punições por atraso e adiantamento em relação à data de entrega para cada tarefa.....	54
FIGURA 4.1 – Árvore de grau 3 com 4 níveis, ramificando todas as sequências possíveis para o estudo do exemplo de três tarefas.....	68
FIGURA 5.1 – Valor da função objetivo & Tempo – Problema com 10 tarefas e 3 Máquinas – E/TCPLEX.....	83
FIGURA 5.2 – Valor da função objetivo & Tempo - Problema com 30 tarefas e 5 máquinas - E/TCPLEX.....	85
FIGURA 5.3 - Valor da função objetivo & número de iterações – Problema com 10 Tarefas – E/TTabu.....	89

FIGURA 5.4 - Valor da função objetivo & número de iterações - Problema com 30 e 5 máquinas tarefas – E/TTabu.....	90
FIGURA 5.5 - Gráfico de ocupação das máquinas - E/TCPLEX, E/TTabu E/THC e E/TBE..	94

LISTA DE TABELAS

Tabela 2.1 - Resumo de alguns trabalhos visto na literatura.....	39
Tabela 4.1 - Parâmetros $a(i)$, $b(i)$, $d(i)$ e $f(i)$	63
Tabela 4.2 - Roteiro de cada tarefa i	63
Tabela 4.3 - Tempo de processamento de cada tarefa i em cada máquina m - $p_{i,m}$	63
Tabela 4.4 - Matriz de reincidência.....	67
Tabela 5.1 - Tempos de processamento do E/TCPLEX.....	79
Tabela 5.2 - Valores de $p(i,j)$, $a(i)$, $b(i)$, $d(i)$, $f(i)$ e roteiro de cada tarefa i - Problema com 10 tarefas, atendidas por 3 máquinas.....	80
Tabela 5.3 - Valores de $p[i,j]$, $a[i]$, $b[i]$, $d[i]$, $f[i]$ e roteiro de cada tarefa i - Problema com 30 tarefas e atendidas por 5 máquinas.....	81
Tabela 5.4 - Resultado do algoritmo E/TCPLEX – Problema com 10 tarefas e 3 máquinas.....	82
Tabela 5.5 - Tabela de ocupação das máquinas - E/TCPLEX - Problema com 10 tarefas e 3 máquinas.....	83
Tabela 5.6 - Resultado do algoritmo E/TCPLEX – Problema com 30 tarefas e 5 máquinas.....	84

Tabela 5.7 - Comportamento da função objetivo em função dos valores dos parâmetros: número de elementos na lista Tabu e número de iterações sem melhoria (porcentagem do número total de iterações) – problema com 30 tarefas e 5 máquinas.....	87
Tabela 5.8 – Resultado da heurística E/TTabu – Problema com 10 tarefas e 3 máquinas.....	88
Tabela 5.9 - Tabela de ocupação das máquinas - E/TTabu - Problema com 10 tarefas e 3 máquinas.....	88
Tabela 5.10 - Resultados da heurística E/TTabu – problema com 30 tarefas.....	89
Tabela 5.11 - Resultado da heurística E/TBE – Problema com 10 tarefas e 3 máquinas	91
Tabela 5.12 - Tabela de ocupação das máquinas – E/TBE – Problema com 10 tarefas e 3 máquinas.....	91
Tabela 5.13 - Resultado da heurística E/THC – Problema com 10 tarefas e 3 máquinas.....	92
Tabela 5.14 - Tabela de ocupação das máquinas – E/THC – Problema com 10 tarefas e 3 máquinas.....	92
Tabela 5.15 - Resultados da heurística E/THC – Problema com 30 tarefas e 3 máquinas.....	93
Tabela 5.16 - Comparação dos resultados dos dois problemas.....	94

Tabela 5.17 - Comparação do algoritmo E/TCPLEX, das heurística E/TTabu e E/TBE e com uma heurística específica proposta por Feldmann e Biskup(2003).....	97
Tabela 5.18 - Comparação dos algoritmo E/TCPLEX e das heurísticas E/TTabu, E/THC e E/TBE para problemas com 10 tarefas com datas de entrega não comuns.....	98
Tabela 5.19 - Resultados do E/TCPLEX para o problema 3 da tabela 5.18.....	100
Tabela 5.20 - Comparação dos algoritmo E/TCPLEX e das heurística E/TTabu e E/THC para problemas com 20, 50 e 100 tarefas.....	102
Tabela 5.21 - Resultados do E/TCPLEX para o problema 25 da tabela 5.20.....	103
Tabela 5.22 - Resultados do E/TCPLEX para o problema 20 da tabela 5.20.....	104
Tabela 5.23 - Comparação E/TTabu e E/TCPLEX com tempo de processamento maior.....	105
Tabela 5.24 - Resultados do E/TCPLEX a partir de solução inicial – (Problemas da tabela 5.20).....	107
Tabela 5.25 - Resultados da heurística <i>Local Branching</i> – (problemas da tabela 5.20)..	108
Tabela 5.26 – Comparação - LBTabu, LBHC, CPTabu e CPHC.....	109
Tabela 5.27 - Comparação das heurísticas E/TTabu e E/TBE para problemas com 10 tarefas, admitindo e não admitindo prioridade no atendimento das tarefas.....	112

CAPÍTULO 1

INTRODUÇÃO

O objetivo do material deste capítulo é o de fornecer uma introdução conceitual sucinta e apresentar os objetivos, abrangência e relevância do trabalho.

A proposta deste trabalho é, por meio da atividade de programação, melhorar a eficiência de um sistema produtivo. Tal sistema produtivo consiste em processar um conjunto de tarefas independentes em um conjunto de máquinas também independentes, considerando um ambiente *flow shop*, onde a tarefa tem a possibilidade de pular máquinas, com cada tarefa tendo uma janela de tempo como período de entrega, sendo que, fora desse período, a entrega implicará punição por atraso ou adiantamento. A melhoria desse sistema produtivo será em termos de custo, portanto a melhoria será minimizar a soma das punições por atraso e adiantamento.

A seguir, para melhor situar o problema escolhido para esta tese, será feita uma abordagem sucinta de sistema produtivo, programação e de problemas de adiantamento e atraso.

1.1 – Sistema de Produção

A eficiência de um sistema de produção vai depender de uma série de decisões tomadas, dentre elas as de planejamento e programação da produção. Segundo Santoro (1982), devido à amplitude e diversidade das características das empresas onde devem ser aplicadas essas atividades, torna-se imprescindível uma partição de sua área de aplicação, de modo a se poder utilizar, de forma prática, os conceitos e modelos conhecidos. Ainda segundo Santoro

(1982), se forem examinados estes conceitos e modelos, nota-se que podem ser claramente associados ao tipo de produção onde se aplicam, fato que favorece uma partição da área por tipos de produção, que podem ser:

- Estoque puro;
- Produção contínua pura;
- Produção contínua com diferenciação;
- Produção intermitente repetitiva;
- Produção intermitente sob encomenda; e
- Produção para grandes projetos sem repetição.

Os tipos de produção variam num contínuo (Santoro, 1982) e podem aparecer juntos numa mesma empresa. Essa continuidade dificulta a distinção dos limites entre um tipo de produção e outro, mas não invalida a caracterização geral dos mesmos, nem sua importância para a área de planejamento e programação.

Para o presente trabalho os tipos de produção que interessam são os de produção intermitente, repetitiva e sob encomenda que se caracterizam, normalmente, por um arranjo funcional de seus recursos produtivos, homens e máquinas, ou seja, os recursos com funções semelhantes no processamento de produtos e/ou serviços encontram-se fisicamente agrupados e esses agrupamentos costumam estar posicionados coerentemente com o fluxo principal de processamento. Para estes tipos de produção, o modelo de planejamento é o planejamento agregado e o modelo de programação é a “sequenciação” (a palavra sequenciação não foi encontrada formalmente na língua portuguesa, por esta razão ela e outras palavras não encontradas estarão entre aspas).

1.2 - Programação (*scheduling*)

Primeiramente será dada uma repassada na nomenclatura dos principais termos para programação intermitente, utilizados no presente trabalho, que seguirá basicamente as mesmas

definições de Santoro (1982), que é bem utilizada nos trabalhos técnicos, com exceção de sequência de prioridade.

- Tarefa – que também pode ser designada por ordem, produto, trabalho (*job*), corrida, entre outros. Pode ser um produto ou um lote de produtos idênticos, que devem ser processados pelos recursos produtivos.
- Máquina – que também pode ser posto, processador, equipamento entre outros, é o recurso produtivo destinado a executar uma operação. Outro recurso produtivo é o recurso humano que pode ser designado por homem.
- Operação ou processamento – é o trabalho realizado pelo recurso produtivo sobre a tarefa.
- Tempo de processamento – é o tempo necessário para se concluir uma operação sobre uma tarefa. Neste trabalho, o tempo de processamento incluirá os tempos de preparação, transporte, colocação e retiradas dos recursos produtivos.
- Roteiro – neste trabalho, roteiro será a sequência ordenada de recursos produtivos pelos quais devem passar as tarefas. Cada tarefa terá um roteiro independente.
- Sequência de prioridade – é a disposição das tarefas por ordem de atendimento, ou seja, cada tarefa terá uma prioridade de atendimento que será ditada pela posição da tarefa na sequência. Deste modo, nas filas formadas nos recursos produtivos, serão atendidas primeiramente as tarefas com melhor prioridade de atendimento, ou seja, tarefas com posições à frente das outras tarefas já posicionadas na fila do recurso produtivo;
- Programa – é uma sequência viável, onde estão especificadas as datas de início e término de processamento de cada tarefa;
- Tempo ocioso – no presente trabalho, a referência a tempo ocioso estará associada ao intervalo de tempo em que uma máquina pode ficar parada quando, estando disponível e tendo uma tarefa para atender, ela espera para atender outra tarefa. Outros tempos em que as máquinas possam estar paradas, sem tarefa nenhuma na fila, não serão designados por tempo “ocioso”. Deste modo, inserção de tempo ocioso será fazer a máquina ficar parada mesmo que tendo uma tarefa pronta para ser atendida na fila.

Segundo BAKER (1997), *schedule* é um plano tangível, tal como um plano de horários de ônibus ou de aulas. Usualmente uma programação informa quando certas coisas devem acontecer; ela mostra o plano de tempos de certas atividades e responde a questão: -

quando alguma coisa terá lugar? A resposta para essa questão usualmente nos informa o horário (data), como por exemplo: uma viagem de ônibus começa às 8:00 hrs e termina às 11:30 hrs. Entretanto, uma outra resposta válida poderia ser em termos de sequência e não horário (data).

Tendo definido *schedule*, Baker, definiu também programação (*scheduling*) como sendo, implicitamente, o processo de geração de *schedules* e que a estrutura dos problemas de programação na indústria é a seguinte: um conjunto de tarefas a serem executadas por um conjunto de recursos disponíveis. Deste modo, dado um conjunto de tarefas e um conjunto de recursos, o problema de programação seria o de determinar os tempos detalhadamente de execução de cada tarefa dentro da capacidade de recursos disponíveis.

A finalidade de um sistema de programação da produção é a utilização eficiente de recursos limitados na fabricação de produtos de modo a satisfazer a demanda dos clientes e gerar lucros para a empresa. As restrições incluem a disponibilidade de recursos e a satisfação dos clientes. A satisfação do cliente consiste em fornecer uma variedade de produtos a baixo custo e entregá-lo sem atraso e com qualidade, (Oliveira, 2002).

A programação da produção intermitente envolve a alocação no tempo de múltiplas ordens ou tarefas a recursos produtivos, máquinas e/ou homens, ordens essas que podem sofrer números diferentes de operações, com diferentes roteiros e tempos.

Programar envolve a especificação, ou consideração implícita, das datas de ocorrência de eventos. Na programação, as datas principais consideradas são: as datas de início e término de cada operação.

A “sequenciação”, que é o modelo de programação para produção intermitente, envolve a ordenação ou prioridade segundo a qual ordens ou tarefas deverão ser processadas em um conjunto de máquinas. A sequência de cada centro é aquela das operações de todas as ordens, cujo roteiro passa pelo centro. Chama-se sequência de um problema ao conjunto de sequências de todos os centros produtivos.

A classificação dos problemas de programação deve considerar fatores internos, que a influenciam por meio das requisições tecnológicas determinadas principalmente pelo fluxo padrão das tarefas nas máquinas, pelo número e tipos de máquinas disponíveis e pela precedência das tarefas, (Nagano, 1998).

O fluxo padrão consiste no direcionamento das tarefas com relação às máquinas, de tal forma que sempre utilize a mesma máquina para processar determinada tarefa numa data fixa.

O número de tipos de máquinas e a quantidade de máquinas de cada tipo disponíveis para desempenhar as tarefas são importantes para a classificação da programação, pois materializam o tamanho da alocação dos recursos no tempo, permitindo a utilização específica dos métodos de otimização.

A determinação da sequência das tarefas, convenientemente, depende primordialmente das etapas requeridas para o processamento de cada uma e dos custos associados à produção.

Para Nagano (1998) os principais problemas de programação encontrados na literatura são: *Job Shop*; *Open Job Shop*; *Batch Shop*; *Flow Shop*; *Batch/Flow Shop*; *Manufacturing Cells* (células de manufatura); *Assembly Shop* (estação de montagem); *Assembly Line* (linha de montagem); *Transfer Line* (linha de transferência); *Flexible Transfer Line* (linha de transferência Flexível). Para este trabalho será considerado o problema do tipo *flow shop*, que será definido mais adiante.

1.3 – Problemas de Adiantamento e Atrasos (earliness/tardiness)

Até agora foi visto em que classe de problemas o problema objeto deste trabalho está inserido. Agora será colocado o objetivo a ser atendido na resolução deste problema, que é minimizar a soma das punições por adiantamentos e atrasos. A seguir serão vistos alguns conceitos relacionados a esta forma de abordagem.

Segundo Baker e Scuder (1989), por muitos anos a pesquisa para resolução de problemas de programação foi focada no estudo de uma única medida de desempenho, por exemplo, o atraso. Porém, com o grande interesse em *Just-in-time* na produção para se economizar em custos e satisfazer clientes, a maneira de focar o problema teve mudanças. Com o *Just-in-time*, foram destacados os atrasos, bem como os adiantamentos. Com isso, tarefas que fossem terminar antes da data prometida para a entrega, ou seja, com adiantamento (*earliness*), deveriam esperar para que sua entrega não cause problemas de armazenagem, enquanto que tarefas que fossem terminar seu processamento depois da data de entrega, ou seja, com atraso (*tardiness*), deveriam ser iniciadas em instantes adequados, para que causem os menores custos

devido a punições por atraso. A melhor programação, ou a melhor sequência de tarefas é aquela em que todas as tarefas acabam seu processamento em suas respectivas datas de entrega, o que não acontece em muitos problemas. A grande dificuldade para se resolver problemas deste tipo é que o número de possibilidades de solução aumenta muito rapidamente com o aumento do número de tarefas, podendo inviabilizar a escolha da melhor alternativa em termos de tempo computacional; basta lembrar que o número de sequências possíveis para um problema com uma única máquina é $(N!)$, sendo N o número de tarefas. Problemas com esta característica são conhecidos na literatura como NP-Difíceis (Não Polinomiais Difíceis). Assim, com estas considerações, se justifica investir em heurísticas, que podem não atingir uma sequência ótima, mas que encontrem soluções boas o suficiente, com tempos de processamento em computador aceitáveis.

1.4 - Problema Proposto

Como será mais bem detalhado adiante, em um problema de programação de tarefas, em um ambiente *flow shop*, cada uma de um conjunto de N tarefas deve ser processada por um subconjunto de M máquinas distintas. Cada tarefa tem um roteiro distinto através das máquinas; assim sendo, as tarefas podem pular máquinas.

Trabalhos na área de programação de tarefas mostram que atender a data de entrega dos clientes é uma preocupação crítica na maioria dos sistemas de manufatura. Um considerável número destes trabalhos se preocupa em minimizar as seguintes medidas de desempenho: *Makespan* (duração total da programação); *Mean flowtime* (tempo médio de fluxo); *Total flowtime* (tempo total de fluxo); *Mean tardiness* (atraso médio na conclusão das tarefas); *Maximum tardiness* (atraso máximo na conclusão das tarefas). Poucos se preocupam com problemas do tipo que punem atrasos e adiantamentos, que é uma consideração comum no mundo real e é o objetivo deste trabalho.

As tarefas serão punidas por atrasos e adiantamentos, não sendo punidas as tarefas que forem entregues em sua janela de tempo e a medida de desempenho a ser minimizada será a soma destas punições.

Atingir o objetivo de melhorar a eficiência de um sistema produtivo a partir da minimização da soma de punições, mediante a busca de sequências de tarefas a serem atendidas, é bem difícil, já que se trata de um problema de natureza combinatória e considerado NP-Difícil como já foi dito. Segundo Morton e Pentico(1993), as possibilidades de sequências possíveis para busca, com troca de ordem, são $(N!)^M$, sem considerar a inserção de tempos ociosos, pois, se for considerado tal inserção, o problema fica mais complicado. Assim, neste trabalho foram feitas algumas simplificações que serão vistas mais adiante.

Para atingir o objetivo proposto, no presente trabalho, foi proposta uma formulação matemática exata na forma de programação inteira mista, para ser resolvida por meio de um método exato. Como esta estratégia não é eficiente, em termos de tempo de processamento, como será visto mais adiante no presente trabalho, foram também desenvolvidas três heurísticas que buscam a melhor sequência de tarefas que minimizem a soma de punições, levando em conta certas simplificações em relação à formulação exata. Além de definir a sequência de tarefas, estas três heurísticas também determinarão a data de liberação das tarefas para processamento, que será a mesma para todas as tarefas.

1.5 - Objetivo e Relevância do Estudo

A globalização dos negócios intensificada nas últimas décadas acirrou a concorrência mundial e tem levado as empresas a fazerem negócios internacionalmente, ou seja, a produzir e distribuir componentes e produtos acabados em várias partes do mundo. Os clientes exigem produtos de qualidade, entrega sem atraso e a um custo baixo, mesmo se os produtos venham de outro continente. Eles estão frequentemente procurando outros fornecedores, se os produtos não são entregues nas datas de entrega acordadas. Diante disso, o planejamento, a programação e o controle da produção têm se tornado mais vital para o sucesso competitivo das empresas, (Oliveira, 2002).

A atividade de programação pode ser considerada estratégica porque permeia diferentes atividades como suprimentos, estoques, produção, distribuição e porque o atendimento a clientes implica atendimento de datas, que é o objeto de estudo e objetivo da programação.

Mesmo quem não aceita que seja uma atividade estratégica, aceita que seu desempenho pode afetar as estratégias das empresas, a ponto de invalidá-las, (Santoro, 1982).

Assim, este trabalho pretende explorar, sob o aspecto teórico e prático, o problema de programação de produção intermitente, mais especificamente o problema de programação em ambiente *flow shop*, apresentando uma abordagem alternativa para sua solução. O problema de *flow shop*, abordado neste trabalho, se diferencia dos encontrados na literatura por considerar conjuntamente particularidades como: janelas de tempo, possibilidade das tarefas pularem máquinas no processamento e como medida de desempenho a minimização da soma dos custos advindos das punições por adiantamentos e atrasos, sendo uma medida de desempenho não regular.

O objetivo principal desta tese foi propor métodos para se resolver o problema. Primeiramente, foi desenvolvida uma formulação matemática do problema e, assim, buscar por meio de um método exato soluções para o problema e verificar a limitação deste procedimento à medida que cresce o tamanho da instância do problema. Dada a dificuldade computacional, em termos de tempo de processamento, de se obter a solução ótima a partir do método exato, foram desenvolvidos métodos heurísticos admitindo simplificações na exploração do espaço de soluções.

Finalmente, foram propostas duas alternativas de busca, uma primeira que admite a solução obtida a partir do método heurístico como solução inicial no método exato e uma segunda que admite a solução obtida a partir do método heurístico como solução inicial em uma heurística consagrada na literatura.

Este tipo de problema foi escolhido por que do ponto de vista prático ele se aproxima de problemas reais e do ponto de vista acadêmico considera características tais como: janelas de tempo, possibilidade das tarefas pularem máquinas e uma medida de desempenho pouco abordada na literatura.

Finalizando, problemas de programação são bem estudados na literatura técnica, mas problemas com a medida de desempenho aqui considerada são casos menos estudados; o problema abordado neste trabalho, que é um problema de difícil resolução, considera características como janelas de tempo que são mais flexíveis que uma data fixa e a possibilidade das tarefas pularem tarefas que deixa o problema mais realista. Além de tudo isto, as estratégias de resolução utilizadas são abrangentes.

1.6 – Delineamento do Trabalho

Como já foi visto, o problema objeto deste estudo foi escolhido por ser um problema de programação pouco estudado na literatura e, com as características adotadas, se torna um problema inédito. No capítulo 2 será apresentada a fundamentação teórica, mostrando a abordagem geralmente utilizada nos problemas de programação, os métodos de resolução mais usados e a possibilidade de aplicação desses métodos no presente trabalho. No capítulo 3 será detalhado o problema objeto deste trabalho, bem como sua modelagem matemática, definindo-se os parâmetros utilizados, variáveis, estratégia de resolução e, também, procedimentos alternativos para sua resolução. No capítulo 4 são apresentados, de forma detalhada, os procedimentos alternativos de resolução, bem como as heurísticas desenvolvidas com as simplificações e parâmetros adotados. No capítulo 5 serão apresentados os resultados dos experimentos computacionais e sua análise. No capítulo 6, serão apresentadas as conclusões e sugestões para próximos trabalhos.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

Neste capítulo, é apresentado formalmente o problema específico de programação da produção objeto deste trabalho; são examinados problemas correlatos encontrados na literatura, será feita uma pequena revisão das principais ferramentas computacionais utilizadas para problemas de programação; e serão tecidos comentários sobre simplificações e estratégias de resolução encontradas na literatura.

2.1 – Introdução

O sistema de produção está inserido em um sistema mais amplo, que pode ser chamado sistema empresa. Neste sistema, os principais grupos que influenciam são: acionistas ou proprietários, administradores, empregados, consumidores, sociedade e governo. Para se garantir o atendimento dos objetivos do sistema de produção, o sistema empresa utiliza atividades de planejamento e programação. A diferenciação destas atividades está no nível hierárquico que elas ocupam. Convenciona-se chamar de programação da produção a atividade de planejamento de produção em nível hierárquico mais baixo. Acima desse nível, encontra-se o planejamento da produção propriamente dito, ou planejamento agregado da produção e, acima deste, o planejamento estratégico.

No nível de programação, as decisões sobre quanto produzir dizem respeito a produtos individualizados. As decisões de quando produzir, dizem respeito, normalmente, a períodos muito menores do que do que o abrangido pelo ciclo produtivo, finalmente as decisões

sobre com que produzir podem ser detalhadas, a ponto de indicar em que recursos específicos devem ser efetuados as operações.

O interesse do presente trabalho está nos modelos que geram as decisões de programação da produção e afetam o comportamento do sistema produtivo.

Os critérios para medição da eficiência de um sistema de produção e a eficiência das decisões tomadas dependem da fixação de objetivos a serem atingidos, (Santoro, 1982).

A necessidade de tomar boas decisões, com respeito ao conjunto de objetivos, faz com que se seja obrigada a considerar prioridades ou pesos relativos aos objetivos individuais. Esses pesos podem variar de sistema de produção para outro. Para os problemas do tipo que punem os atrasos e adiantamentos, os objetivos principais de atendimento, segundo Santoro (1982), seriam:

1. Minimizar investimentos com produtos acabados, desta maneira os adiantamentos são punidos;
2. Minimizar atrasos e não atendimentos de ordens, punindo estes atrasos;
3. Maximizar os fluxos monetários, adotando uma solução que minimize a somatória das punições.

Este trabalho tem como objetivo melhorar a eficiência de um problema de programação da produção intermitente que se caracteriza, normalmente, por um arranjo funcional de seus recursos produtivos, homens e máquinas. O volume de produção por produto costuma ser pequeno e caracteriza lotes de produção com roteiros diferenciados pelos recursos e funções.

O presente capítulo, também apresenta uma pequena revisão dos métodos de resolução de problemas de programação, com intuito de apresentar alternativas para a resolução do problema proposto no capítulo 3, que trata de problemas muito difíceis de resolver e ainda pouco estudados. Como já foi visto, o problema que será estudado neste trabalho é um problema de natureza combinatória e por esta razão é bastante complexo e de difícil solução analítica. Assim, um método de resolução exato que encontra a solução ótima só é viável, em termos de tempo computacional, em problemas de pequeno porte; para problemas maiores, foram desenvolvidas alternativas mais viáveis em termos de tempos computacionais.

2.2 – O Problema de Programação Intermitente

Segundo Ashour (1972), o problema de programação de produção intermitente surge sempre que seja necessário programar um conjunto de ordens num determinado número de máquinas, obedecendo a roteiros especificados, tais que a sequência dessas ordens seja ótima com respeito à determinada medida de desempenho.

Para não confundir programação com “sequenciação”, a definição usada neste trabalho é a mesma de Santoro (1982), como vista a seguir.

A palavra programação envolve a especificação, ou consideração implícita, das datas de ocorrência de eventos, sendo os principais eventos os inícios e terminos de operações.

A palavra “sequenciação” envolve a ordenação segundo a qual ordens deverão ser processadas em máquinas. Não se consideram, neste caso, as datas em que essas ordens deverão ser processadas, mas sim suas posições relativas.

A palavra sequência representa uma ordenação total do conjunto de ordens em questão, por todas as máquinas.

Deste modo, dada uma sequência e definida uma data inicial, se for admitido que todas as ordens iniciassem seu processamento nas máquinas tão cedo quanto possível, pode-se gerar um programa.

Segundo Morton e Pentico (1993), programação ocorre em um vasto campo das atividades econômicas e sempre envolve execução de certo número de itens atados a uma limitação de recursos por período de tempo. Morton e Pentico definem que estas coisas que deverão ser executadas podem ser chamadas de tarefas ou trabalhos e são compostas por partes elementares chamadas de atividades ou operações. Cada atividade necessita de certo montante de recursos específicos por certo tempo específico chamado de tempo de processo. Os recursos também têm partes elementares que podem ser designadas como máquinas, células, etc.

Os problemas de programação são frequentemente complicados por um grande número de restrições, que dependem de cada atividade e da relação entre essas atividades, da relação entre atividades e recursos e por fatores externos ao sistema. Por exemplo: podem existir restrições de precedência entre tarefas, assim não é possível começar a execução de uma antes de outra determinada tarefa; um recurso não pode ser utilizado simultaneamente por duas tarefas;

algum recurso não pode ser utilizado durante certo período. Como estas complicações podem trazer muitas dificuldades à resolução de um problema de programação, é comum se resolver antes versões mais simples deste problema.

Um sistema de programação toma decisões dinamicamente entre disputa de atividades e recursos em ordem a fim de finalizar o atendimento em um centro produtivo, satisfazendo as disponibilidades de recursos bem como a minimização dos custos de operação.

Os tipos básicos de programação incluem:

- “Sequenciação”;
- Distribuição/despacho;
- “Roteirização”.

O tipo de programação que será considerado neste trabalho será a “sequenciação”; assim encontrar a melhor programação significa encontrar a melhor sequência de tarefas a serem processadas.

2.3 - Flow Shops

Dentro dos problemas de programação de produção intermitente, o tipo que será objeto de estudo é o *flow shop*. Em um *flow shop*, segundo Baker (1997), cada tarefa tem sua própria sequência de processamento com fluxo linear unidirecional, ou seja, sem retorno no fluxo. O fluxo pode ser discreto, contínuo ou “semicontínuo”, cada tarefa só pode ser atendida por uma máquina de cada vez e cada máquina só pode atender uma tarefa por vez, não podendo a tarefa retornar para atendimento em máquinas anteriores. No caso mais simples, cada tarefa exige o mesmo conjunto de atividades, na mesma sequência e no mesmo conjunto de máquinas. Assim, no ambiente *flow shop*, n tarefas devem ser programadas para processamento em um conjunto de m máquinas distintas, tendo o mesmo fluxo de processamento, o fluxo unidirecional. Quando, em todas as máquinas, a ordem de processamento das tarefas é a mesma, tem-se o *flow shop permutacional*.

A figura 2.1 mostra o esquema genérico de um *flow shop*, conforme Baker (1997), onde as tarefas podem entrar por qualquer máquina, pular máquinas, mas sempre seguindo um fluxo unidirecional.

Figura 2.1 – Fluxo de processamento em um *flow shop* generalizado.

Outra forma de descrever um *flow shop* é por meio de uma matriz de transferência, encontrada no trabalho de Hax e Candea (1984), em que só pode haver entrada na linha i e coluna j , se $j \geq i$. Estas entradas mostram a fração de processamento transferida da máquina i para máquina j como é mostrado na figura 2.2, tirada de Hax e Candea (1984). A primeira linha mostra que a tarefa pode entrar na estação de processamento pelas máquinas 1, 2 e 4. A última coluna indica que as tarefas podem sair da estação de processamento pelas máquinas 2, 3, 4 e 5.

		PARA					
		Maq1	Maq2	Maq3	Maq4	Maq5	Saída
DE	Entrada	0.7	0.2		0.1		
	Maq1		0.5	0.3	0.1	0.1	
	Maq2			0.7	0.2		0.1
	Maq3				0.6	0.2	0.2
	Maq4					0.6	0.4
	Maq5						1.0

Figura 2.2 – Matriz de transferência de um *flow shop* com 5 máquinas extraído de Hax e Candea (1984).

Existem algumas variações simples do *flow shop* que são conhecidas, segundo Morton e Pentico (1993), como: *skip shops*, *reentrant flow shop*, *compound flow shop*, *finit line flow shop*.

Em um *skip shop*, as tarefas podem pular algumas máquinas; no *reentrant flow shop*, as tarefas podem ser processadas mais de uma vez em uma máquina. No *compound flow shop*, cada máquina na série pode ser substituída por um grupo de máquinas e normalmente estes grupos seriam máquinas paralelas, que é um tipo de ambiente diferente do *flow shop*, ou uma linha de batelada (*batch line*) seguida por máquinas paralelas. Em um *finite line flow shop*, ou seja, em um *flow shop* com fila finita, existe limite superior de estoque para as máquinas que vêm a seguir da primeira.

Neste trabalho, a única variação que será seguida, dentre as vistas acima, é a possibilidade de as tarefas pularem máquinas na linha de processamento.

Quando se busca uma sequência ótima para um *flow shop*, a questão que vem é, se é suficiente determinar a permutação (troca das tarefas na sequência) em que as tarefas percorreram o sistema inteiro, ou talvez seja necessário, por exemplo, considerar inserção de tempo ocioso. Fisicamente, pode ser possível para uma tarefa passar outra enquanto na fila por uma máquina que está ocupada. As máquinas podem não operar de acordo com princípio FIFO (Primeiro a chegar, primeiro a ser atendido) e a sequência em que as tarefas vão atravessando as máquinas pode mudar de uma máquina para outra. Mudando a sequência das tarefas que esperam na fila entre duas máquinas, pode haver diferenças nos resultados da função objetivo.

Para Pinedo (1995), existe uma sequência ótima para certos problemas com duas e três máquinas que não requerem mudanças entre duas máquinas. Pode-se encontrar *flow shops* com quatro máquinas em que a sequência ótima requer mudança na ordem das tarefas, entre a segunda e a terceira máquinas.

Encontrar sequências ótimas, quando a sequência em cada máquina não é a mesma, é bem mais difícil do que quando é a mesma. Segundo Morton e Pentico (1993), as possibilidades de sequências com troca de ordem são $(N!)^M$ e sem troca de ordem são $(N!)$, em que N é o número de tarefas e M o número de máquinas.

Na literatura existem varias medidas de desempenho e, dentre elas, as mais utilizadas são:

- *Makespan*: duração total da programação;
- *Mean flow-time*: tempo médio de fluxo, ou tempo médio de duração de programação das tarefas;
- *Total flow-time*: tempo total de fluxo ou tempo total de duração de programação das tarefas;
- *Mean tardiness*: atraso médio na execução das tarefas;
- *Maximum tardiness*: atraso máximo na conclusão das tarefas;
- A soma das punições devido a atrasos e adiantamentos.

O problema de programação em *flow shop* tem sido tema de pesquisa durante as últimas cinco décadas, segundo Rajendran (1999). Algoritmos de otimização para casos particulares com duas e três máquinas em ambiente *flow shop*, com o objetivo de minimizar o *makespan* e o total *flowtime*, foram desenvolvidos por autores como Johnson (1954), Ignall e Schrage (1965).

Para Rajendran (1999), problemas em ambiente *flow shop* geralmente supõem que todas as tarefas (*jobs*) têm igual importância, porém isto não acontece necessariamente, como é o caso do problema objeto deste trabalho, em que as tarefas têm punições para adiantamentos e atrasos independentes e janelas de tempo de entregas também independentes. Assim, para este trabalho, a medida de desempenho que vai interessar, será aquela que minimiza a soma das punições devido a atrasos e adiantamentos.

Este tipo de problema é reconhecidamente um problema de natureza combinatória e, em termos de complexidade computacional, é classificado como NP-Difícil; assim sendo, ele só pode ser resolvido eficientemente de maneira ótima em casos de pequeno porte, como no caso do trabalho de Zhu e Heady (2000). Como o problema do tipo *flow shop* é de natureza combinatória, duas das propostas deste trabalho correspondem a resolvê-lo utilizando-se de métodos heurísticos. Estes métodos, por serem mais simples e utilizarem esforços computacionais menores, são muito utilizados para a solução desses tipos de problemas na literatura.

Outra questão importante, para se resolver problemas do tipo *flow shop*, é o estoque considerado entre máquinas, ou seja, quantas tarefas podem ficar esperando em filas. Na prática, o estoque entre máquinas adjacentes frequentemente é, segundo Pinedo (1995), virtualmente ilimitado. Este é o caso quando os produtos sendo processados são fisicamente pequenos (exemplo: circuitos integrados), assim é relativamente fácil estocar grandes quantidades entre máquinas.

Quando os produtos são fisicamente grandes (exemplo: televisores), os estoques entre máquinas são limitados. Se o estoque é limitado, então pode acontecer que a tarefa tenha que permanecer na máquina, não podendo seguir para a fila da próxima máquina, impedindo o início de outra, isto é conhecido como *Blocking* (impedimento) segundo Pinedo (1995).

Este trabalho supõe que o estoque é ilimitado; assim sendo, sempre que a máquina terminar o processamento de uma tarefa, ela estará disponível para uma próxima tarefa e não haverá impedimento.

2.4 - Problemas do Tipo Adiantamento/Atraso (*Earliness/Tardiness*)

Segundo Duarte (2001), nos últimos 30 anos, revelou-se especial atenção ao desenvolvimento de pesquisas na área de programação da produção. A proliferação desses trabalhos indica tanto a importância operacional dos problemas de programação para indústria quanto o atrativo matemático que tais problemas representam para as pesquisas científicas.

Os primeiros modelos de programação de tarefas, segundo Sakuraba (2005) e Henriques (2005), utilizavam, em sua maioria, medidas regulares de desempenho (não-decrescentes com o instante de termino das tarefas) como critério de decisão, como por exemplo, a minimização do *makespan* (instante de termino da ultima tarefa). Com o passar dos anos, critérios relacionados à data de entrega passaram a ser mais explorados, tendo em vista a crescente importância para o mercado de fatores ligados à determinação e ao cumprimento destas datas. Segundo Sakuraba (2005), problemas deste tipo foram estudados por diversos autores para o caso de uma máquina e, segundo Henriques (2005), um outro fator de destaque na literatura de

programação de tarefas é o fato da maioria dos artigos não tratar de problemas com limitação de estoques intermediários.

Na literatura, os trabalhos que tratam de problemas de *flow shop* que consideram atrasos e adiantamentos, em grande parte, consideram uma particularização do *flow shop*, conhecida como sequência única (*permutational scheduling*).

Assim, um dos objetivos deste trabalho também é fazer uma contribuição no estudo de problemas pouco explorado na literatura.

Os problemas com punições para adiantamentos e atrasos (adiantamento/atraso) são aqueles que consideram que tarefas que acabem antes de sua data de entrega podem gerar custos, tais como custos de armazenagem e as tarefas que acabam após sua data de entrega podem gerar outros custos, tais como custos subjetivos de insatisfação de clientes ou multas pelo atraso. Como já foi dito, a resolução destes tipos de problemas consistiria em encontrar uma sequência de atendimento que minimize estes custos.

Estes tipos de problema podem aparecer em vários ambientes de produção, tais como: *job shop*, *open shop*, máquinas paralelas e no ambiente de produção *flow shop* que será objeto deste trabalho. O ambiente mais simples seria aquele que considera apenas uma máquina e, segundo Pinedo (1995), estes modelos são muito importantes, pois, além de sua simplicidade, eles servem como caso especial para outros tipos de problemas que tenham fluxo unidirecional de processamento, sem retorno. Além disto, estes problemas têm propriedades que servem para modelos de múltiplas máquinas e seus resultados não só podem ser usados como resultados preliminares de análise, como também fornecem uma base para heurística de problemas mais complicados.

Os problemas encontrados na literatura técnica podem utilizar os mais variados métodos em sua resolução. Estes métodos podem buscar a solução ótima ou soluções aproximadas a partir de heurísticas. Os métodos que buscam a solução ótima frequentemente têm a desvantagem de só serem viáveis, em termos de tempo computacional, para problemas de porte pequeno. Apesar de não garantirem a obtenção de solução ótima, as heurísticas têm a vantagem de poder encontrar soluções boas em tempos computacionais viáveis. No desenvolvimento de heurísticas, o que se procura é, portanto, um equilíbrio adequado entre tempo computacional e a qualidade das soluções por elas geradas.

Pesquisadores têm desenvolvido heurísticas para as mais diferentes configurações de sistemas de programação, segundo Henriques (2005), desde *flow shops* com características “determinísticas” até job shops com limitação de área de armazenagem, custos ponderados e características estocásticas. As pesquisas contemplam também diferentes medidas de desempenho; o presente trabalho utilizará como medida de desempenho a soma dos custos devido a adiantamentos e atrasos e, como já foi visto, estes problemas são conhecidos como do tipo adiantamento/atraso e também contemplam as mais diferentes configurações. A seguir, serão vistos alguns destes problemas e, também, alguns dos métodos de resolução utilizados, de forma bem sucinta, já que não contemplam problema parecido com o tratado neste estudo.

2.5 – Métodos de Resolução de Problemas de Programação

Em um problema de otimização deseja-se maximizar ou minimizar (i.é, extremar) alguma função, denominada função objetivo e isto deve ser feito determinando os valores de certas variáveis de controle. Na maioria dos problemas são encontrados vínculos (restrições), (Converse, 1970).

Segundo Dantzig (1974), o processo de otimização matemática pode ser descrito aproximadamente como:

1. Desenvolver uma estrutura matemática que modela algumas “situações do mundo real”; geralmente a estrutura apresenta restrições no valor das variáveis e a função objetivo fornece uma medida de desempenho;
2. Investigar a existência de atributos que levem a solução ótima ou próxima da ótima, procurando maneiras de caracterizar os pontos ótimos;
3. Desenvolver ou utilizar algoritmos de busca da solução ótima ou próxima da ótima;
4. Implementar este algoritmo em situações particulares para avaliação e modificação.

A seguir, será vista uma classificação sucinta dos métodos computacionais, comumente encontrada na literatura para problemas de programação, lembrando que no caso

deste trabalho o problema de programação será do tipo “sequenciação”. Esta classificação foi extraída do trabalho de Pacheco e Santoro (1999).

Com esta classificação, será possível ressaltar as diferentes formas de abordagem destes métodos, podendo-se entender as semelhanças e diferenças entre os diversos métodos disponíveis, assim, tendo uma visão conjunta destes métodos.

Os métodos computacionais podem ser separados em dois grandes grupos: “otimizantes” e heurísticas. Dentro dos métodos “otimizantes”, há os processos de enumeração e, dentro dos métodos heurísticos, há os processos de busca e os construtivos. Os métodos de enumeração normalmente atingem a solução ótima do problema, porém com a desvantagem de um alto tempo computacional, dependendo do porte do problema, enquanto que os métodos heurísticos não garantem que se chegue na solução ótima, mas sim a soluções boas, dependendo do método, em tempos computacionais aceitáveis.

Métodos de Enumeração

Estes métodos podem ser classificados, quanto à forma de enumerar as alternativas, da seguinte maneira:

- **Enumeração explícita:** que seria a enumeração total de todas as alternativas. Este tipo é conhecido como busca exaustiva e de todos os métodos este é o menos sofisticado. Por isso mesmo é muito utilizado, pois é fácil de ser implementado e se aplica a maioria dos problemas de “sequenciação” sem exigir ajustes ou alterações. Normalmente, os métodos de enumeração explícita quando são utilizados o são para se encontrar a solução exata;
- **Enumeração inexistente:** isto se dá quando é possível se estabelecer regras bem definidas de se gerar as sequências, de tal forma que se permita a implementação de um algoritmo eficiente de otimização;
- **Enumeração implícita:** neste caso não é necessário se fazer uma enumeração total, pois existem métodos que garantem que se chegue à solução ótima. Casos bem conhecidos neste tipo de enumeração seriam: programação linear; programação dinâmica que é uma abordagem para se otimizar processos de decisão com vários estágios; e o método branch and bound (ramificação e poda).

Métodos Heurísticos

Estes métodos podem ser divididos em heurísticas construtivas e heurísticas de busca.

As heurísticas construtivas podem ser divididas em heurísticas orientadas a gargalo ou orientadas a sequência. Dentro das orientadas a sequências pode haver heurísticas com sequências impostas e heurísticas com as sequências construídas. As heurísticas que constroem as sequências podem ser de passo único, em que uma única sequência é construída a partir de regras determinadas, ou podem não ser de passo único construindo varias sequências a partir de variações das regras determinadas. É importante salientar que as heurísticas construtivas não são procedimento iterativo de melhora e que o sucesso dessas heurísticas vai depender das características do problema que se vai solucionar, ou seja, se é possível se determinar regras que possam aproximar bem a solução encontrada da solução ótima.

Nas heurísticas de busca pode-se citar a busca aleatória, busca em árvore e busca em vizinhança.

A busca aleatória é bem simples, escolhendo-se as sequências a serem estudadas de forma aleatória.

Na busca em árvore, se escolhem as sequências em um procedimento de ramificação em que se podem impor limites a esta ramificação e utilizar sistemas especialistas que usam conhecimentos anteriores.

A busca em vizinhança, que é um procedimento de busca de novas alternativas de solução que possam melhorar uma medida de desempenho, pode utilizar algoritmos desenvolvidos para problemas de programação, conhecidos como “meta-heurísticas”, dos quais pode-se citar o recozimento simulado (simulated annealing), o algoritmo genético e a busca tabu. Estas “meta-heurísticas” são muito utilizadas, com sucesso, de acordo com a literatura.

Como se trata de heurísticas, fica muito difícil dizer qual é a melhor, isto irá depender do tipo de problema e também das implementações dessas heurísticas. Porém as “meta-heurísticas” são muito utilizadas, como já foi dito e de fácil implementação.

2.6 – Método de Enumeração Implícita

Um dos métodos de enumeração que terá relevância para o presente estudo será o de enumeração implícita, já que este método pode ser usado em problemas de Programação Linear Inteira Mista (MIP – *mixed integer program*). Um problema de MIP é um problema de programação matemática envolvendo variáveis inteiras, variáveis reais, função objetivo e restrições lineares. Um dos métodos mais utilizados para a resolução de problemas de programação inteira mista é o *Branch and Bound* (Ramifica e Poda). O método *Branch and Bound* descreve uma estratégia geral de busca da solução ótima que, na prática, pode ser comprometida pelo número de problemas gerados que cresce exponencialmente com o número de variáveis inteiras. Não só o porte do problema é um fator determinante do número de nós da árvore. As variáveis de decisão e a estrutura das restrições também têm influência direta no desempenho dos métodos *Branch and Bound*, Mendes (2007).

De acordo com Mendes (2007), para se obter uma formulação mais restrita, de forma a melhorar os limitantes inferiores providos pelos problemas relaxados resolvidos em cada nó, usa-se uma estratégia de reformulação. Esta estratégia de reformulação visa introduzir na estrutura matemática do problema novas restrições lineares, chamadas de desigualdades válidas que não modificam o espaço de solução do problema.

Ainda segundo Mendes (2007), os pacotes de otimização matemática disponíveis no mercado trabalham com algumas famílias de desigualdades. Ao invés de serem específicas para um problema, estas desigualdades são aplicáveis à maioria dos problemas de programação inteira, pois abordam aspectos gerais como, por exemplo, a proposição de cortes para eliminação de soluções não-inteiras.

Uma extensão do método *Branch and Bound* é o método conhecido como *Branch and Cut*, o qual tem um procedimento próprio de inclusão de desigualdades válidas. O *Branch and Cut* se caracteriza quando uma solução, obtida pela resolução do problema de programação linear, é submetida à introdução de restrições que limitem o espaço de solução, permitindo que limitantes inferiores de melhor qualidade sejam obtidos, ao mesmo tempo em que o espaço de soluções é reduzido. Na figura 2.3 é apresentado um pseudocódigo, baseado em Pochet and Wolsey (2006), de um algoritmo *Branch and Cut*.

Algoritmo Branch and Cut (simplificado)

1. Inicialização

Carrega a lista de formulações de soluções iniciais relaxadas;
Carrega a solução inicial com um valor muito alto para o problema;
Assumi limite inicial um valor alto negativo;

2. Término

Se a lista de formulação estiver vazia **então**
 Se solução calculada continuar com valor inicial **então** problema inviável;
 Se solução calculada for menor que a inicial, **então** solução pertence ao conjunto de
 soluções viáveis e é ótima;

Fim

3. Seleção de um nó e solução do problema

Seleciona uma sub-lista de formulações (pertencente à lista completa);
3a - Achar uma solução ótima desta sub-lista;
3b - Resolver o problema de separação (introduzindo as restrições);
 se todas as desigualdades são satisfeitas **então** a solução é ótima => ir para o passo 4.
 se alguma desigualdade foi violada **então** adicionar um valor ao nó da solução e calcula
 uma nova solução relaxada (assim, teremos
 um nó com uma solução ótima relaxada) e
 volta ao passo 3b.;

4. Eliminação

Se a solução ótima da sub-lista é maior que a solução anterior da lista
 então sub-lista é inviável ou dominada pela melhor solução inteira
 encontrada;
Se a solução ótima da sub-lista é menor
 então **Se** nó pertence ao conjunto domínio
 então (a solução é integral) atualizar o limitante
 superior e atualizar a lista de formulações de
 soluções removendo os problemas dominados
 => ir para o passo 2.;

5. Ramificação (só quando a sub-lista não for eliminada)

Selecione nó que não pertence ao conjunto de nós visitados;
Atualiza a lista de formulações;
=> Ir para passo 2.

Figura 2.3 – Pseudocódigo de um algoritmo *Branch and Cut* - Baseado em Pochet and Wolsey (2006)

2.7 – Método de Enumeração Explícita

Outro método de enumeração que será utilizado no presente trabalho é o método de enumeração explícita, por sua facilidade de implementação.

Segundo Swait Jr (1991), os procedimentos para se gerar sequências a serem estudadas podem variar de acordo com as características dos problemas estudados. No caso deste trabalho, o procedimento será criar as sequências possíveis por meio de uma ramificação, como se fosse uma árvore criando ramos. Assim, este procedimento será chamado de árvore e suas definições serão vistas a seguir.

Uma árvore é um conjunto finito de átomos (nós) tal que um átomo é denominado de raiz e os átomos restantes são subdivididos em subconjuntos mutuamente exclusivos e coletivamente exaustivos, que também são árvores. Esta definição é recursiva, já que uma árvore é composta de outras árvores, que são compostas de outras árvores....

A figura 2.4 mostra um exemplo de árvore que lembra uma árvore genealógica. Neste exemplo de árvore, é definido que o nó A é a raiz dos ramos B, C, e D e no subconjunto B, E e F o nó B será a raiz e os nós E e F são os ramos, bem como os nós C e D são raízes dos ramos G e H, I e J respectivamente. Assim, pode ser dito que os ramos de um átomo são raízes das “sub-árvores” ligadas a eles.

O grau de um nó é dado pelo número de “sub-árvores” a ele ligadas. Assim, o nó A tem grau 3, o nó B tem grau 2, o nó C tem grau 1, o nó D tem grau 3...e o nó K tem grau zero.

Da mesma forma que um átomo específico tem grau, fala-se também do grau de uma árvore, definido como o máximo, entre todos os nós, do grau de cada nó. Portanto, a árvore da figura 2.4 tem grau 3.

Figura 2.4 – Exemplo de árvore com grau 3

A árvore pode ser definida como uma estrutura de duas dimensões. De certa maneira, o conceito de grau da árvore já definido expressa uma medida de sua largura. Sua altura é dada pelo nível mais profundo entre todos os seus nós. O nível de um nó é o número de nós no caminho dele até a raiz, mais um. Portanto, o nó A tem nível 1 (dado por $0 + 1$), o nó H tem nível 3 (dado por $2 + 1$) etc. A árvore do exemplo tem altura 4.

Outra definição que pode ser feita é de que os ramos E e F são ramos irmãos, pois estão ligados a mesma raiz, bem como os ramos H, I e J.

Para se determinar todas as sequências possíveis, o procedimento seria então, a partir da raiz, passar por todas as ramificações possíveis. No caso do exemplo visto acima, teríamos as seguintes ramificações: (A, B, E); (A, B, F); (A, C, G); (A,D,H,K); (A,D,I); (A,D,J).

O método de enumeração explícita, ou busca exaustiva, é o método considerado de menor eficiência em termos de tempo de processamento, apesar de sua simplicidade de implementação, já que gera todas as soluções possíveis, mas pode se enumerar todas as soluções que chegam ao valor ótimo. Outro motivo para a escolha desse método é a falta de regras conhecidas, que pudessem ser utilizadas para se propor um algoritmo eficiente.

2.8 - Método Heurístico - “Meta-heurísticas”

Entende-se por otimização não um processo de busca do melhor absoluto, mas a procura sistemática do melhor prático, (Novaes, 1978). Algumas vezes, conhecer o mecanismo de certo problema, determinando as relações entre as variáveis, já constitui um avanço considerável. Ainda segundo Novaes (1978), modelos desse tipo, embora não objetivando a otimização no sentido estrito, ainda assim podem ser enquadrados nessa categoria visto que podem levar a resultados práticos melhores do que os modelos que só abordam o ótimo.

Nesta seção serão vistas as definições encontradas na literatura das heurísticas chamadas “meta-heurísticas” e também será vista com mais detalhes a “meta-heurística” Busca Tabu, que foi escolhida para implementação neste trabalho.

O termo “meta-heurística” foi usado primeiramente no mesmo trabalho em que foi introduzido o termo Busca Tabu (Glover, 1986) e vem sendo muito utilizado na literatura devido ao grande sucesso de algumas destas “meta-heurísticas”.

A abundância de dificuldades em problemas de otimização, encontradas na prática, em problemas de telecomunicações, logística, planejamento financeiro, transportes, projetos e produção, têm motivado o desenvolvimento de técnicas de otimização cada vez mais poderosas. Estas técnicas são normalmente o resultado de adaptações de ideias vindas de uma variedade de áreas de pesquisas. A intenção é desenvolver procedimentos que sejam eficientes e capazes de lidar com uma grande gama de problemas de otimização. As ideias que motivam estes desenvolvimentos às vezes vêm de lugares inesperados. Alguns destes procedimentos, por exemplo, se baseiam em partes de modelos que exploram ideias da eletricidade e da hidráulica.

“Meta-heurísticas” referem-se a estratégias que guiam e modificam outras heurísticas para produzir soluções melhores que não se limitem a ótimos locais. Assim, uma grande vantagem das “meta-heurísticas” é de que todas têm algum procedimento para se escapar de ótimos locais. O procedimento de escape dos ótimos locais e maiores detalhes da “meta-heurística”, usada neste trabalho, serão vistos mais adiante. Além desta característica de escape de ótimos locais, as “meta-heurísticas” têm outra vantagem para a resolução de problemas, que é conhecida como robustez.

Segundo Goldberg (1989), a robustez tem sido o tema central da pesquisa com “meta-heurísticas”, ou seja, o equilíbrio entre a velocidade de convergência para a solução e a qualidade desta solução. Para Bazaraa, Shereli e Shetty (1993), a robustez é a habilidade de um procedimento (algoritmo) resolver a maioria dos problemas na classe de problemas para o qual foi designada com razoável precisão, não havendo nenhuma menção à velocidade de convergência. Para o presente trabalho, robustez engloba velocidade de convergência, que é importante para heurísticas, precisão da solução encontrada e abundância de problemas resolvidos com esta velocidade e precisão.

Para Goldberg (1989), as implicações da robustez para sistemas artificiais são muitas. Se sistemas artificiais podem ser feitos mais robustos, custos de se refazer projetos podem ser reduzidos ou eliminados. Se altos níveis de adaptação podem ser conseguidos, existem sistemas que podem tornar suas funções mais longas e melhores.

As “meta-heurísticas”, devido a sua robustez, são muito utilizadas para pesquisa em espaços complexos (Dias, 1998). Muitos trabalhos e dissertações têm estabelecido a validade da abordagem para problemas que requerem uma busca eficiente e efetiva. Assim, as “meta-heurísticas” agora são mais comumente encontradas em aplicações científicas, de negócios e de engenharia. Isto se deve ao fato de que as “meta-heurísticas” são computacionalmente simples e poderosas na busca de pontos ótimos, nos mais variados tipos de problemas de otimização. Outro fator é que as “meta-heurísticas” não são limitadas por hipóteses, como os algoritmos tradicionais, acerca de continuidade, “unimodalidade”, existência de derivadas, etc.

2.9 - “Meta-heurística” Escolhida - Busca Tabu

A “meta-heurística” Busca Tabu foi escolhida devido a sua grande utilização em problemas de natureza combinatória, sua robustez e também porque foram feitos testes preliminares com as “meta-heurísticas” Recozimento Simulado “(*Simulated Annealing*)” e Algoritmo Genético e os melhores resultados foram com a Busca Tabu. Claro que estes resultados dependem muito da implementação, não sendo possível afirmar categoricamente que uma “meta-heurística” é melhor que a outra.

A “meta-heurística” Busca Tabu foi desenvolvida por Glover e Laguna (em 1997 eles publicaram o seu trabalho intitulado Tabu Search) como sendo uma técnica de busca para resolver uma grande variedade de problemas NP-Hard. Para James (1996), este método se classifica como “meta-heurística”, já que é uma melhora de uma heurística construtiva e que foge de ótimos locais. Assim, a heurística construtiva é usada em conjunto com a Busca Tabu, o espaço de soluções e a vizinhança a ser pesquisada. A vizinhança define todos os possíveis movimentos, a partir do ponto atual, no espaço de soluções. A Busca Tabu escolhe a próxima solução que melhore a função objetivo. O movimento feito então se torna proibido (tabu) e não pode ser visitado novamente até que um critério predefinido seja alcançado ou que encontre algum critério de aspiração, que usualmente é obter um valor melhor que o melhor atual.

Para Baker (1995), também, uma das limitações dos procedimentos de pesquisa em vizinhança é o de se encontrar um ótimo local. Com a “meta-heurística” Busca Tabu esta limitação é atenuada, ou até eliminada, pois este trabalha como guia de busca local direcionando seus movimentos para a exploração de diferentes regiões do espaço de soluções, assim evitando que a mesma estacione em ótimos locais.

Definições Básicas

De acordo com Glover (1997), Tabu é definido como algo profano, proibido de usar ou contatar, algo que deve ser banido pela moralidade ou algo que deve ser tratado como perigoso que tem riscos ao ser tratado. A “meta-heurística” Busca Tabu não utiliza literalmente estas considerações a respeito do tabu, o que ela faz é simular esta definição para certos movimentos do procedimento de busca, ou seja, impondo restrições a estes movimentos. Estas restrições operam de várias formas, todas por exclusão direta de alternativas de procura, classificadas como proibidas. Estas restrições são impostas ou criadas levando em conta as estruturas de memórias que são específicas para cada tipo de problema (Glover, 1997).

Busca Tabu é baseada na premissa de que, para se resolver problemas, se devem incorporar memória adaptativa e exploração responsiva (Glover, 1997). Uma boa analogia é um alpinista, que na escalada coloca assessórios de escalada que devem ser lembrados onde estão (usando memória adaptativa) e também se deve poder fazer escolhas estratégicas de caminhos alternativos (usando exploração responsiva).

As características da memória adaptativa da Busca Tabu permitem a implementação de procedimentos que são capazes de pesquisar o espaço de soluções economicamente e eficazmente. Como as escolhas locais são guiadas por informações coletadas durante a pesquisa, a Busca Tabu contrasta com modelos sem memória que confiam muito em processos semi-aleatórios que implementam uma forma de amostra. Exemplos de métodos sem memória incluem heurísticas gulosas, o tradicional Recozimento (Annealing) e aproximações evolucionárias (Glover, 1997). Memória adaptativa também contrasta com modelos de memória rígida, típica de estratégias de Branch and Bound.

A ênfase na exploração responsiva na Busca Tabu, quer na implementação “determinística” ou probabilística, deriva da suposição de que uma estratégia de escolha ruim rende mais informações do que uma boa escolha aleatória (Glover, 1997).

Exploração responsiva integra os princípios básicos da pesquisa inteligente. Assim, a Busca Tabu está preocupada em encontrar maneiras novas e mais eficientes de levar vantagens do mecanismo de associar a memória adaptativa e a exploração responsiva.

O desenvolvimento de novas estratégias mistas faz da Busca Tabu uma área fértil para pesquisa e o estudo empírico.

Uso da Memória

A estrutura de memória da Busca Tabu opera a partir de quatro dimensões principais, que consistem em: acontecimentos recentes, frequência, qualidade e influência, conforme mostrado na figura 2.5.

Figura 2.5 - Quatro dimensões da Busca Tabu

Guardar acontecimentos recentes e frequentes é a base da memória e se complementam. A dimensão qualidade refere-se à habilidade de diferenciar o mérito das soluções visitadas durante a pesquisa. Neste contexto, a memória pode ser utilizada para identificar elementos que são comuns às boas soluções ou partes destas soluções. Operacionalmente, a qualidade torna-se um fundamento que incentiva a aprendizagem, encorajando a visita a boas soluções e penalizando visitas onde se encontram más soluções. A flexibilidade na estrutura da memória permite a pesquisa em um ambiente de múltiplos objetivos, podendo assim direcionar a pesquisa. Com a possibilidade de guardar acontecimentos recentes, é possível trabalhar com memória de curto prazo, tornando a pesquisa mais rápida, é com a memória de curto prazo que se são definidos os movimentos que devem se tornar “tabu”, durante um certo período, para se escapar de soluções ótimas locais. Com a frequência, é possível trabalhar com estruturas que nos permitam a diversificação ou intensificação do espaço de soluções possíveis. A dimensão influência considera o impacto das escolhas feitas durante a pesquisa, não somente a qualidade como também a sua estrutura. Registrar informações sobre a influência de escolhas, em particular das soluções incorporadas, se torna um novo nível de aprendizagem que pode ser usado na Busca Tabu. Está claro que certas decisões têm mais influência do que outras como uma função da vizinhança de movimentos empregados e a maneira pela qual esta vizinhança é negociada (Glover, 1997).

Para Glover (1997), a memória usada na Busca Tabu é explícita e atributiva. Memória explícita registra as soluções completas, consiste tipicamente na elite das soluções visitadas durante a pesquisa. Uma extensão desta memória registra vizinhanças de elite de soluções atrativas, mas não exploradas. A elite de soluções memorizada é usada para expandir o local de pesquisa, em alguns casos a memória explícita tem sido usada para guiar a pesquisa e evitar a visita de soluções mais de uma vez. Esta aplicação é limitada, porque uma estrutura de dados inteligente deve ser projetada para evitar a excessiva necessidade de memória.

Alternativamente, a Busca Tabu usa memória atributiva para guiar propostas. Este tipo de memória registra informações sobre atributos de soluções que mudam o movimento de uma para a outra. Em programação de produção, o índice das tarefas pode ser usado como atributo para inibir ou encorajar o método para seguir certas direções de pesquisa (Glover, 1997).

Intensificação e Diversificação

Para Glover (1997), dois componentes muito importantes da Busca Tabu são as estratégias de intensificação e diversificação. A estratégia de intensificação é baseada na modificação das regras de escolha para encorajar movimentos que encontrem combinações de soluções já visitadas que tenham boas características. A intensificação pode também iniciar um retorno a regiões já visitadas e que tenham boas características. Como a elite das soluções deve ser registrada de modo a examinar suas vizinhanças imediatas, a memória explícita é implementada para o uso da estratégia de intensificação. A estratégia de diversificação, diferentemente da intensificação, promove a visita à regiões não visitadas para gerar soluções que diferem muito das já vistas. De novo, tal aproximação pode ser baseada na geração de “sub-montagens” de componentes de soluções que produzam soluções inteiras ou pode confiar em avaliações modificadas como as incorporadas, por exemplo, no uso de funções de punições/incentivo.

Estratégias de intensificação requerem um meio para identificar um conjunto de soluções de elite como para incorporar bons atributos dentro das soluções criadas recentemente. O conjunto de soluções de elite é frequentemente determinado fixando-se um princípio que é

conectado com o valor da função objetivo da melhor solução encontrada durante a pesquisa. Entretanto, considerações de agrupamento e desfazer agrupamento são também importantes na geração de tal conjunto e, mais particularmente, para geração de subconjuntos de soluções de elite que possam ser usadas para uma específica fase da intensificação.

Matriz de Reincidência

A matriz de reincidência trata-se de uma estratégia de intensificação e diversificação que busca melhorar o desempenho do método Busca Tabu. É uma matriz $N \times N$, em que N , neste trabalho, é o número de tarefas e são registrados, ainda neste trabalho, as tarefas e as posições das tarefas permutadas entre as subsequências. As colunas indicam as N posições em uma sequência e as linhas são as N tarefas. Assim, a cada sequência que se gera esta matriz se atualiza da seguinte maneira: em cada tarefa da sequência, verifica-se a posição que ela está ocupando, assim se vai até a linha da matriz e para esta tarefa incrementa-se 1 na coluna da matriz correspondente à posição que esta tarefa ocupa.

Construindo-se a matriz conforme descrito, se tem em cada posição a frequência de ocupação das tarefas. Assim, se a intenção for utilizar uma estratégia de intensificação basta se gerar uma sequência em que cada posição é assumida pela tarefa com maior frequência. Já para a estratégia de diversificação cada posição é assumida pela tarefa com menor frequência.

Generalizando

A Busca Tabu é uma “meta-heurística” cuja eficiência está ligada ao uso das estratégias de memória que são divididas em duas:

- Memória de curto prazo (básica): é a memória que guarda atributos de soluções que mudaram recentemente, é com este tipo de memória que será definido os movimentos que não serão visitados (tabus) durante um certo tempo, desta forma fugindo de ótimos locais;

- Memória de longo prazo: armazena características das soluções já visitadas para direcionar a busca para movimentos muito frequentes ou pouco frequentes.

Em alguns casos, a memória de curto prazo é suficiente para gerar soluções suficientemente boas, exigindo um tempo de processamento menor do que se usando as memórias de longo prazo conjuntamente. Porém, existem muitos casos em que o uso da memória de longo prazo torna o procedimento de busca muito mais potente e assim valendo a pena o gasto a mais no tempo de processamento.

A memória de longo prazo normalmente se baseia na frequência de transição e de residência. A frequência de transição mantém um registro da frequência com que atributos se modificam e a frequência de residência mantém o registro da quantidade de vezes que cada atributo aparece nas soluções. A partir destas frequências podem ser aplicados punições ou incentivos aos movimentos candidatos na pesquisa. Com a utilização da memória de longo prazo, podem-se implementar duas estratégias muito importantes que são: intensificação e diversificação.

A intensificação favorece combinações de movimentos e características de soluções que historicamente tenham sido bons e, também, incentivam o regresso a regiões atrativas para analisá-las de forma mais detalhada.

A diversificação promove a condução da pesquisa a novas regiões. Ela é importante para estruturas de vizinhança em que certas trajetórias de busca podem tornar-se isoladas e distantes de novas alternativas promissoras. E usualmente estão baseadas em modificar o processo de escolha a partir das frequências de residência e de transição.

A figura 2.6 mostra um possível diagrama de blocos resumido da “meta-heurística” Busca Tabu.

Figura 2.6 – Diagrama de blocos da Busca Tabu

2.10 – Local Branching

Uma outra heurística, que também tem procedimentos de fuga de ótimos locais, assim também podendo ser tratada de “meta-heurística”, é o *Local Branching*. Esta é uma heurística promissora e foi desenvolvida por Fischetti e Lodi (2003), como uma alternativa para resolver problemas de programação inteira mista de grande porte, podendo utilizar para resolução pacotes computacionais, tal como o pacote computacional CPLEX. Esta heurística gerencia ou delimita uma sub-região do espaço de solução que será explorada. A busca por soluções melhores possibilita que novas soluções sejam encontradas de maneira mais fácil e rápida, quando comparado com o desempenho dos softwares de otimização existentes, que exploram todo o espaço de solução. Fischetti e Lodi (2003).

A heurística *Local Branching* é usualmente precedido pelo algoritmo *Branch and Bound*, este último gera uma solução inicial, porém esta solução inicial pode ser gerada por um outro algoritmo ou heurística. O *Local Branching* adiciona uma nova restrição para reduzir o espaço de soluções. Se uma solução posterior for encontrada com um valor de função objetivo melhor, dentro de um limite de tempo de processamento, então um novo problema é gerado e, deste modo, um novo espaço de soluções fica gerado. A seguir o *Branch and Cut* volta a ser executado para se encontrar novas soluções enquanto houver melhoria nessas soluções, respeitando o limite de tempo, Fischetti e Lodi (2003). Se não houver melhoria na solução, a vizinhança de busca é reduzida, etapa chamada de intensificação e se for constatado inviabilidade ou, em um tempo estipulado para a busca, uma solução viável não for encontrada, a heurística passa para uma etapa de diversificação.

A estrutura genérica a ser empregada para o problema de Programação Inteira Mista (MIP) desenvolvida por Fischetti e Lodi (2003) é:

$$\begin{aligned} \text{(P)} \quad & \min c^T x \\ & Ax \geq b \\ & x_j \in \{0,1\} & \forall j \in B \neq \emptyset \text{ (Conjunto vazio)} \\ & x_j \geq 0, \text{ inteiro} & \forall j \in G \\ & x_j \geq 0 & \forall j \in C \end{aligned}$$

Onde (B, G, C) correspondem às variáveis binárias, inteiras e reais.

Uma sub-região é construída na vizinhança de uma solução existente. Para um parâmetro k informado, define-se como vizinhança o conjunto de soluções de (P) que satisfazem à restrição de ramificação local (“*local branching constraint*”):

$$\Delta(x, \bar{x}) = \sum_{j \in \bar{S}} (1 - x_j) + \sum_{j \in B \setminus \bar{S}} x_j \leq k$$

sendo: $\bar{x}$ solução de referência de (P) e $\bar{S} = \{j \in B : \bar{x}_j = 1\}$

Esta restrição faz com que, no máximo, k variáveis binárias sejam exploradas simultaneamente.

O algoritmo LB gera uma solução inicial $\bar{x}^1$ para o problema MIP. Em seguida, uma nova restrição é adicionada para reduzir o espaço de solução X , usando a função de distância $\Delta(x, \bar{x}^1)$, formando o conjunto $X_1 = X \cap N(\bar{x}^1, k)$. Se uma solução $\bar{x}^2$ for encontrada com o valor da função objetivo melhorado então o novo espaço de solução X_2 para este problema fica definido como:

$$X_2 = X \cap (N(\bar{x}^2, k) \setminus N(\bar{x}^1, k)) = (X \setminus X_1) \cap N(\bar{x}^2, k).$$

Este procedimento de ramificação será iterado enquanto houver melhoria nos valores da função objetivo, respeitando o limite de tempo. Após l iterações, l restrições terão sido acrescentadas e o espaço de soluções terá a seguinte estrutura:

$$X_l = (X \setminus X_1 \setminus X_2 \dots \setminus X_{l-1}) \cap N(\bar{x}^l, k)$$

Se a solução não for melhorada em um certo tempo, a vizinhança de X_l é reduzida, substituindo k por $k/2$ ($\Delta(x, \bar{x}^1) \leq k/2$), no passo chamado intensificação. Caso haja a constatação de inviabilidade ou, se durante o limite de tempo uma solução viável não for encontrada, ocorre o passo de diversificação agora substituindo k por $k/2$ ($\Delta(x, \bar{x}^1) \leq k + k/2$), uma restrição do tipo $\Delta(x, \bar{x}^1) > 1$ é introduzida (visando forçar a exploração desta vizinhança) e todas as demais restrições de ramificação são removidas. O número de diversificações que poderão ser executadas durante a busca é limitado por um parâmetro e, por último, tem-se um parâmetro que limita a duração do LB.

2.11 - Caracterização do Problema Objeto deste Estudo

Neste capítulo, até agora, foi caracterizado o tipo de produção na qual se enquadra este estudo, que seria a produção intermitente; também foi definido que este estudo se enquadra na programação da produção. O modelo de programação utilizado para a tomada de decisões é a “sequenciação”. A medida de desempenho escolhida foi a soma dos custos devidos a adiantamentos, que podem ser de armazenagem, e dos custos devidos a atrasos, que podem ser devido a multas ou insatisfação dos clientes, sendo que este tipo de medida de desempenho caracteriza os problemas conhecidos na literatura como de adiantamentos/atrasos (earliness/tardiness).

O ambiente tratado será o *flow shop*, que consiste em programar n tarefas para serem processadas em um conjunto de m máquinas distintas, tendo fluxo unidirecional de processamento e tendo a característica de as tarefas poderem pular máquinas na sequência de atendimento.

O detalhamento do problema escolhido e sua modelagem serão vistos mais adiante neste trabalho, mas já se pode dizer que o problema objeto desta tese incorpora particularidades que, em conjunto, o diferenciam dos encontrados na literatura, tais como:

- Admite-se que as datas de entrega terão intervalo independente para cada tarefa, designado por janelas de tempo; assim, se a tarefa terminar antes do início de sua janela de tempo, será punida por adiantamento, se terminar depois do fim da janela, será punida por atraso e se terminar dentro deste intervalo, não será punida;
- Em uma das abordagens propostas, não se considera a inserção de tempos de espera (ociosidade da máquina) no atendimento das tarefas, ou seja, se uma tarefa estiver para ser atendida, ela será atendida e não esperará por outra tarefa, isto é um procedimento que se usa na prática e visa diminuir a ociosidade nas máquinas. Nesta modelagem, a posição das tarefas na sequência determinada será sua prioridade de atendimento e esta sequência poderá sofrer mudanças ao longo do processamento. Isto será comentado melhor mais à frente;
- Em outras duas abordagens, será considerada a inserção de tempo de espera (ociosidade da máquina); isto também será mais bem descrito mais adiante;

- Uma outra característica importante, neste trabalho, é a possibilidade de as tarefas pularem máquinas no processamento. Esta característica de pular tarefas, segundo Ruben (2006), refere-se ao ambiente de um *hybrid flexible flow shop problem* e este tipo de ambiente é realista e pode aparecer em setores como o de processamento de alimentos, manufatura de telhas de cerâmica e muitos outros. No trabalho de Ruben (2006), a medida de desempenho é o *makespan*.

A consideração de janelas de tempo auxilia na flexibilização do problema e é utilizada em outros trabalhos como em Koulamas (1996), que estuda o problema de minimizar a soma das penalidades de atrasos e adiantamentos para uma única máquina com janelas de tempo. Em outros, podem ser consideradas, no lugar de janelas de tempos, tolerâncias nas datas de entrega, como no caso de Wilamwsky, Epstein e Dickan (1996), que trata de problemas com máquina única e Biskup e Cheng (1999), que trata do problema de máquinas paralelas.

A não consideração de tempos ociosos nas máquinas é uma simplificação que facilita a busca e foi encontrada em trabalhos como os de Liaw (1999) e Li (1997), que tratam do problema de máquina única e, em sua modelagem, consideram a determinação de limites mínimos e máximos (*lower* e *upper bounds*) eficientes para a função objetivo, por meio de relaxação “lagrangeana”, em Liaw e Li, e a incorporação do método *branch and bound* (ramifica e poda), somente em Liaw.

Segundo Baker (1997), em geral uma programação requer além da decisão de “sequenciação”, a decisão a respeito de alocação de recursos. Quando existe apenas um recurso, a alocação daquele recurso é completamente determinada pela decisão do “sequenciação”. Como consequência, não existe distinção entre “sequenciação” e alocação de recursos nos modelos de uma única máquina. Em modelos com múltiplas máquinas, entretanto, existe uma distinção fazendo com que a modelagem de problemas com uma única máquina seja diferente da modelagem do estudo em questão. Na teoria da programação, segundo Baker (1997), existem três tipos básicos de modelos com múltiplas máquinas: sistemas em paralelo, sistemas em séries (*flow shop*) e sistemas híbridos (*job shop*).

Segundo Ruben, Funda e Thijs (2006), as pesquisas, na literatura, conduzem a diversos experimentos de problemas com conjuntos de dados aleatórios, bem como problemas com os dados reais, que podem aparecer em setores como processamento de alimentos, produção

de telhas cerâmicas e muitos outros. Embora muitos aspectos e restrições realísticas sejam considerados em diversos artigos na literatura, muito poucos artigos os examinam conjuntamente. Para Ruben (2006), até aquele momento não havia nenhuma tentativa de considerar juntamente estas diversas variantes do problema básico. Uma das principais características realistas que Ruben (2006) considera é a possibilidade de as tarefas poderem pular máquinas no atendimento, ele também considera diversas máquinas em paralelo por estágio. Outros trabalhos incluem nas definições de seus problemas considerações e restrições realísticas, tais como o de Kocchhar et al (1988), que faz uma pesquisa local para um problema do tipo “*flexible flow line*” considerando tempos de preparação, possibilidade de bloqueio, paradas e tempos de reinício. Kurs e Askin (2003) desenvolveram uma heurística para resolver um problema do tipo “*flexible flow line*” usando máquinas paralelas idênticas e considerando sequências dependentes dos tempos de preparação, com objetivo de minimizar o “*makespan*”.

Na tabela 2.1 é apresentado um resumo de alguns trabalhos analisados na literatura, que apesar de serem bem diferentes do tratado no presente trabalho, podem dar uma idéia de como a pesquisa de problemas de programação pode caminhar.

Tabela 2.1 – Resumo de alguns trabalhos visto na literatura

Autor	Med.Des.	Tipo	Hipóteses	Metodologia
Koulamas (1996)	A/A	M.U	Janelas de Tempo	Heurística
Wilamwsky et al (1996)	Min. Desv. abs	M.U	Tolerância	Heurística
Biskup e Cheng (1999)	A/A	M.P	Tolerância	Heurística
Liaw (1999)	A/A	M.U	N/ ociosidade	Rel. Lag./B&B
Li (1997)	A/A	M.U	N/ ociosidade	Rel. Lag.
Kochhar et al (1988)	<i>Makespan</i>	M.Est	Tempos preparação/Bloqueio	MIP
Kurs e Askin (2003)	<i>Makespan</i>	M.P	Tempos de preparação	Heurística
Zhu e Hedy (2000)	A/A	M.P	Tempos de Preparação	MIP
Rubén et al (2006)	<i>Makespan</i>	M.Est	Tempos de Preparação	MIP/Heurística

Dois trabalhos analisados mais profundamente foram: Zhu e Hedy (2000) e Ruben, Funda e Thijs (2006). No trabalho de Zhu e Hedy (2000), foi desenvolvido um

algoritmo de Programação Linear Inteira Mista (MIP) com o objetivo de minimizar a soma das punições por adiantamentos e atrasos para um problema com múltiplas máquinas, com tarefas independentes, tempo de processamento dependendo da combinação tarefa/máquina, datas de entregas distintas para cada tarefa e penalidades e tempos de preparação das tarefas também distintos para cada tarefa. As diferenças do trabalho de Zhu e Heady e este é a consideração do tempo de preparação das tarefas e, apesar de considerar varias máquinas, cada tarefa só pode passar por uma delas, deste modo não sendo um *flow shop*. Em Zhu e Heady (2000), é comentado que o modelo pode encontrar facilmente a sequência ótima em problemas com nove tarefas e três ou menos máquinas.

O trabalho de Ruben, Funda e Thijs (2006) propõe uma formulação que utiliza Programação Linear Inteira Mista (MIP) para pequenas instâncias e uma heurística simples, para problemas maiores, utilizando o algoritmo NEH proposto por Nawaz, Ensore e Ham (1983), para problemas que trabalhem com ambientes de programação *flow shop* admitindo restrições realísticas simultâneas. Estes problemas podem incluir instantes em que as máquinas ficam disponíveis para a programação, máquinas paralelas não relacionadas em cada estágio, máquina com prioridades, tarefas que podem pular máquinas no atendimento e operações com tempo de espera.

O presente trabalho usou a formulação proposta por Ruben, Funda e Thijs (2006) como ponto de partida para a formulação do modelo matemático apresentado no capítulo 3; apesar do trabalho deles ser diferente, considerando múltiplos estágios, cada estágio com várias máquinas em paralelo, não considerando janelas de tempo, considerando tempos de preparação e considerando o *makespan* como medida de desempenho, que é uma medida regular diferente da considerada no presente trabalho. Considerando estas diferenças, para a formulação proposta no presente trabalho houve a necessidade de incluir restrições que evitem ciclos de tarefas numa dada máquina, enquanto que na formulação de Rubén não foi incluídas estas restrições.

2.12 – Modelagens Matemáticas em Problemas em Ambiente *Flow Shop*

No capítulo 3 será apresentada a modelagem matemática para o problema cerne deste trabalho e nesta seção serão mostradas algumas modelagens encontradas na literatura.

Morton e Pentico (1993) propõem uma modelagem matemática para o problema de programação de tarefas em ambiente *flow shop* com múltiplas máquinas, utilizando o *makespan* como medida de desempenho. Convém observar que neste *flow shop* cada tarefa passa por todas as máquinas. As variáveis de decisão e os parâmetros dos modelos de Morton e Pentico são:

$x_{j,k,t}$ é uma variável de decisão binária, que assume o valor 1, se a tarefa j é atendida na máquina k no período t , e zero, caso contrário;

$c_{j,k}$ é o instante de conclusão da tarefa j na máquina k ;

$c_{j,k,t}$ é uma variável de decisão binária, que assume valor 1, se $c_{jk} = t$, e zero, caso contrário;

c_m é a variável de decisão que expressa o instante de conclusão de todas as tarefas;

$p_{j,k}$ é tempo de processamento da tarefa j na máquina k ;

T é o número total de períodos de tempo no modelo (estimado a partir de heurísticas);

r_j é o instante de chegada da tarefa j ;

a_k é o instante em que a máquina k fica disponível;

n é o número de tarefas.

O modelo matemático proposto é o seguinte:

$$\text{Min}(c_m) \tag{2.1}$$

sujeito as seguintes restrições:

$$c_{j,k} = \sum_{t=1, T} t.c_{j,k,t}, \quad \text{para todo } j, k \tag{2.2}$$

$$x_{j,k,t} = \sum_{u=t, t+p_{j,k}-1} c_{jku}, \quad \text{para todo } j, k, t \quad (2.3)$$

$$c_m \geq c_{j,m}, \quad \text{para todo } j \quad (2.4)$$

$$c_{j,1} \geq r_j + p_{j,1}, \quad \text{para todo } j \quad (2.5)$$

$$c_{j,k} \geq c_{j,k-1} + p_{j,k}, \quad \text{para todo } j, k \quad (2.6)$$

$$\sum_{j=1,n} x_{j,k,t} \leq 1, \quad \text{para todo } k, t \quad (2.7)$$

$$x_{j,k,t} = 0 \quad t < a_k, \quad \text{para todo } j, k \quad (2.8)$$

$$c_{j,k,t} \in \{0,1\}, \quad \text{para todo } j, k, t \quad (2.9)$$

A restrição (2.2) garante que precisamente um termo da soma será não nulo. A restrição (2.3) impõe que a tarefa j será processada na máquina k no tempo t , se e somente se for completada em algum instante entre t e $t + p_{j,k} - 1$. A restrição (2.4) impõe que o *makespan* deve garantir a conclusão de cada tarefa. A restrição (2.5) impõe que nenhuma tarefa pode começar enquanto não chegar ao shop. A restrição (2.6) impõe que o instante de conclusão da tarefa j na máquina k deve ser no mínimo igual ao instante de conclusão na máquina $(k-1)$ acrescido do tempo de processamento na máquina k . A restrição (2.7) impõe que uma máquina não pode processar mais de uma tarefa por vez.

Já para problemas do tipo *flow shop* “Permutacional”, que são problemas onde se considera que o atendimento das tarefas siga uma sequência determinada, ou seja, todas as tarefas passam por todas as máquinas na mesma ordem, Morton e Pentico (1993) propôs em sua modelagem matemática um índice s , que designa a posição no atendimento e

como variável de decisão ele propõe uma variável binária z_{js} que assume valor 1 caso a tarefa j esteja na posição s e zero, caso contrário. Este tipo de variável não é apropriada quando as tarefas podem pular máquinas, assim não seguindo a mesma sequência.

No trabalho de Zhu e Heady (2000), já mencionado anteriormente, é apresentado um modelo de Programação Linear Inteira Mista (MIP) com o objetivo de minimizar a soma das punições por adiantamentos e atrasos para um problema com múltiplas máquinas em que cada tarefa só pode ser processada por uma, não se tratando de ambiente *flow shop*, com tarefas independentes, tempo de processamento dependendo da combinação tarefa/máquina, datas de entregas distintas para cada tarefa e penalidades e tempos de preparação das tarefas também distintos para cada tarefa.

Para a elaboração do modelo matemático, Zhu e Heady utilizaram os seguintes parâmetros e variáveis de decisão:

B um número muito grande;

d_i é a instante desejado para entrega da tarefa i ;

e_i é a penalidade por adiantamento na conclusão da tarefa i ;

E_i é o intervalo de tempo com que a tarefa i se adianta (em relação ao instante de entrega desejada) ;

$p_{i,m}$ é o tempo de processamento da tarefa i na máquina m ;

M é o número de máquinas;

N é o número de tarefas;

$s_{j,i}$ é o tempo de preparação da tarefa i , quando ela sucede imediatamente a tarefa j ;

$s_{0,i}$ é o tempo de preparação para a tarefa i , quando ela é a primeira na fila;

t_i é a penalidade por atraso na conclusão da tarefa i ;

T_i é o intervalo de tempo com que a tarefa i se atrasa (em relação ao instante de entrega desejada);

X_i é o instante de conclusão da tarefa i ;

$Y_{i,j,m}$ é uma variável binária, que assume valor 1, se a tarefa i precede a tarefa j na máquina m , e valor zero, caso contrário;

$Z_{i,m}$ é uma variável binária, que assume valor 1, se a tarefa i é processada na máquina m , e valor zero, caso contrário;

O modelo matemático de Zhu e Heady é:

$$\text{Min} \sum_{i=1}^N (e_i E_i + t_i T_i) \quad (2.10)$$

Sujeito as restrições:

$$X_i - T_i + E_i = d_i \quad i = 1, \dots, N \quad (2.11)$$

$$\sum_{m=1}^M Z_{i,m} = 1 \quad i = 1, \dots, N \quad (2.12)$$

$$\sum_{j=0}^N Y_{i,j,m} \leq Z_{i,m} \quad j \neq i, \quad i = 0, \dots, N; \quad m = 1, \dots, M \quad (2.13)$$

$$\sum_{i=0}^N Y_{i,j,m} \leq Z_{j,m} \quad i \neq j, \quad j = 1, \dots, N; \quad m = 1, \dots, M \quad (2.14)$$

$$X_i - X_j - BY_{j,i,m} \geq p_{i,m} + s_{j,i} - B \quad j \neq i; \quad i = 1, \dots, N; \quad j = 0, 1, \dots, N; \quad m = 1, \dots, M \quad (2.15)$$

$$X_i, T_i, E_i \geq 0 \quad i = 1, \dots, N \quad (2.16)$$

$$Y_{i,j,m} \in \{0, 1\} \quad i = 1, \dots, N; \quad j = 0, 1, \dots, N; \quad m = 1, \dots, M \quad (2.17)$$

Segundo Zhu e Heady, a restrição (2.11) mede o atraso ou adiantamento de cada tarefa; a restrição (2.12) garante que uma tarefa só pode ser processada por uma máquina. As restrições (2.13) e (2.14) garantem que, imediatamente antes e imediatamente depois de cada

tarefa deve haver somente uma tarefa. A restrição (2.15) garante que o instante de conclusão de uma tarefa i deve estar suficientemente distante do instante de uma tarefa j para incluir os tempos de preparação e processamento da tarefa i .

Rubén, Funda e Thijs (2006) consideram a programação de tarefas em um *flow shop* flexível, com varias máquinas em cada estágio. Para a modelagem matemática, propõe o *makespan* como função objetivo; entre outras, as restrições impostas são as seguintes: uma dada tarefa só pode ser precedida por uma tarefa em cada máquina, em cada estágio; uma dada tarefa só pode ser sucedida por uma tarefa em cada máquina, em cada estágio; se uma tarefa é processada em uma dada máquina em um estágio, então deve ter um antecessor nesta mesma máquina (para que a restrição se aplique a todas as tarefas, Rubén, Funda e Thijs introduzem uma tarefa fictícia de índice zero, em cada máquina, com tempo nulo de processamento); não pode haver a ocorrência de precedências cruzadas; a tarefa fictícia zero só pode ser antecessora de uma tarefa em cada máquina; a tarefa fictícia zero é completada no instante zero em todos os estágios. Há ainda restrições para cálculo do instante de conclusão das tarefas em cada estágio. Na modelagem de Rubén, Funda e Thijs não há uma restrição geral que impeça a formação de ciclos de tarefas em uma dada máquina. As variáveis de decisão utilizadas por Rubén, Funda e Thijs (2006) foram: x_{ijk} , que assume o valor 1, se a tarefa j precede a tarefa k na máquina l , no estágio i , e zero, caso contrario e $C_{i,j}$ que é o instante de conclusão da tarefa j no estágio i .

O *flow shop* estudado por Rubén, Funda e Thijs é mais complexo que o do presente trabalho, substituindo as máquinas por estágio. Assim, as variáveis de decisão x_{ijk} de Rubén, Funda e Thijs se referem ao estágio i e a máquina l ao contrário das variáveis $x_{m,i,j}$ apresentadas no capítulo 3, que somente se refere à máquina m . Por esta razão e também por outras características específicas do *flow shop* tratado por Rubén, Funda e Thijs, como tempo de ajuste dependendo da sequência das tarefas, as expressões matemáticas do modelo de Rubén, Funda e Thijs não são aqui apresentadas. Cabe, porém ressaltar que esse modelo trouxe contribuições importantes para o modelo matemático proposto no capítulo 3.

CAPÍTULO 3

DESCRIÇÃO E MODELAGEM DO PROBLEMA

Neste capítulo será descrito o problema objeto deste trabalho, bem como sua modelagem, definindo-se os parâmetros utilizados, variáveis, estratégia de resolução. Também serão apresentados procedimentos alternativos para resolução deste problema

3.1 – Descrição do Problema

O problema objeto deste trabalho considera que, em uma série de M máquinas independentes, será processado um conjunto de N tarefas independentes, tendo fluxo unidirecional de processamento nas máquinas, sendo assim um *flow shop*. Será admitido que as N tarefas não precisem passar por todas as M máquinas, tendo cada tarefa um roteiro independente pelas máquinas. Admite-se que cada máquina poderá ter um estoque intermediário sem limitação; deste modo poderá se formar uma fila de tarefas em cada máquina. Também serão consideradas diferentes janelas de tempo para a entrega, ou seja, intervalos de tempo diferentes para cada tarefa, nos quais a tarefa pode ser entregue sem que haja punição por atraso ou adiantamento. Cada tarefa i , das N tarefas, terá um tempo de processamento “determinístico” $p(i,m)$, em cada máquina m . Para cada uma das tarefas, uma punição individual $b(i)$ será atribuída aos atrasos, caso ela termine depois do encerramento de sua janela de tempo de entrega e uma punição individual $a(i)$ aos adiantamentos, caso a tarefa termine antes da abertura de sua janela de tempo de entrega.

Será proposto na próxima seção um modelo matemático para determinar, para cada máquina a sequência de atendimento das tarefas que minimize a soma das punições por

adiantamento e por atraso. Na modelagem, são consideradas todas as possibilidades de sequências possíveis, com troca de ordem nas máquinas e considerando a inserção de tempo ocioso; esta consideração significa que uma máquina pode ficar ociosa mesmo havendo uma tarefa pronta para ser atendida.

3.2 – Formulação Matemática do Problema Geral

A modelagem matemática aqui apresentada tem como base a apresentada por Rubén, Funda e Thijs (2006), como já foi comentado na seção 2.12 do capítulo 2.

Assim são utilizadas as mesmas variáveis de decisão para definir a sequência das tarefas nas máquinas e os tempos de término das tarefas em cada máquina. Há, no entanto, diferenças significativas no modelo proposto: em primeiro lugar, são propostas restrições para evitar que haja ciclos de tarefas numa dada máquina, a exemplo do que se faz para o problema de caixeiro viajante. Em segundo lugar, a função objetivo do modelo proposto, no presente trabalho, é diferente, pois considera a soma das punições por adiantamento e atraso na conclusão das tarefas. Para tanto, são introduzidas variáveis que representam esses adiantamentos e atrasos e restrições que impeçam de premiar tarefas que não são concluídas dentro da janela de tempo.

As hipóteses usadas na programação serão então:

1. Cada máquina estará disponível continuamente sem interrupções;
2. Cada máquina pode processar apenas uma tarefa de cada vez;
3. Cada tarefa pode ser processada por uma máquina de cada vez;
4. Os tempos de processamento das tarefas nas diversas máquinas são determinados e fixos;
5. As tarefas terão a mesma data de liberação, a partir da qual, qualquer uma pode ser programada e executada;
6. Para cada tarefa i haverá um intervalo de tempo $[d_i, f_i]$ chamado de janela de tempo em que a tarefa não sofrerá nenhuma punição se sua data de término estiver dentro deste intervalo;
7. Os tempos de preparação das operações nas diversas máquinas são incluídos nos tempos de processamento e independem da sequência de operação em cada máquina;

8. As operações nas diversas máquinas, uma vez iniciadas, não devem ser interrompidas; e, finalmente,
9. Por ser um *flow shop*, o fluxo de atendimento será unidirecional, deste modo, as tarefas não poderão retornar para máquinas anteriores no roteiro.

A notação utilizada para a formulação é a seguinte:

T : é um número suficientemente grande (horizonte de planejamento).

Índices:

i/j : índice para as tarefas;

m : índice para as máquinas;

Conjuntos de tarefas e máquinas:

G_m : conjunto de tarefas que visitam a máquina m ;

CM_i : conjunto das máquinas visitadas pela tarefa i ;

M : conjunto de máquinas;

N : conjunto de tarefas, há uma tarefa fictícia de índice zero;

Parâmetros:

$p_{i,m}$: tempo de processamento da tarefa i na máquina m ;

d_i : instante inicial (limite inferior) da janela de tempo para a conclusão da tarefa i ;

f_i : instante final (limite superior) da janela de tempo para a conclusão da tarefa i ;

- a_i : punição unitária para a tarefa i por término anterior ao da abertura da janela de tempo (adiantamento);
- b_i : punição unitária para tarefa i por término posterior ao do encerramento da janela de tempo (atraso);
- n_m : número de tarefas que passam pela máquina m , (para cada máquina há uma tarefa fictícia, de índice zero, com tempo de processamento nulo);
- lm_i : última máquina pela qual passa a tarefa i ;
- $ma_{m,i}$: máquina antecessora da máquina m no fluxo da tarefa i ;

Variáveis:

- c_{mi} : instante de término ou conclusão da tarefa i na máquina m , variável não negativa;
- $fl_{m,i,j}$: fluxo da tarefa i para a tarefa j na máquina m (variável, não negativa introduzida para evitar que haja um ciclo de tarefas numa dada máquina);
- u_i : variável utilizada para medir o adiantamento na conclusão da tarefa i , definida como: $u_i = d_i - c_{lm_i,i}$ se $c_{lm_i,i} < d_i$, caso contrário $u_i = 0$, com $i \in N$;
- v_i : variável utilizada para medir o atraso na conclusão da tarefa i , definida como: $v_i = c_{lm_i,i} - f_i$ se $c_{lm_i,i} > f_i$, caso contrário $v_i = 0$, com $i \in N$;

Variável de decisão:

$x_{m,i,j}$: variável binária que assume valor 1 quando a tarefa i precede a tarefa j na máquina m e 0 caso contrário;

A resolução do problema corresponde a:

$$\text{Min} \sum_{i \in N} (a_i u_i^+ + b_i v_i^+) \quad (3.1)$$

Sujeito à:

$$\sum_{\substack{i \neq j \\ i \in \{G_m, 0\}}} x_{m,i,j} = 1 \quad j \in N, m \in CM_i \quad (3.2)$$

$$\sum_{\substack{i \neq j \\ i \in G_m}} x_{m,j,i} \leq 1 \quad j \in N, m \in CM_i \quad (3.3)$$

$$\sum_{\substack{h \neq j, h \neq i \\ h \in \{G_m, 0\}}} x_{m,h,i} \geq x_{m,i,j} \quad i, j \in N; m \in CM_i \cap CM_j \quad (3.4)$$

$$x_{m,i,j} + x_{m,j,i} \leq 1 \quad i \in N; j = i + 1, \dots, n; m \in CM_i \cap CM_j \quad (3.5)$$

$$\sum_{i \in \{G_m\}} x_{m,0,i} \leq 1 \quad m \in M \quad (3.6)$$

$$c_{m,i} \geq c_{m,j} + p_{i,m} - (1 - x_{m,j,i})T \quad m \in M; j \in \{G_m, 0\}; j \neq i \quad (3.7)$$

$$c_{m,i} \geq c_{ma_{m,i}} + p_{i,m} \quad m \in M; i \in G_m \quad (3.8)$$

$$\sum_{j \in G_m} fl_{m,0,i} \geq n_m \quad m \in M \quad (3.9)$$

$$\sum_{\substack{j \in \{G_m, 0\} \\ j \neq i}} fl_{m,j,i} - \sum_{\substack{j \in G_m \\ j \neq i}} fl_{m,i,j} = 1 \quad m \in M; i \in G_m \quad (3.10)$$

$$fl_{m,i,j} \leq n_m x_{m,i,j} \quad m \in M; i \in G_m; j \in G_m; i \neq j \quad (3.11)$$

$$u_i^+ - u_i^- = d_i - c_{lm_i,i} \quad i \in N \quad (3.12)$$

$$v_i^+ - v_i^- = c_{lm_i,i} - f_i \quad i \in N \quad (3.13)$$

$$x_{m,i,j} = 0 \text{ ou } x_{m,i,j} = 1 \quad m \in M; i, j \in N, i \neq j \quad (3.14)$$

$$fl_{m,i,j} \geq 0 \quad m \in M; i, j \in N, i \neq j \quad (3.15)$$

$$u_i^+ \geq 0; u_i^- \geq 0 \quad i \in N \quad (3.16)$$

$$v_i^+ \geq 0; v_i^- \geq 0 \quad i \in N \quad (3.17)$$

O objetivo é definir a sequência das tarefas em cada máquina com intuito de minimizar a somatória das punições por atrasos e adiantamentos. A restrição 3.2 garante que há exatamente um antecessor da tarefa j na máquina m , seja ela uma tarefa real ou a tarefa fictícia; a restrição 3.3 garante que há no máximo uma tarefa sucedendo a tarefa j na máquina m . A restrição 3.4 garante que, para que a máquina m passe da tarefa i para a tarefa j , é necessário que ela tenha processado antes a tarefa i , cuja antecessora foi uma tarefa real h ou a tarefa fictícia zero. A restrição 3.5 garante que, se j sucede i na máquina m , então j não pode suceder j na máquina m , esta restrição evita ciclos de tarefas com duas únicas tarefas, mas não evita ciclos maiores. A restrição 3.6 impõe que haverá, no máximo, uma tarefa inicial para cada máquina; a restrição 3.7 impõe que, se a tarefa i for processada na máquina m após a tarefa j , ela não poderá ser iniciada antes do término da tarefa j . A restrição 3.8 garante que o processamento da tarefa i na máquina m não poderá começar antes que termine o processamento da tarefa i na máquina anterior a m dentro do seu roteiro, ela é desnecessária em problemas de máquina única. As

restrições 3.9, 3.10, e 3.11 evitam a formação de ciclos de tarefas numa dada máquina. Isto é, evitam soluções como, por exemplo, $x_{m,0,j} = 1$, $x_{m,j,k} = 1$, $x_{m,k,i} = 1$ e $x_{m,i,j} = 1$ mostrado na figura 3.1.

Figura 3.1 – Exemplo de ciclo

Obs.: 0 é a tarefa fictícia.

No capítulo 5 serão feitos testes com problemas de diferentes tamanhos. Considerando, porém, as dificuldades de se encontrar a solução ótima para instâncias maiores de problemas, devido sua natureza combinatória, são propostas, a seguir na seção 3.3, alguns procedimentos alternativos para a resolução do problema.

3.3 – Procedimentos Alternativos para Resolução do Problema

Considerando a complexidade computacional do modelo matemático apresentado na seção 3.2 e, em consequência, a dificuldade de se obter soluções exatas para instâncias maiores do problema, decidiu-se explorar procedimentos aproximados para resolução do problema. A solução gerada por esses procedimentos, além de poder ser implementada diretamente para a “sequenciação” das tarefas no *flow shop* podeseer utilizada como solução inicial do método *Branch and Cut* (encontrado no pacote CPLEX), para explorar o modelo matemático do capítulo 3.

Duas são as propostas para tratamento aproximado do problema: aproximação do problema real e resolução do problema real utilizando uma heurística construtiva.

1 – Aproximação do problema real

Nesta aproximação somente serão examinadas as soluções com as seguintes características: gera-se uma sequência de prioridades para as tarefas. As prioridades de cada tarefa (pri_i), são as posições que as tarefas ocupam nesta sequência de prioridades e em cada máquina m , quando ela acaba de processar uma tarefa, a seguinte é aquela que está na fila e tem o menor pri_i . Assim, somente são examinadas $N!$ sequências.

2 – Resolução do problema real a partir de uma heurística construtiva

Gera-se, por meio de regras empíricas de problemas com uma única máquina (como vistas em Baker e Scuder (1989)), sequência de atendimento das tarefas em cada máquina. O atendimento é feito nesta sequência mesmo que, havendo tarefas na espera, a máquina fique ociosa aguardando a chegada da tarefa programada para suceder a tarefa que acabou de ser processada.

Quando se utiliza o primeiro procedimento aproximado, independentemente de como se explora o espaço de soluções em busca ótima, sabe-se que esse espaço é um subconjunto do espaço de soluções do problema real modelado na seção 3.2. Dessa forma, a resolução do problema aproximado fornece um limite superior para a solução ótima do problema original.

Para a primeira proposta de tratamento aproximado do problema real, foram desenvolvidas duas heurísticas de busca de sequências:

A primeira heurística gera, inicialmente, um conjunto de cinco sequências, a partir de regras empíricas de problemas com uma máquina (como vistas em Baker e Scuder (1989)) considerando apenas os parâmetros que não dependem das máquinas, para se escolher a melhor delas, a partir de uma heurística construtiva interna. A sequência escolhida como melhor servirá como sequência inicial para ser melhorada por meio de um método computacional conhecido na literatura como Busca Tabu (Tabu Search), visto na seção 2.10.

Para a segunda heurística foi utilizado o método de busca exaustiva, gerando todas as sequências de prioridade possíveis. Esta heurística garante que será obtida a sequência ótima do problema, considerando as simplificações do modelo, mas não garante o ótimo para o problema geral. Mesmo sendo uma aproximação do problema real, a busca exaustiva, que usa um método de busca em forma de árvore no presente trabalho, como visto na seção 2.8, tem sua eficiência

diminuída, em termos de tempo de processamento, conforme se aumenta o número de tarefas. Desta forma, instâncias maiores do problema, a Busca Tabu passa a ser a opção mais apropriada.

Para a segunda proposta de tratamento aproximado, foi desenvolvida uma heurística construtiva que gera também um conjunto de sequências, porém neste caso não é uma sequência única de prioridades e sim uma sequência de atendimento, efetivo das tarefas, para cada máquina. Deste conjunto de sequências, cujas regras empíricas utilizadas serão vistas no próximo capítulo, se escolhe as sequências em cada máquina que minimize a somatória de punições por atrasos e adiantamentos.

A estratégia de se gerar conjuntos de sequências, tanto na heurística da primeira proposta, como início para a Busca Tabu, como na heurística construtiva da segunda proposta, tem como objetivo explorar a estrutura do problema. Neste tipo de problema, que se pune adiantamentos e atrasos, o comportamento da punição por atraso e adiantamento de cada tarefa tem uma característica peculiar que é mostrada na figura 3.1.

Para as heurísticas desenvolvidas para a primeira e segunda proposta foi introduzido um tempo de espera para o início, pois dependendo dos valores relativos das punições por adiantamentos e por atraso, pode ser vantajoso se fazer um deslocamento no eixo do tempo para início do processamento. Esta espera será a data de liberação para o início do processamento e, o presente trabalho, será uma variável de decisão para o objetivo de minimizar a soma das punições.

Figura 3.2 - Comportamento das punições por atraso e adiantamento em relação à data de entrega para cada tarefa

CAPÍTULO 4

HEURÍSTICAS ALTERNATIVAS PROPOSTAS

No capítulo anterior foi apresentado um modelo matemático para o problema de “sequenciação” tratado neste trabalho e dada a dificuldade de resolução exata desse modelo matemático para instâncias maiores do problema, foram propostos procedimentos alternativos para obtenção de soluções aproximadas.

Neste capítulo serão detalhados estes procedimentos alternativos para resolução do problema de programação. Serão mostrados os parâmetros utilizados em cada um e as simplificações nos modelos e ainda será feita uma aplicação de cada heurística em um problema de pequeno porte.

4.1 – Introdução

Segundo Knuth (1973), o termo algoritmo vem da palavra árabe alkhwarizmi, originária do nome de um matemático persa chamado Abu Ja'far Mohammed ibn Musa al Khwarizmi, autor de um livro-Texto sobre matemática no século IX.

Em português, a palavra quer dizer “operação ou processo de cálculo”, entre outras definições possíveis. De uma maneira geral, o termo algoritmo evoca a noção de um “processo para resolver um dado problema”. Obviamente, a razão de ser do algoritmo vem da existência de uma dissonância entre o estado desejado e aquele observado na realidade. Implícito está que há uma caracterização dos estados em termos de algumas dimensões e que existe uma ordenação entre os dois estados, pois um é mais desejável que o outro, havendo uma função objetivo que mensura essa “desejabilidade” de um estado, (Swait Jr, 1991).

O algoritmo não é a solução do problema, mas o meio de obtê-la, (Swait Jr, 1991). Algoritmo também pode ser definido como uma lista de instruções para se realizar algum processo passo a passo, Forsythe et al (1989).

O termo heurística também é muito utilizado na literatura e Peal (1984) define como sendo critérios, métodos ou princípios para decidir entre diversos planos de ação as alternativas que prometem serem mais eficazes a fim de se atingir algum objetivo. Representam a combinação entre duas necessidades: a necessidade de fazer tais critérios simples e, ao mesmo tempo, a necessidade de escolher corretamente entre opções boas e ruins. Uma heurística pode ser uma regra empírica que seja usada para guiar ações para se atingir objetivos.

Resumindo, heurísticas são regras e algoritmos são instruções que podem usar estas regras, assim, uma heurística pode usar um algoritmo para ser implementada. Na literatura são usados os dois termos para se designar a implementação de procedimentos que visem otimizar um problema.

Neste trabalho será diferenciada a designação heurística da designação algoritmo, sendo tratado como algoritmo o método computacional de resolução visto no capítulo anterior, que é um método exato; e de heurísticas os procedimentos aproximados que serão propostos nesse capítulo.

Como já foi visto anteriormente, para se atingir o objetivo deste trabalho, de melhorar a eficiência de um determinado sistema produtivo, é necessário resolver o problema de programação de tarefas em múltiplas máquinas em ambiente *flow shop*, com janelas de tempo como tolerância para a entrega dos produtos finalizados, punindo os adiantamentos bem como os atrasos e com a possibilidade de as tarefas pularem máquinas. Resolver este problema significa encontrar uma sequência que minimize a soma das punições por adiantamentos e atrasos. No capítulo 3 foi proposta uma formulação para um problema geral, onde se busca uma sequência que minimize a somatória de punições, por atraso e adiantamento, considerando todas as possibilidades de sequências possíveis, com troca de ordem nas máquinas e inserção de tempo ocioso. Esta formulação, como já foi comentado, teve a função objetivo e restrições estruturadas em VBA no Excel da Microsoft e executada a partir do pacote “CPLEX” versão 10.0. Como os tempos de processamento são muito altos para execução até o final, foram propostos procedimentos alternativos de resolução. Estes procedimentos consistiram em duas abordagens com hipóteses simplificadoras em relação ao problema geral. Para a primeira dessas abordagens,

a estratégia de resolução foi admitir uma sequência de prioridades de atendimento, que será mais bem visto mais à frente neste trabalho e foram desenvolvidas duas heurísticas para ela: uma que utiliza uma “meta-heurística” conhecida como Busca Tabu e uma outra que utiliza um método de enumeração explícita de busca exaustiva. Para a segunda abordagem a estratégia de resolução não admitiu uma sequência de prioridades e sim uma sequência efetiva de atendimento das tarefas para cada máquina e admitiu inserção de tempo ocioso, para isto foi desenvolvida uma heurística construtiva. Para as abordagens com hipóteses simplificadoras, as heurísticas definem também o instante de liberação para o processamento das sequências geradas, com a finalidade de minimizar a somatória de punições. Este instante de liberação está designado como tempo de espera (*TE*).

4.2 – Heurísticas Propostas

A primeira heurística foi desenvolvida para abordagem que usa como estratégia de resolução a geração de uma sequência de prioridades de atendimento e também a não inserção de tempo ocioso nas máquinas, ou seja, quando uma tarefa estiver na fila de uma máquina, pronta para ser atendida, com prioridade maior que as outras tarefas na mesma fila, ela será atendida, sem esperar a chegada de uma tarefa com prioridade maior. Esta sequência de prioridades de atendimento das tarefas impõe que o atendimento efetivo, de cada tarefa, dependerá de suas prioridades e das filas de tarefas formadas em cada máquina, já que não há limite para a quantidade de tarefas a espera de processamento numa dada máquina. Admitida estas características, esta heurística primeiramente gera, a partir de uma heurística construtiva interna, um conjunto de cinco sequências de prioridades, para se escolher a melhor delas. A sequência escolhida como melhor servirá como sequência inicial para ser melhorada por meio de uma “meta-heurística” conhecida na literatura como Busca Tabu (Tabu Search). A heurística construtiva desenvolvida para a Busca Tabu se baseou em problemas com uma única máquina, já que existem regras bem definidas para estes problemas, como, por exemplo, como é visto em Baker (1997) e Pinedo (1995), para se minimizar o *Total flowtime*, basta colocar as tarefas em ordem crescente os tempos de processamento e o *Maximum tardiness* é minimizado colocando as

tarefas em ordem crescente de data de entrega. Problemas de adiantamento e atraso têm bons resultados colocando as tarefas, que têm data de término anterior à data de entrega, em ordem decrescente de tempo de processamento e em ordem crescente de tempo de processamento as tarefas que terminam depois da data de entrega. Estas regras também foram utilizadas neste trabalho, sendo classificadas como regras empíricas.

A segunda heurística foi desenvolvida para a abordagem acima, ou seja, a mesma forma de atendimento das tarefas, considerando a sequência gerada como uma sequência de prioridades e a não inserção de tempo ocioso nas máquinas, porém a busca da melhor sequência de prioridade se dá a partir de uma busca exaustiva, gerando todas as sequências possíveis de prioridades.

A escolha de se buscar uma sequência de prioridades de tarefas deve-se ao fato de que, elaborar uma heurística para avaliar uma única sequência de n tarefas para as m máquinas é bem mais simples que elaborar uma heurística que avalie cada sequência em cada uma das m máquinas.

A terceira heurística foi desenvolvida para a abordagem que tem como estratégia de resolução a geração de uma sequência efetiva de atendimento para cada máquina, a partir de regras empíricas de problemas com uma única máquina. Esta abordagem admite que quando uma dada máquina fica disponível para atendimento, mas entre as tarefas que estão na fila não está a próxima na sequência de atendimento desta máquina, esta máquina ficará aguardando a tarefa que seja a próxima na sequência de atendimento. Esta espera, no presente trabalho, é o chamado tempo ocioso, como definido no capítulo 3. Para cada sequência gerada, é determinado qual será seu melhor instante de liberação. Esta heurística é uma heurística construtiva que gera a sequência de atendimento efetivo das tarefas.

A seguir, serão apresentadas as três heurísticas propostas, detalhando as características que as diferenciam.

4.3 – Busca Tabu

A heurística que utilizou Busca Tabu como método de resolução foi chamada de E/TTabu. Em sua implementação, foi considerado que uma sequência inicial, a ser melhorada pelo procedimento da Busca Tabu, seria escolhida dentre cinco sequências em que se podem estabelecer regras empíricas simples para serem geradas por uma heurística construtiva; assim, a sequência que tiver a menor soma de punições será a escolhida. Em caso de empate, a sequência escolhida será a que for gerada antes. A heurística que gera a sequência inicial é chamada de construtiva porque ela constrói cada uma das cinco sequências com regras próprias para cada uma dessas sequências, sem a utilização de algum procedimento iterativo de mudança nelas.

Sequência Inicial

A primeira sequência é composta pela união de duas subsequências, que são colocadas uma após a outra para se formar a sequência inteira. Uma primeira subsequência é formada por tarefas onde as punições unitárias por atraso são maiores que as punições unitárias por adiantamento e a segunda subsequência é formada por tarefas onde as punições unitárias por adiantamento são maiores que as punições unitárias por atraso. Em caso de empate, a tarefa ocupará a subsequência com menor número de elementos e se houver empate novamente neste número, a tarefa ocupará a primeira subsequência. Na primeira subsequência, as tarefas são sequenciadas em ordem decrescente de punições unitárias por atraso, pois se for considerado só estas punições, esta regra gerará uma punição total por atraso menor no primeiro conjunto. Na segunda subsequência, as tarefas são sequenciadas em ordem crescente de punição unitária por adiantamento, de modo a minimizar a punição total por adiantamento para as tarefas da segunda subsequência. Normalmente, em problemas com máquina única estas “sequenciações”, nas duas subsequências, são divididas pelos tempos de processamento (Baker e Scuder, 1989); no presente trabalho foi tentado utilizar um tempo de processamento médio, ou seja, a média dos tempos de processamento de cada tarefa em todas as máquinas, porém todos os valores conseguidos, na função objetivo em exemplos preliminares propostos, foram mais altos. Se todas as punições por atraso forem maiores que as de adiantamentos ou vice-versa esta alternativa será contemplada na quarta e quinta sequências, assim, somente se estipula que as subsequências tenham mesmo

número de elementos caso o número de tarefas seja par, ou com a primeira subsequência com um elemento a mais caso o número de tarefas seja ímpar.

Para a geração da segunda sequência, as tarefas também são separadas em duas subsequências da mesma forma que para geração da primeira sequência. Na primeira subsequência, as tarefas são sequenciadas em ordem decrescente de punições unitárias por atraso divididas pela amplitude de sua janela de tempo e na segunda subsequência são sequenciadas em ordem crescente de punição unitária por adiantamento divididas pela amplitude de sua janela de tempo.

Na terceira sequência, se dividem as tarefas em duas subsequências como para gerar a primeira sequência, porém agora se faz a sequência das tarefas de acordo com sua janela de tempo. Para a primeira subsequência, são atendidas as tarefas em ordem crescente de fim de janela de tempo f_i e em caso de empate escolhem-se as tarefas de menor índice. Para a segunda subsequência, atende-se as tarefas em ordem crescente de início de janela de tempo d_i e, em caso de empate, escolhe-se novamente as tarefas de menor índice.

Na quarta sequência, todas as tarefas são sequenciadas em ordem crescente de punição unitária de adiantamento. Caso esta sequência tenha o valor de função objetivo menor, esta sequência será dividida em duas subsequências, com os mesmos números de elementos que as subsequências da primeira sequência, de forma que se colocando a segunda subsequência após a primeira subsequência se a quarta sequência inteira. Isto para facilitar os procedimentos da Busca Tabu.

Na quinta sequência, todas as tarefas são sequenciadas em ordem decrescente de punição de atraso e, novamente se esta sequência obtiver o menor valor de função objetivo, esta será dividida em duas subsequências com o mesmo procedimento realizado na quarta sequência.

Procedimentos da Busca Tabu

Escolhida uma sequência inicial dentre as cinco sequências descritas acima, aplica-se o procedimento de busca de novas sequências a fim de se diminuir a soma de punições. Estes procedimentos para a Busca Tabu são:

- O procedimento de pesquisa da vizinhança, que na prática é a geração de novas sequências a partir de sequências anteriores, é o de permutação (mudança de posição) de

cada elemento da primeira subsequência com cada elemento da segunda subsequência, um elemento de cada vez, até estar esgotada as trocas possíveis, tomando como novos conjuntos a permutação do par que obteve o menor valor para a função objetivo, colocando este par na lista de movimentos proibidos (lista tabu);

- O critério de parada será o de número de iterações;
- O número de elementos na lista tabu dependerá do número de tarefas do problema;
- O critério de saída da lista tabu será: depois de cheia a lista, os pares de elementos sairão conforme forem entrando outros pares de elementos na lista, já que a lista tem um tamanho definido;
- Também é usada uma matriz de reincidência como sugerido por Glover(1997), para se fazer uma diversificação, quando o resultado da função objetivo não se altere durante um certo número de iterações; (obs.: foi tentada uma intensificação com o auxílio da matriz de reincidência, mas não houve melhoria nos resultados em comparação com a não utilização desta intensificação e diversificação e os resultados foram um pouco pior em comparação com a utilização da diversificação)
- O critério de aspiração será o de melhorar o valor da função objetivo mostrada na expressão (4.1).

Variando $0 \leq TE \leq G$

$$Min_{0 \leq TE \leq G} \left\{ Min \sum_{i=1}^N [a_i (d_i - (ct_i + TE))^+ + b_i ((ct_i + TE) - f_i)^+] \right\} \quad (4.1)$$

Onde:

- TE : instante de liberação comum a todas as tarefas (variável de decisão para minimização da soma de punições - tempo de espera);
- G : constante calculada para a determinação do limite superior do tempo de espera TE ;
- i : índice para as tarefas;
- f_i : fim (limite superior) da janela de tempo da tarefa i ;
- ct_i : instante de término ou conclusão da tarefa i (considerando TE igual a zero);
- N : número de tarefas;
- a_i : punição unitária para a tarefa i por término anterior ao instante de abertura da janela de tempo (adiantamento);

b_i : punição unitária para tarefa a i por término posterior ao instante de encerramento da janela de tempo (atraso);

O sinal positivo do lado esquerdo dos parênteses significa que só os valores positivos serão considerados.

Como já visto no capítulo 3, foi introduzida uma variável de espera chamada tempo de espera (TE), este tempo de espera será determinado pela própria heurística no processo de minimização e ele desloca o início do atendimento da sequência de tarefas; assim, para o valor efetivo de tempo de conclusão das tarefas é necessário somá-lo na equação 4.1. Para restringir o espaço de busca, foi determinado um limite superior G para TE , que é calculado de maneira que se garanta que a partir desse valor não haja nenhuma punição por adiantamento, que seria a situação limite para a minimização. Como a partir deste instante, a função objetivo somente crescerá em função das punições por atraso o melhor instante de liberação que minimize as punições estaria neste limite ou abaixo. Assim, TE pode variar de zero até o limite superior de tempo de espera (G) e este limite pode ser definido da seguinte forma, como mostra a equação 4.2.

$$G = [Max(d_1, \dots, d_N) - Min(\sum_{m=1}^M p_{1,m}, \dots, \sum_{m=1}^M p_{N,m})] \quad (4.2)$$

Onde:

$Min(\sum_{m=1}^M p_{1,m}, \dots, \sum_{m=1}^M p_{N,m})$ é o menor instante possível de término entre as N tarefas.

Se, eventualmente $Max(d_1, \dots, d_N) \leq Min(\sum_{m=1}^M p_{1,m}, \dots, \sum_{m=1}^M p_{N,m})$, G terá valor nulo.

Exemplo de aplicação

O problema que será resolvido nesta seção utilizará os dados de entrada mostrados nas tabelas de dados 4.1, 4.2 e 4.3. Os parâmetros utilizados serão: o número de iterações será 5; o número de elementos da lista tabu será 3; a matriz de reincidência para a diversificação será utilizada quando não houver melhora no resultado durante 2 iterações.

Tabela 4.1 – Parâmetros a_i, b_i, d_i, f_i

Tarefa i	1	2	3	4	5
a_i	3	4	2	3	2
b_i	1	2	3	3	4
d_i	9	5	6	5	10
f_i	11	6	7	6	12

Tabela 4.2 – Roteiro de cada tarefa

Tarefas	Máquinas		
	1	2	3
1	X	X	X
2		X	X
3		X	
4	X	X	
5	X		X

Tabela 4.3 – Tempo de processamento de cada tarefa i em cada máquina m - p_{im}

Tarefa i	Máquinas		
	1	2	3
1	2	3	1
2	0	2	2
3	0	3	0
4	3	2	0
5	3	0	2

Sequência inicial:

Serão construídas cinco sequências. Para a primeira se constroem duas subsequências, na primeira subsequência as punições unitárias por atrasos são maiores que as de adiantamento, portanto as tarefas desta subsequência seriam: subsequência 1 = 3; 4; 5. Na segunda subsequência as punições unitárias por adiantamentos são maiores, portanto: subsequência 2 = 1; 2.

Depois de ordenadas, a primeira subsequência em ordem decrescente e a segunda em ordem crescente, as subsequências ficariam: subsequência 1=5;3;4 e subsequência 2 =1; 2, portanto, a primeira sequência a ser analisada será 5; 3; 4; 1; 2.

O cálculo de seus instantes de término (datas de entregas) dará: $C_1=12.00$; $C_2=7.00$; $C_3=3.00$; $C_4=8.00$; $C_5=5.00$. O cálculo de sua função objetivo dará: $FO=25.00$ com $TE=0$ e variando o TE não houve melhora.

Para a segunda sequência, as subsequências geradas para a primeira sequência foram sequenciadas, levando em conta agora uma divisão pela amplitude das janelas de tempo de cada tarefa, que são (2, 1, 1, 1, 2), ficando as seguintes subsequências : subsequência 1=3; 4; 5 e subsequência 2=1; 2. A segunda sequência inteira será 3; 4; 5; 1; 2, com instantes de término $C_1=12.00$; $C_2=10.00$; $C_3=3.00$; $C_4=5.00$; $C_5=8.00$, com melhor valor de função objetivo (soma de punições) $FO=18$ para uma variação do instante de liberação (TE) de 1 unidade de tempo.

A terceira sequência ordena as mesmas subsequências da primeira sequência, porém agora a primeira subsequência é ordenada em ordem crescente de fim de janela de tempo f_i e a segunda subsequência em ordem crescente de início de janela de tempo d_i . Desta forma a sequência ficou a seguinte: 4; 3; 5; 2; 1 com os mesmos instantes de término da segunda sequência, logo com mesmo valor para a função objetivo.

Para a quarta sequência as tarefas foram sequenciadas em ordem crescente de punição unitária de adiantamento, e a sequência resultante foi: 3; 5; 1; 4; 2 com instante de término de cada tarefa $C_1=9.00$; $C_2=7.00$; $C_3=3.00$; $C_4=10.00$; $C_5=5.00$. Assim, o melhor resultado da soma de punições foi $FO=30.00$ sem variação de TE , portanto $TE=0$. A divisão em subsequências ficou: subsequência 1=3; 5; 1 e subsequência 2=4; 2.

Para a quinta sequência as tarefas foram ordenadas em ordem decrescente de punição unitária de atraso, resultando na seguinte sequência: 5; 3; 4; 2; 1. com instantes de término de cada tarefa iguais às da primeira sequência, assim sendo, o resultado da soma das punições também foi $FO=25.00$, com instante de liberação $TE=0$ também. A divisão em subsequências ficou: subsequência 1=5; 3; 4 e subsequência 2=2; 1.

Deste modo, a sequência admitida pela heurística para aplicação da Busca Tabu foi a segunda sequência gerada 3; 4; 5; 1; 2 com $FO=18$ e $TE=1$.

Aplicando a Busca Tabu

(Com a sequência inicial aplica-se o método de busca, que será a permutação dos elementos da primeira subsequência com os elementos da segunda subsequência).

A sequência inicial fica sendo 3; 4; 5; 1; 2 com a primeira subsequência 3; 4; 5 e segunda subsequência 1; 2.

Sequência	Função Objetivo	Espera
1;4;5;3;2;	30	0
2;4;5;1;3;	10	1
3;1;5;4;2;	33	1
3;2;5;1;4;	30	0
3;4;1;5;2;	20	0
3;4;2;1;5;	13	0

(O movimento que deu o melhor resultado foi a troca da tarefa 2 da segunda subsequência com a tarefa 3 da primeira, portanto este será o primeiro movimento a entrar na lista tabu e a nova sequência será utilizada na nova busca)

Nova sequência: 2;4;5;1;3 com $FO = 10$ e espera de 1 unidade de tempo

Lista tabu:[(3,2)(0,0)(0,0)]

Número de iterações sem melhoria: 0

Nova busca:

A sequência, para a busca, agora fica sendo 2; 4; 5; 1; 3 com a primeira subsequência 2; 4; 5 e a segunda subsequência 1; 3.

Sequência	Função Objetivo	Espera	
1;4;5;2;3;	24	1	
3;4;5;1;2;	18	1	(este é um exemplo de movimento não aceito por estar na lista tabu)
2;1;5;4;3;	29	1	
2;3;5;1;4;	21	1	
2;4;1;5;3;	7	1	
2;4;3;1;5;	7	1	

(O melhor resultado desta iteração é o conseguido com o movimento de troca das tarefas 5 da primeira subsequência com a tarefa 1 da segunda subsequência, que será o segundo movimento tabu. Nesta iteração houve melhora no resultado)

Nova sequência: 2;4;1;5;3 com $FO=7.00$ e espera de 1 unidade de tempo

Lista tabu:[(2,5)(5,1)(0,0)]

Número de iterações sem melhora: 0

Nova busca:

A sequência, para a busca, agora fica sendo 2; 4; 5; 1; 3 com a primeira subsequência 2; 4; 1 e a segunda subsequência 5; 3.

Sequência	Função Objetivo	Espera
5;4;1;2;3;	17	1
3;4;1;5;2;	20	0
2;5;1;4;3;	21	1
2;3;1;5;4;	19	1
2;4;5;1;3;	10	1
2;4;3;5;1;	10	1

(O melhor resultado desta iteração é o conseguido com o movimento de troca das tarefas 1 da primeira subsequência com a tarefa 3 da segunda subsequência, que será o segundo movimento tabu, pois o movimento das tarefas 5 e 1 é tabu. Nesta iteração não houve melhora no resultado)

Nova sequência: 2; 4; 3; 5; 1 com $FO = 10.00$ e espera de 1 unidade de tempo pois a outra sequência com $FO=10$ o movimento é tabu.

Lista tabu:[(2,5)(5,1)(1,3)]

Número de iterações sem melhora: 1

Nova busca:

A sequência, para a busca, agora fica sendo 2; 4; 3; 5; 1 com a primeira subsequência 2; 4; 3 e a segunda subsequência 5; 1.

Sequência	Função Objetivo	Espera
5;4;3;2;1;	25	0
1;4;3;5;2;	30	0
2;5;3;4;1;	17	1
2;1;3;5;4;	29	1
2;4;5;3;1;	10	1
2;4;1;5;3;	7	1

(Nesta iteração o primeiro par de tarefas (2,5) da lista tabu sai para dar lugar a outro par (4,5))

Nova sequência: 2; 5; 3; 4; 1 com $FO = 17.00$ e espera de 1 unidade de tempo. Os outros movimentos são tabus.

Lista tabu: [(4,5)(5,3)(1,3)]

Número de iterações sem melhoria: 2 (Não houve melhoria em 2 iterações consecutivas, portanto com o uso da matriz de reincidência será definida uma nova sequência. Esta nova sequência considerará as tarefas nas posições em que elas estiveram o menor número de vezes. A lista tabu será zerada)

Matriz de reincidência

Tabela 4.4 - Matriz de reincidência

	Posições				
Tarefa	1	2	3	4	5
1	3	3	7	7	4
2	13	1	1	3	6
3	6	2	6	2	8
4	0	16	0	4	5
5	2	2	10	8	2

O procedimento de construção da nova sequência, a partir da matriz de reincidência, vista na tabela 4.4, foi: a partir da primeira posição, a tarefa que a ocupou o menor número de vezes foi a tarefa 4; para a posição 2 a tarefa que a ocupou o menor número de vezes foi a tarefa 2; para a posição 3 a tarefa 2 seria a tarefa com menor ocupação, porém ela já ocupará a posição 2, assim a próxima tarefa é a 3; para a posição 4 as tarefas 3, 2 e 4 já ocupam outras posições, portanto nesta posição ficará a tarefa 1; e finalmente a posição 5 será ocupada pela tarefa 5, pois além de ter a menor ocupação nesta posição também é última tarefa restante.

Nova sequência: 4;2;3;1;5 com $FO = 14.00$ e espera de 1 unidade de tempo

Nova busca:

A sequência, para a busca, agora fica sendo 4; 2; 3; 1; 5 com a primeira subsequência 4; 2; 3 e a segunda subsequência 1; 5.

Sequência	Função Objetivo	Espera
1;2;3;4;5;	27	1
5;2;3;1;4;	21	1
4;1;3;2;5;	20	0
4;5;3;1;2;	18	1
4;2;1;3;5;	7	1
4;2;5;1;3;	10	1

Nova sequência: 4; 2; 1; 3; 5 com $FO = 7.00$ e espera de 1 unidade de tempo

Lista tabu: [(3,1)(0,0)(0,0)]

(As cinco iterações terminaram, tendo como resultado a sequência: 2;4;1;5;3; com $FO=7.00$ e espera de 1 unidade de tempo. O valor 7.00 é o valor que otimiza o problema, considerando a estratégia adotada e este valor aparece em dez sequências.)

4.4 – Busca Exaustiva

A heurística que utilizou a enumeração explícita (Busca Exaustiva) como método de resolução foi chamado de E/TBE e o procedimento de busca exaustiva foi por meio de um processo de ramificação, tal como uma árvore, como foi visto no capítulo 2.

Este procedimento garante que serão conseguidas as sequências ótimas do problema, considerando as simplificações do modelo, mas não garante o ótimo para o problema geral, em que pode haver sequências diferentes em cada máquina, por isso foi designado de heurística.

A árvore para a determinação de todas as sequências neste estudo terá as características descritas a seguir, exemplificadas na figura 4.1.

Figura 4.1 – Arvore de grau 3 com 4 níveis, ramificando todas as sequências possíveis para o estudo do exemplo de três tarefas.

- Será considerado que o nível 1 terá um nó fictício e ligado a ele aparecem todas as tarefas como nós. Na figura 4.2, o nível 1 é o nó 0;
- O número de níveis desta árvore será o número de tarefas mais um, já que há um nó fictício;
- Cada nó será uma raiz de um nó de um nível acima quando estes nós estiverem ligados. Na figura 4.2, no nível 4, o nó mais a esquerda ocupado pela tarefa 3 é raiz do nó ocupado pela tarefa 2 no nível 3 e eles estão ligados por uma seta ;
- Ramificação será a ligação dos nós, no caso da figura 4.2 são as setas;
- Ligados a cada nó, em um dado nível, por meio de ramos estarão tarefas que não foram raiz em uma mesma ramificação nos níveis acima;
- O número de raízes ligadas diretamente a um nó, em um certo nível, será o número de tarefas menos as tarefas contempladas nos níveis acima em uma mesma ramificação menos um. No caso da figura 4.2, na ramificação mais à esquerda (ou seja, o nó fictício 0 e os nós ocupados pelas tarefas 1, 2, 3) o nó ocupado pela tarefa 2 terá uma raiz, que é o nó ocupado pela tarefa 3, pois o nó ocupado pela tarefa 1 já foi contemplado na mesma ramificação em nível acima.

Esta árvore não será criada inteira na implementação da heurística e sim será criada cada sequência para estudo com informações desta árvore. Assim, o procedimento será: a partir do primeiro nó com tarefa, no nível 2, segue-se construindo a sequência com as tarefas que fazem parte do mesmo ramo até o último nível. Quando se atinge o último nível se tem a sequência completa. Neste ponto se pode determinar o instante de liberação da sequência (TE), os instantes de término de todas as tarefas (ct_i) e a soma de punições por atraso e adiantamento (FO) a partir da diferença dos instantes de término e os instantes de início de janela de tempo e fim de janela de tempo, de acordo com a expressão (4.1).

Depois de se criar a primeira sequência, ou seja, sequência de tarefas que ocupam os nós da ramificação mais à esquerda, em ordem crescente de nível, a partir do nível 2 (1,2,3). As próximas sequências a serem criadas, seguem o seguinte procedimento: se atualizam todas as raízes dos níveis mais baixos, daí passando para os níveis mais altos de uma mesma ramificação, voltando a atualizar as raízes dos níveis abaixo, até que se tenha passado por todas as

ramificações. O primeiro valor da função objetivo, ou seja, soma das punições, é guardado e vai sendo atualizado a medida que vai se encontrando valores menores. Também são guardadas todas as sequências que originam o valor de soma mais atualizado e se descartam as outras.

Este procedimento poderá ser compreendido melhor no exemplo de aplicação desta heurística que será visto a seguir.

Exemplo de Aplicação

Utilizando-se a árvore da figura 4.1 como exemplo, o procedimento de se determinar as sequências, para análise dos valores da função objetivo, é dado a seguir.

1. Segue-se o ramo mais à esquerda até o nível mais baixo, considerando a sequência a partir do nível dois. No exemplo considerado esta sequência seria: 1, 2 e 3;
2. Faz-se o cálculo do instante de liberação, dos instantes de término das tarefas e o cálculo da soma das punições, tomando como dados de entradas os valores dados nas tabelas 4.2, 4.3 e 4.4;
3. Guardam-se o valor da função objetivo e a sequência; nas iterações seguintes, serão atualizados o valor da função objetivo e a sequência, sempre que se encontrar uma função objetivo com valor inferior ao valor guardado;
4. Para a próxima sequência, verifica-se se há no último nível, que no caso do exemplo é o quarto nível, mais tarefas ligadas à mesma raiz do nível acima. Havendo, troca-se a tarefa; não havendo, passa-se para o nível acima e procede-se da mesma maneira, ou seja, havendo tarefas que não estejam nos nós acima, troca-se pela primeira tarefa que estará em ordem crescente de índice de tarefa e não havendo mais tarefas passa-se para o nível acima. A cada sequência diferente calculam-se os instantes de término e a soma das punições para comparação.

Deste modo, usando os mesmos dados do exemplo anterior, as sequências seriam determinadas da seguinte maneira no exemplo:

Primeira Sequência

– segue o primeiro ramo até o nível mais baixo começando a sequência no nível 2 ficando 1, 2 e 3.

- Os instantes de término seriam: $C_1=6.00$; $C_2=4.00$; $C_3=8.00$;

O melhor valor da função objetivo (Soma das punições) se dá com o instante de liberação $TE=1$ e este valor é $FO=12$;

Segunda Sequência

- Como no último nível (nível 4) só há uma raiz ligada na mesma ramificação, troca-se o nó do nível 3 ocupado pela tarefa 2 pelo nó ocupado pela tarefa 3, que está na mesma ramificação do nó ocupado pela tarefa 1 e na mesma ramificação, no nível 4, agora se tem a tarefa 2. Assim, a nova sequência fica sendo 1, 3 e 2. Os novos instantes de término seriam agora: $C_1=7.00$; $C_2=10.00$; $C_3=3.00$. Com uma variação de uma unidade de tempo para o instante de liberação, ou seja, $TE=1$, obtêm-se o melhor valor de função objetivo, $FO=14$ para os instantes de término encontradas.

Terceira Sequência

- Novamente no nível 4 só há a tarefa 2 e, no nível 3, também não há mais tarefas, portanto, troca-se o nó do nível 2 ocupado pela tarefa 1 pelo nó ocupado pela tarefa 2, que está na mesma ramificação do nó fictício. Depois na mesma ramificação troca-se os nós dos níveis abaixo. Assim, a terceira sequência fica sendo: 2, 1 e 3 e os respectivos instantes de término são $C_1=6.00$; $C_2=4.00$; $C_3=8.00$ e portanto, $TE=1$ e $FO=12$.

Quarta Sequência

- Repete-se o procedimento acima e consegue-se a sequência: 2, 3 e 1.

- $C_1=9.00$; $C_2=4.00$; $C_3=5.00$; $TE=1$; $FO=0$.

Quinta Sequência

- Repete-se novamente o procedimento e consegue-se a sequência: 3, 1 e 2.

- $C_1=7.00$; $C_2=10.00$; $C_3=3.00$; $TE=2$; $FO=14$.

Sexta Sequência

- Repete-se o procedimento e consegue-se a sequência: 3, 2 e 1.
- $C_1=9.00$; $C_2=7.00$; $C_3=3.00$; $TE=0$; $FO=8$.

Continuando o procedimento, não haverá mais tarefas no nível dois, portanto sobe-se ao nível 1 onde temos uma raiz que não é uma tarefa e temos o final do procedimento, tendo assim seis sequências possíveis para o estudo. Deste modo pode-se dizer com segurança que o valor mínimo de soma de punições é $FO=0$ e $TE=1$, sendo que a sequência que origina este valor é: 2, 3 e 1.

Na heurística proposta, além de se guardar o menor valor da soma de punições, também serão guardadas todas as sequências que geram este valor. Esta característica é uma vantagem deste método de enumeração, mas não foi observado neste exemplo de aplicação.

4.5 – Heurística Construtiva

A terceira heurística, que usou uma técnica construtiva (sem vínculo com a heurística construtiva do E/TTabu) como método de resolução foi chamado E/THC. Uma heurística construtiva também foi usada internamente na heurística E/TTabu, para se encontrar uma solução inicial, porém as simplificações eram diferentes. As diferenças aqui são: será gerado para cada máquina um conjunto de 10 séries de sequências de atendimento efetivo de tarefas e não prioridades, a partir de critérios que serão vistos mais adiante; deste modo cada tarefa será atendida por todas máquinas que estiverem no seu roteiro. A série de sequências que minimiza a somatória de penalidades será escolhida deste conjunto de 10 séries geradas. Assim, como as tarefas podem pular máquinas no atendimento, cada máquina poderá ter uma sequência de atendimento diferente e poderá haver tempo ocioso, ou seja, a máquina poderá ficar esperando a próxima tarefa da sequência, mesmo tendo uma outra tarefa em sua fila pronta para atendimento.

Sequências Geradas

As sequências geradas seguirão os seguintes critérios:

- A primeira série de sequências geradas, em cada máquina, considera a união de duas subsequências, uma colocada após a outra. Uma primeira subsequência de tarefas onde as punições unitárias por atraso são maiores que as punições unitárias de adiantamento e uma segunda onde as punições unitárias por adiantamento são maiores. Em caso de empate, a tarefa ocupará a subsequência com menor número de elementos e novamente se houver empate neste número a tarefa ocupará o primeiro conjunto. Na primeira subsequência, as tarefas são sequenciadas em ordem decrescente de punições unitária por atraso e na segunda são sequenciadas em ordem crescente de punições por adiantamento. Se todas as punições unitárias por atraso forem maiores que as de adiantamentos ou vice-versa, ainda assim haverá a divisão em duas subsequências, porém primeiro se faz à sequência ou em ordem decrescente de punição unitária por atraso, caso estas sejam maiores, ou em ordem crescente de punição por adiantamento, caso estas sejam maiores. Depois se divide as tarefas em duas subsequências com o mesmo número de elementos, no caso do número de tarefas for par, ou, caso contrário, com o primeiro conjunto com um elemento a mais;
- A segunda série de sequências é gerada praticamente igual à primeira série, com a diferença de que, na ordenação, as punições são divididas pelos tempos de processamento das tarefas nas máquinas;
- Na terceira série de sequências, todas as tarefas são sequenciadas em ordem decrescente de punição unitária por atraso;
- Na quarta série de sequências, todas as tarefas são sequenciadas em ordem crescente de punição por adiantamento.
- A quinta série de sequências é igual à terceira, porém agora as punições são divididas pelo tempo de processamento da tarefa da máquina em que está sendo gerada a sequência, já que cada máquina terá uma sequência própria;
- A sexta série de sequências é igual à quarta, porém agora as punições são divididas pelo tempo de processamento da tarefa na máquina como quinta série de sequências;

- A sétima série de sequências é igual à segunda, porém agora além de dividir as punições das tarefas pelo seu tempo de processamento, também se divide pela sua amplitude de janela de tempo;
- Na oitava série de sequências, se determinam os pontos médios das janelas de tempo de cada tarefa e um valor médio de instante de entrega, que seria o valor médio dos pontos médios das janelas de tempo. Com estes valores, se constroem duas subsequências. Uma primeira com os pontos médios de janelas de tempo menores que o instante de entrega médio e um segundo com estes pontos médios maiores. Em caso de empate, se procede como nas primeiras sequências. A ordenação das tarefas também será igual à ordenação da primeira série de sequências;
- Para a nona série de sequências se dividem as tarefas em duas subsequências, como na primeira sequência, porém agora se ordenam as tarefas de acordo com sua janela de tempo, ou seja, para a primeira subsequência, as tarefas são atendidas por ordem crescente de fim de janela de tempo f_i e em caso de empate escolhem-se as tarefas de menor índice. Para a segunda subsequência as tarefas são atendidas por ordem crescente do instante do início de janela de tempo d_i e em caso de empate escolhem-se, novamente, as tarefas de menor índice;
- A décima série de sequências é praticamente igual à sétima, porém agora não se dividem as punições unitárias pelos tempos de processamento das tarefas da máquina em questão.

Para cada uma dessas sequências, também se determina o instante de liberação e, dentre essas dez sequências, se escolhe a que resulte no menor valor da função objetivo (soma das punições por atraso e adiantamento).

Para problemas de máquina única, com punições por atraso e adiantamento, as regras mais eficazes utilizadas, segundo Baker e Scuder (1989), são as regras que separam as punições por atraso e adiantamento e as dividem pelo tempo de processamento. Neste trabalho, além deste tipo de estratégia, foram geradas sequências com regras para problemas específicos, como minimização de atrasos totais e minimização de adiantamentos totais, vistos na terceira, quarta, quinta e sexta série de sequências geradas.

Foram geradas cinco sequências iniciais na heurística E/TTabu e dez sequências na heurística E/THC a partir de métodos construtivos utilizando regras diferentes, pois se pode dizer que as diferentes regras utilizadas afetam, de modo significativo, as medidas de desempenho, sem que seja possível se dizer que alguma delas é superior devido a diferentes parâmetros e hipóteses de cada problema.

Exemplo de Aplicação

Serão geradas dez séries de três sequências, uma sequência efetiva de atendimento para cada máquina. Para cada sequência será levada em conta uma estratégia de ordenação das tarefas, como já foi descrito anteriormente. O exemplo de aplicação será o mesmo que o da Busca Tabu e Busca Exaustiva.

Para a primeira série de sequências, cada máquina considera a união de duas subsequências. Uma primeira subsequência de tarefas onde as punições unitárias por atraso são maiores que as punições unitárias por adiantamento e uma segunda onde as punições por adiantamento são maiores. Na primeira subsequência, as tarefas são sequenciadas em ordem decrescente de punições unitárias por atraso e na segunda são sequenciadas em ordem crescente de punições unitárias por adiantamento. Assim, as sequências em cada máquina, seus instantes de término, o valor da função objetivo e a correspondente instante liberação seriam:

Máquina 1 => 5;4;1; Máquina 2 => 3;4;1;2; Máquina 3 => 5;1;2;
 Instante de término das tarefas: $C_1=12.00$; $C_2=15.00$; $C_3=3.00$; $C_4=8.00$; $C_5=5.00$
 Instante de liberação: $TE = 0$ Valor da função objetivo: $FO = 41$

A segunda série de sequências é gerada praticamente igual a primeira série, com a diferença de que na ordenação as punições são divididas pelos tempos de processamento das tarefas nas máquinas. Assim, o que se teria é:

Máquina 1 => 5;4;1; Máquina 2 => 4;3;1;2; Máquina 3 => 5;2;1;
 Instante de término das tarefas: $C_1=19.00$; $C_2=18.00$; $C_3=11.00$; $C_4=8.00$; $C_5=5.00$
 Instante de liberação: $TE = 0$ Valor da função objetivo: $FO = 60$

Na terceira série de sequências, todas as tarefas são sequenciadas em ordem decrescente de punição de atraso.

Máquina 1 => 5;4;1; Máquina 2 => 3;4;2;1; Máquina 3 => 5;2;1;
 Instante de término das tarefas: $C_1=14.00$; $C_2=12.00$; $C_3=3.00$; $C_4=8.00$; $C_5=5.00$
 Instante de liberação: $TE = 0$ Valor da função objetivo: $FO = 37$

Na quarta série de sequências todas as tarefas são ordenadas em ordem crescente de punição de atraso.

Máquina 1 => 5;4;1; Máquina 2 => 3;4;1;2; Máquina 3 => 5;1;2;
 Instante de término das tarefas: $C_1=12.00$; $C_2=15.00$; $C_3=3.00$; $C_4=8.00$; $C_5=5.00$
 Instante de liberação: $TE = 0$ Valor da função objetivo: $FO = 41$

A quinta série de sequências é igual à terceira, porém agora as punições são divididas pelo tempo de processamento da tarefa na máquina.

Máquina 1 => 5;4;1; Máquina 2 => 4;2;3;1; Máquina 3 => 5;1;2;
 Instantes de término das tarefas: $C_1=17.00$; $C_2=19.00$; $C_3=13.00$; $C_4=8.00$; $C_5=5.00$
 Instante de liberação: $TE = 0$ Valor da função objetivo: $FO = 66$

A sexta série de sequências é igual à quarta, porém agora as punições são divididas pelo tempo de processamento da tarefa na máquina.

Máquina 1 => 5;4;1; Máquina 2 => 3;1;4;2; Máquina 3 => 5;2;1;
 Instantes de término das tarefas: $C_1=18.00$; $C_2=17.00$; $C_3=3.00$; $C_4=13.00$; $C_5=5.00$
 Instante de liberação: $TE = 0$ Valor da função objetivo: $FO = 66$

A sétima série de sequências é igual à segunda, porém agora além de dividir as punições das tarefas pelo seu tempo de processamento, também se dividi pelo seu intervalo de janela de tempo.

Máquina 1 => 4;5;1; Máquina 2 => 3;4;2;1; Máquina 3 => 5;2;1;
 Instantes de término das tarefas: $C_1=12.00$; $C_2=10.00$; $C_3=3.00$; $C_4=5.00$; $C_5=8.00$
 Instante de liberação: $TE = 1$ Valor da função objetivo: $FO = 18$

Na oitava série de sequências, se determinam os pontos médios das janelas de tempo de cada tarefa e um valor médio de instante de entrega. Com estes valores se constrói dois

conjuntos: um primeiro com os pontos médios de janelas de tempo menores que o instante de entrega médio e um segundo com estes pontos médios maiores. A ordenação das tarefas também será igual à ordenação da primeira série de sequências.

Máquina 1 $\Rightarrow$ 4;5;1; Máquina 2 $\Rightarrow$ 4;2;3;1; Máquina 3 $\Rightarrow$ 2;5;1;

Instante de término das tarefas: $C_1=14.00$; $C_2=9.00$; $C_3=10.00$; $C_4=5.00$; $C_5=11.00$

Instante de liberação: $TE = 0$

Valor da função objetivo: $FO = 18$

Para a nona série de sequências se dividi as tarefas em dois conjuntos e se ordenam às tarefas de acordo com sua janela de tempo. Para o primeiro conjunto atende-se primeiro as tarefas com fim de janela de tempo f_i menores. Para o segundo conjunto atende-se primeiro as tarefas com início de janela de tempo d_i menores.

Máquina 1 $\Rightarrow$ 5;4;1; Máquina 2 $\Rightarrow$ 3;4;1;2; Máquina 3 $\Rightarrow$ 5;1;2;

Instante de término das tarefas: $C_1=12.00$; $C_2=15.00$; $C_3=3.00$; $C_4=8.00$; $C_5=5.00$

Instante de liberação: $TE = 0$

Valor da função objetivo: $FO = 41$

A décima série de sequências é praticamente igual à sétima, sem a divisão das punições pelos tempos de processamento.

Máquina 1 $\Rightarrow$ 5;4;1; Máquina 2 $\Rightarrow$ 4;3;1;2; Máquina 3 $\Rightarrow$ 5;1;2;

Instante de término das tarefas: $C_1=15.00$; $C_2=18.00$; $C_3=11.00$; $C_4=8.00$; $C_5=5.00$

Instante de liberação: $TE = 0$

Valor da função objetivo: $FO = 56$

Foram gerados duas séries de sequências com valor de soma de punições igual a 18, mas as sequências que a heurística irá adotar será as que foram geradas primeiras, portanto a sétima série de sequência. O tempo de processamento foi tão baixo que não pôde ser medido em “milisegundos”.

CAPÍTULO 5

RESULTADOS

Neste capítulo serão apresentados os resultados obtidos, a partir de uma série de problemas propostos, tais que permitam avaliar os modelos e heurísticas propostas.

5.1 – Introdução

A função objetivo e as restrições do modelo matemático, apresentado no capítulo 3 foram estruturadas em ambiente VBA (Visual Basic for Applications) em uma planilha Excel da Microsoft. O modelo matemático foi executado em um pacote de software, com reconhecida eficiência, conhecido como “CPLEX” versão 10.0 e esta estratégia foi chamada E/TCPLEX. As heurísticas E/TTabu, e E/TBE que resolvem, de forma aproximada e exata, respectivamente, uma versão simplificada do problema real e a heurística E/THC, que resolve de forma aproximada o problema real, foram implementados em ambiente Delphi-5 e o seu processamento foi feito em um microcomputador (notebook) com processador Intel com velocidade 1,86 Ghz e com memória RAM de 512Kb.

Para a heurística E/TBE, de busca exaustiva que analisa todas as sequências, o tempo de processamento vai variar na mesma razão em que cresce o número de sequências que serão analisadas, assim, para problemas com 9, 10 e 11 tarefas e 3 máquinas, o número sequências seriam respectivamente 362.880, 3.628.800 e 39.916.800 e os tempos de processamento, utilizando o microcomputador descrito acima, foram respectivamente 42 segundos, 435 segundos e 5.561 segundos.

Para a heurística E/TTabu, o tempo avaliar cada solução (sequência de prioridade) é igual à da heurística E/TBE, porém o método Busca Tabu usa uma estratégia de busca na qual pode-se analisar um número bem menor de sequências para se obter soluções de boa qualidade. Os tempos de processamento da heurística E/TTabu dependem do número de iterações que será escolhido.

Para a heurística E/THC o tempo de processamento será muito pequeno, já que ele só analisa as 10 seqüências propostas para construção.

Para o E/TCPLEX os tempos de processamento não são uniformes como no caso do E/TBE, ou seja, podem variar muito, como pode ser visto na tabela 5.1, que mostra, para uma pequena série de problemas propostos, os tempos de processamento para se encontrar soluções onde a diferença entre estas soluções e o limite inferior é menor que 1%, deste modo o algoritmo E/TCPLEX assume a solução como ótima.

Tabela 5.1- Tempos de processamentos (em segundos) do E/TCPLEX

	n=7	n=8	n=9	n=10
m=2	<0,01	<0,01	431,32	>43000
m=3	0,02	12,74	2023,57	>43000
m=4	0,82	27,07	<1	>43000
m=5	0,01	5031,65	21934,21	>43000

Onde: n é o número de tarefas; m é o número de máquinas.

Como já era previsto, esta modelagem só é eficiente em termos de tempo de processamento para problemas pequenos, de até 9 tarefas e até 3 ou 4 máquinas. Porém, dependendo dos dados do problema, é possível se obter soluções ótimas rapidamente, como no caso do problema com 9 tarefas e 4 máquinas, que chegou ao resultado zero em 1 segundo e isto pode acontecer com problemas com mais tarefas.

Os problemas propostos neste trabalho foram gerados tendo como semente os dados propostos por Feldmann e Biskup (2003), para problemas de atraso e adiantamento com uma única máquina e data comum de entrega. Feldmann e Biskup (2003) resolveram o problema utilizando três heurísticas: pesquisa evolucionara (*evolutionary search*), recozimento simulado (*simulate annealing*) e (*Threshold accepting*), utilizando como regra a propriedade forma V (*V-shaped*), característica de problemas com uma única máquina e data de entrega comum. Com estas heurísticas Feldmann e Biskup (2003) apresentaram 280 *benchmarks*, que eram encontrados no endereço (site) da Internet <http://mscmga.ms.ic.ac.uk/info.html> (que a partir de 2007 não estava mais disponível). A data comum de entrega era determinada por Feldmann e Biskup (2003) a partir da somatória dos tempos de processamento. Esta data, juntamente com os tempos de processamento das tarefas na única máquina, foram utilizados como semente (ponto de partida), para que, por meio de um processo aleatório, fossem determinados: as janelas de tempo para cada tarefa, os tempos de processamento de cada tarefa em cada máquina. As punições por

adiantamento e atraso foram utilizadas sem alteração. Finalmente, os roteiros de cada tarefa foram também determinadas em um processo aleatório.

5.2 – Resultados de Dois Problemas Exemplos com 10 e 30 Tarefas

Primeiramente, foram gerados dois problemas exemplos para se verificar a eficiência do algoritmo e heurísticas propostas; os dados de entrada destes problemas são apresentados nas tabelas 5.2 e 5.3. O primeiro problema, com 10 tarefas a serem processadas em 3 máquinas, foi resolvido pelas quatro propostas e o outro, com 30 tarefas a serem processadas em 5 máquinas, somente não foi resolvido pelo E/TBE devido ao tempo de processamento, que seria excessivamente alto e também porque este tipo de heurística não é eficiente em uma busca parcial, ou seja, parando antes do fim da busca. Após a apresentação dos resultados destes dois primeiros problemas, é apresentada, na seção 5.3, mais uma série de problemas para avaliação da eficiência das quatro propostas.

As tabelas de dados 5.2 e 5.3 dos dois primeiros problemas propostos mostram os tempos de processamento de cada tarefa i em cada máquina m ($p_{i,m}$), o roteiro de cada tarefa, ou seja, a sequência de máquinas pelas quais a tarefa deve passar; as punições por atrasos e adiantamentos (a_i e b_i); e os instantes de início e fim da janela de tempo de cada tarefa (d_i e f_i).

Tabela 5.2 - Valores de $p(i,j)$, $a(i)$, $b(i)$, $d(i)$, $f(i)$ e roteiro de cada tarefa i - Problema com 10 tarefas e atendidas por 3 máquinas

Tar(i)	$p_{i,1}$	$p_{i,2}$	$p_{i,3}$	Roteiro	a_i	b_i	d_i	f_i
1	6	11		1, 2	10	5	65	80
2	3	6	4	1, 2, 3	3	13	70	83
3	2		2	1, 3	9	7	90	100
4	4	6		1, 2	6	4	75	90
5			11	3	10	1	65	82
6	4		5	1, 3	6	15	68	84
7	3	4	2	1, 2 3	3	2	71	88
8	15		13	1, 3	6	13	67	80
9		20		2	7	10	69	80
10	5	5		1, 2	5	6	85	100

Tabela 5.3 – Valores de $p_{i,m}, a_i, b_i, d_i, f_i$ e roteiro de cada tarefa i - Problema com 30 tarefas atendidas por 5 máquinas

Tar(i)	$p_{i,1}$	$p_{i,2}$	$p_{i,3}$	$p_{i,4}$	$p_{i,5}$	Roteiro	a_i	b_i	d_i	f_i
1			6	9		3, 4	10	5	782	797
2	4					1	3	13	789	805
3	2		2	2	2	1, 3, 4, 5	9	7	783	799
4			5	3	4	3, 4, 5	6	4	786	796
5		14	8	11		2, 3, 4	10	1	782	793
6		6	5			2, 3	6	15	786	807
7	2	4	2	2		1, 2, 3, 4	3	2	789	794
8	15			11		1, 4	6	13	786	805
9	16				16	1, 5	7	10	785	802
10		11	9	9		2, 3, 4	4	14	788	806
11		3				2	6	12	786	804
12	10				10	1, 5	3	4	789	796
13			4			3	7	4	785	796
14			11		11	3, 5	4	10	788	802
15	2	3	2			1, 2, 3	2	13	790	805
16	2	5				1, 2	4	9	788	801
17					11	5	4	5	788	797
18	5	8		6		1, 2, 4	1	7	791	799
19				8		4	5	7	787	799
20	1	1			1	1, 2, 5	8	14	784	806
21					4	5	7	12	785	804
22	10			8	10	1, 4, 5	5	11	787	803
23			15	10	10	3, 4, 5	8	9	784	801
24	6	10	5	7	6	1, 2, 3, 4, 5	4	7	788	799
25	5	9	4	6		1, 2, 3, 4	3	9	789	801
26				5		4	1	6	791	798
27	8	16		13	10	1, 2, 4, 5	9	10	783	802
28		4			2	2, 5	9	8	783	800
29	2		3	3	2	1, 3, 4, 5	2	12	790	804
30		19		10		2, 4	5	12	787	804

Obs.: i = tarefa e m = máquina

5.2.1 – Resultados do Pacote CPLEX – E/TCPLEX

Na tabela 5.4 são mostrados os resultados do exemplo com 10 tarefas utilizando o pacote “CPLEX”. A figura 5.1 mostra como se comportou a função objetivo na resolução do modelo matemático do capítulo 3. A tabela 5.5 apresenta o gráfico de ocupação das máquinas, para o problema com 10 tarefas e 3 máquinas, apresentando os instantes de início e fim de cada tarefa e sequência de atendimento das tarefas em cada máquina.

Tabela 5.4– Resultado do algoritmo E/TCPLEX – Problema com 10 tarefas e 3 máquinas:

Soma total de punições final	:	7;
Sequência em cada máquina	:	máquina 1 => 4, 3, 2, 1, 6, 7, 8, 10;
		máquina 2 => 1, 2, 9, 7, 4, 10;
		máquina 3 => 8, 2, 6, 5, 7, 3;
Tempo de processamento	:	11.330,3 segundos;
Melhor Limitante (Inferior)	:	6,95;
Número de nós da Arvore Branch & Cut	:	346979;
Número de Restrições	:	552;
Número de Variáveis (todas)	:	337;
Número de Variáveis Binárias	:	116;

Tempo de Processamento (segundos)	Função Objetivo	Tempo de Processamento (segundos)	Função Objetivo
0	1328	241,5	117
0,1	823	697,5	89
0,1	208	698	79
0,8	200	698,3	63
1,1	180	698,7	61
2,9	168	699,2	57
14,3	142	700,4	53
19,5	136	702,9	41
36,1	134	703,7	31
42,2	128	708,6	27
71,4	125	742,7	18
97,2	123	776,1	7

Figura 5.1 - Valor da função objetivo & Tempo – Problema com 10 tarefas e 3 máquinas - E/TCPLEX

Tabela 5.5 - Tabela de ocupação das máquinas - E/TCPLEX - Problema com 10 tarefas e 3 máquinas

Maq.1	tar4	tar3	Tar2	Tar1	tar6	Ociosa	tar7	tar8	Tar10
Instantes de Início/fim	0/4	4/6	6/9	9/15	15/19	19/36	36/39	39/54	54/59
Maq2	tar1	tar2	Tar9	Tar7	tar4	tar10			
Instantes de Início/fim	43/54	54/60	60/80	80/84	84/90	90/95			
Maq3	tar8	tar2	tar6	Tar5	tar7	tar3			
Instantes de Início/fim	54/67	67/71	71/76	76/87	87/89	89/91			

A solução ótima deste problema, a partir do E/TCPLEX, foi atingida após um tempo de processamento de 11.330 segundos e o valor da função objetivo é 7; este valor já havia sido obtido com 776,1 segundos de processamento, mas a estimativa de limitante inferior nesse momento ainda era zero (permanecendo zero por mais de uma hora de processamento). O programa só foi assumir que o valor 7 correspondia à solução ótima após a estimativa de limitante inferior chegar ao valor 6,95 (menor que 1% do valor da função objetivo). Lembrando, que de acordo com Mendes (2007), a estimativa de limite inferior é obtida a partir do problema relaxado adicionando-se novas restrições usando-se uma estratégia de reformulação.

Os resultados do algoritmo E/TCPLEX para o problema com 30 tarefas e 5 máquinas são mostrados na tabela 5.6. A figura 5.2 mostra como se comportou a função objetivo durante a resolução do problema utilizando o algoritmo E/TCPLEX.

Para este problema com 30 tarefas foi estabelecido um tempo máximo de processamento de 5 horas e, mesmo com este tempo, a estimativa de limitante inferior não subiu acima do zero. Dessa forma, levando em conta que a estimativa de limite inferior pode ficar em valores muito baixos durante muito tempo, como no problema anterior, não é possível avaliar a qualidade da solução obtida.

Tabela 5.6 – Resultado do algoritmo E/TCPLEX – Problema com 30 tarefas e 5 máquinas:

Soma total de punições tempo 1 hora e meia	:	2382,00;
Tempo de processamento	:	18.000 segundos;
Melhor Limitante (Inferior)	:	0.00;
Número de Nós Arvore Branch & Cut	:	21.2381;
Número de Restrições	:	4.108;
Número de Variáveis (todas)	:	2.405;
Binárias	:	1.028;
Sequência em cada máquina:		
máquina 1=>	7, 9, 15, 27, 20, 12, 24, 18, 16, 25, 29, 8, 22, 3, 2;	
máquina 2=>	5, 10, 30, 25, 27, 18, 15, 6, 7, 24, 16, 11, 28, 20;	
máquina 3=>	7, 5, 3, 14, 6, 29, 10, 1, 4, 24, 25, 23, 13, 15;	
máquina 4=>	3, 4, 8, 7, 18, 23, 26, 30, 25, 19, 24, 22, 1, 5, 27, 29;	
máquina 5=>	4, 9, 29, 14, 27, 17, 3, 21, 20, 22, 28, 23, 12, 24;	

Tempo de Processamento (segundos)	Função Objetivo	Tempo de Processamento (segundos)	Função Objetivo
0.7	7640.00	395.0	2998.00
4.9	6174.00	1081.0	2973.00
20.5	5308.00	1258.8	2548.00
35.0	4734.00	3441.9	2534.00
96.8	4285.00	3943.2	2403.00
110.8	3882.00	4001.5	2382.00
197.8	3473.00		

Continuação tabela 5.6

Instantes de Término das Tarefas (segundos):

Tarefa	Instante de Término	Tarefa	Instante de Término	Tarefa	Instante de Término
1	830	11	789	21	794
2	789	12	827	22	805
3	790	13	796	23	817
4	738	14	767	24	833
5	841	15	805	25	784
6	708	16	667	26	773
7	752	17	788	27	777
8	750	18	758	28	807
9	754	19	797	29	756
10	806	20	795	30	783

Figura 5.2 – Valor da função objetivo & Tempo – Problema com 30 tarefas e 5 máquinas - E/TCPLEX

5.2.2 - Resultados utilizando a “meta-heurística” Tabu Search – E/TTabu

Parametrização da Heurística E/TTabu

Para a heurística E/TTabu, foi considerada uma parametrização para que seja possível avaliar como certos parâmetros interferem na função objetivo. É claro que uma parametrização eficiente teria que considerar muitas possibilidades de número de tarefas, número de máquinas e dados de entrada para que se pudessem selecionar os melhores parâmetros a se utilizar, mas isto não será feito no presente trabalho por ser um procedimento muito extenso e só serão apresentados alguns valores para a parametrização.

Para a Busca Tabu, a qualidade da solução está ligada à escolha do número de iterações, do número de elementos na lista tabu e do número de iterações consecutivas em que não há melhoria para se aplicar a matriz de diversificação. A alteração do número de elementos da lista tabu e do número de iterações sem redução do valor da função objetivo não causa mudanças significativas no tempo de processamento, assim será feito uma análise destes parâmetros. O simples aumento do tempo de processamento, por meio do aumento do número de iterações, por exemplo, já pode resultar em solução melhor, logo parâmetros que aumentem o tempo de processamento podem não ter grande relevância.

O problema utilizado para este estudo será aquele com 30 tarefas e 5 máquinas, com os dados apresentados na tabela 5.3 e será fixado o número de iterações em 5.000, já que este estudo tem a intenção apenas de mostrar o comportamento dos parâmetros que serão analisados e não encontrar o melhor valor de cada parâmetro para a maioria dos problemas que possam ser gerados. As análises feitas estão mostradas na tabela 5.6, onde o número de iterações sem redução do valor da função objetivo, para se utilizar a matriz de reincidência, é dado em porcentagem do número total de iterações.

Pela tabela 5.7, pode-se ver que o valor 5 para o número de elementos na lista tabu é o melhor valor, independente do número de iterações consecutivas sem redução do valor da função objetivo. É claro que seria necessário fazer muito mais testes, com muitos mais problemas com variação dos números de tarefas e de máquinas, para se garantir que este valor é o melhor para uma classe de problemas. Para o presente trabalho, será utilizado em todos os problemas propostos o valor 5 para o número de elementos na lista tabu e o valor de 60% do número total de

iterações para o número de iterações sem redução da função objetivo e não o de 20%, isto porque o tempo de processamento desta alternativa foi ligeiramente menor. O uso de um valor de 60% do número total de iterações para o número de iterações sem redução da função objetivo, que é um valor alto, significa que a matriz de reincidência, para a diversificação, será pouco aplicada, dando mais tempo ao procedimento de busca, da heurística Busca Tabu, usando uma solução inicial com boas características.

Tabela 5.7 – Comportamento da função objetivo em função dos valores dos parâmetros: número de elementos na lista Tabu e número de iterações sem melhoria (porcentagem do número total de iterações) – problema com 30 tarefas e 5 máquinas.

Numero de Elementos Lista Tabu	Numero de Iterações Sem melhoria (% do Número de iterações)	Função Objetivo	Tempo de processamento em Milisegundos
0	20	2671,00	787
5	20	2498,00	594
10	20	2571,00	614
15	20	2604,00	713
25	20	2571,00	683
30	20	2571,00	713
0	40	2671,00	833
5	40	2498,00	743
10	40	2571,00	753
15	40	2571,00	484
20	40	2571,00	824
25	40	2571,00	633
30	40	2571,00	763
0	60	2671,00	613
5	60	2498,00	558
10	60	2571,00	714
15	60	2579,00	613
20	60	2579,00	753
25	60	2571,00	743
30	60	2571,00	693

Na tabela 5.8 são mostrados os resultados obtidos para o problema proposto com 10 tarefas e 3 máquinas, cujos dados aparecem na tabela 5.2 utilizando a heurística E/TTabu e os parâmetros utilizados foram: número de iterações 10 mil, pois a heurística converge rapidamente; número de elementos na lista tabu 5; e 60% do número total de iterações para se utilizar a matriz de reincidência.

Esta heurística resolve uma versão aproximada do problema real determinado uma única sequência de prioridade de atendimento para as tarefas e como pode ser visto na tabela 5.8 o valor da função objetivo foi maior que o resultado obtido pelo algoritmo E/TCPLEX.

A figura 5.3 mostra como se comportou a função objetivo durante a resolução do problema utilizando a heurística E/TTabu. A tabela 5.9 apresenta o gráfico de ocupação das máquinas, para o problema com 10 tarefas e 3 máquinas, apresentando o início e fim de cada tarefa e sequência de atendimento das tarefas em cada máquina.

Tabela 5.8 – Resultado da heurística E/TTabu – Problema com 10 tarefas e 3 máquinas:

Soma de punições na 234ª iteração da Busca Tabu	:	120,00;
Soma de punições inicial	:	227,00;
Melhoria	:	47,13%;
Tempo de espera para início de atendimento das tarefas	:	47;
Sequência	:	8, 9, 2, 1, 4, 6, 7, 10, 5, 3;
Tempo de processamento	:	1 segundo;

Tabela 5.9 - Tabela de ocupação das máquinas - E/TTabu - Problema com 10 tarefas e 3 máquinas

Maq.1	Tar 8	Tar 2	Tar1	Tar 4	Tar 6	Tar 7	Tar 10	Tar 3	
Instantes de Início/fim	0/15	15/18	18/24	24/28	28/32	32/35	35/40	40/42	
Maq2	Tar 9	Tar 2	Tar 1	Tar 4	Tar 7	Tar 10			
Instantes de Início/fim	0/20	20/26	26/37	37/43	43/47	47/52			
Maq3	Tar 5	ociosa	Tar 8	Tar 2	Tar 6	ociosa	Tar 3	Ociosa	Tar 7
Instantes de Início/fim	0/11	11/15	15/28	28/32	32/37	37/42	42/44	44/47	47/49

Obs.: O tempo de espera é de 47, portanto o ponto de partida zero equivale a 47.

Figura 5.3 - Valor da função objetivo & número de iterações – Problema com 10 tarefas e 3 máquinas - E/TTabu

Na tabela 5.10 são mostrados os resultados obtidos para o problema proposto com 30 tarefas e 5 máquinas e os parâmetros utilizados foram: número de iterações 12.000, número de elementos na lista tabu 5 e 60% do número de iterações para se utilizar a matriz de reincidência. A figura 5.4 mostra como se comportou a função objetivo durante a resolução do problema utilizando a heurística E/TTabu.

Tabela 5.10 – Resultados da heurística E/TTabu – problema com 30 tarefas:

Soma das punições na 10.530ª iteração da Busca Tabu	:	2411,00;
Soma das punições inicial	:	3618,00;
Melhoria	:	33,36%;
Tempo de espera para início de atendimento das tarefas	:	725;
Sequência: 7, 17, 26, 24, 9, 21, 25, 23, 15, 4, 19, 3, 22, 30, 29, 28, 8, 6, 10, 11, 16, 20, 18, 27, 5, 14, 1, 13, 2, 12;		
Tempo de processamento	:	12,22 segundos .

Figura 5.4 – Valor da função objetivo & número de iterações – Problema com 30 tarefas e 5 máquinas - E/TTabu

5.2.3 – Resultados utilizando a heurística de busca exaustiva – E/TBE

Na tabela 5.10 são mostrados os resultados obtidos para o problema proposto com 10 tarefas e 3 máquinas utilizando a heurística E/TBE e o resultado obtido foi igual ao resultado do E/TTabu e maior do que o atingido pelo algoritmo E/TCPLEX, mas tendo a vantagem de enumerar as 180 sequências com o mesmo valor de função objetivo.

O problema proposto com 30 tarefas não foi executado por essa heurística por demandar um tempo de processamento excessivamente longo.

Tabela 5.11 – Resultado da heurística E/TBE – Problema com 10 tarefas e 3 máquinas:

Número de sequências possíveis	:	3628800;
Soma total de punições	:	120;
Tempo de espera para início de atendimento das tarefas	:	47;
Número de sequências com solução ótima	:	180;
Quatro exemplos de sequências com solução ótima	:	5, 8, 2, 1, 4, 6, 7, 9, 10, 3 8, 2, 1, 4, 6, 7, 5, 10, 9, 3 5, 8, 2, 1, 6, 4, 9, 7, 10, 3 9, 8, 5, 2, 1, 6, 4, 7, 10, 3
Tempo de processamento	:	7 Min 9 s

A heurística E/TBE, para aproximação adotada, consegue enumerar todas as sequências que geram o menor valor de função objetivo, mas em termos de tempo de processamento só é interessante para problemas de pequena instância e neste exemplo o E/TTabu também chegou ao mesmo resultado, mas só com uma sequência. Se o tempo for um fator crucial, a heurística E/TTabu é uma alternativa em relação ao E/TBE.

A tabela 5.12 apresenta o gráfico de ocupação das máquinas, para o problema com 10 tarefas e 3 máquinas, tomando a primeira sequência, dos quatro exemplos apresentados na tabela 5.9, como base. Nesta tabela são apresentados os instantes de início e fim de cada tarefa e a sequência de atendimento das tarefas em cada máquina.

Tabela 5.12 – Tabela de ocupação das máquinas – E/TBE – Problema com 10 tarefas e 3 máquinas

Maq.1	Tar 8	Tar 2	Tar1	Tar 4	Tar 6	Tar 7	Tar 10	Tar 3	
Instantes de Início/fim	0/15	15/18	18/24	24/28	28/32	32/35	35/40	40/42	
Maq2	Tar 9	Tar 2	Tar 1	Tar 4	Tar 7	Tar 10			
Instantes de Início/fim	0/20	20/26	26/37	37/43	43/47	47/52			
Maq3	Tar 5	ociosa	Tar 8	Tar 2	Tar 6	ociosa	Tar 3	ociosa	Tar 7
Instantes de Início/fim	0/11	11/15	15/28	28/32	32/37	37/42	42/44	44/47	47/49

Obs.: O tempo de espera é de 47, portanto o ponto de partida zero equivale a 47.

5.2.4 – Resultados utilizando uma heurística construtiva – E/THC

Na tabela 5.13 são mostrados os resultados obtidos para o problema proposto com 10 tarefas e 3 máquinas utilizando a heurística E/THC, que se propõe a obter uma solução aproximada para o problema real. A tabela 5.14 apresenta o gráfico de ocupação das máquinas, para o problema com 10 tarefas e 3 máquinas, apresentando o início e o fim de cada tarefa e sequência de atendimento das tarefas em cada máquina. Na tabela 5.15, são mostrados os resultados obtidos a partir da heurística E/THC para o problema proposto com 30 tarefas e 5 máquinas.

Tabela 5.13 – Resultado da heurística E/THC – Problema com 10 tarefas e 3 máquinas:

Soma total de punições	:	214;
Unidades de tempo de espera para início	:	34;
Sequência em cada máquina	:	máquina 1=> 8, 10, 7, 2, 6, 4, 1, 3;
		máquina 2=> 9, 2, 7, 1, 10, 4;
		máquina 3=> 8, 2, 5, 6, 7, 3;
Tempo de processamento	:	0,01 segundos.

Tabela 5.14 – Tabela de ocupação das máquinas – E/THC – Problema com 10 tarefas e 3 máquinas

Maq.1	Tar 8	Tar 10	Tar 7	Tar 2	Tar 6	Tar 4	Tar 1	Tar 3
Instantes de Início/fim	0/15	15/20	20/23	23/26	26/30	30/34	34/40	40/42
Maq2	Tar 9	ociosa	Tar 2	Tar 7	Ociosa	Tar 1	Tar 10	Tar 4
Instantes de Início/fim	0/20	20/26	26/32	32/36	36/40	40/51	51/56	56/62
Maq3	ociosa	Tar 8	Ociosa	Tar 2	Tar 5	Tar 6	Tar 7	Tar 3
Instantes de Início/fim	0/15	15/28	28/32	32/36	36/47	47/52	52/54	54/56

Obs.: A espera é de 34, portanto o ponto de partida zero equivale a 34.

Tabela 5.15 – Resultado da heurística E/THC – Problema com 30 tarefas e 3 máquinas:

Soma total de punições	:	3971,00;
Tempo de espera para início de atendimento das tarefas	:	553;
Sequência em cada máquina:		
máquina 1=>		18, 29, 15, 25, 12, 2, 24, 16, 22, 8, 9, 20, 27, 7, 3;
máquina 2=>		18, 15, 25, 24, 16, 10, 30, 11, 6, 20, 27, 7, 28, 50;
máquina 3=>		29, 15, 25, 24, 14, 10, 6, 23, 7, 4, 13, 3, 5, 10;
máquina 4=>		26, 18, 29, 25, 24, 10, 30, 22, 19, 8, 23, 27, 7, 4, 3, 5, 1;
máquina 5=>		29, 12, 24, 17, 14, 22, 21, 9, 23, 20, 27, 4, 28, 3;
Tempo de processamento	:	0,01 segundos.

5.2.5 – Comparação do algoritmo E/TCPLEX e das heurísticas E/TTabu, E/TBE e E/THC na resolução dos dois problemas

A tabela 5.16 apresenta um apanhado dos resultados obtidos pelo algoritmo E/TCPLEX e das heurísticas E/TTabu, E/TBE e E/THC para os dois problemas tratados nesta seção.

Para o algoritmo E/TCPLEX, que explora todo o espaço de soluções do problema, foi obtido um valor de função objetivo bem menor para o problema com 10 tarefas e 3 máquinas e ligeiramente melhor para o problema com 30 tarefas e 5 máquinas, embora os custos de um tempo de processamento bem maior.

A figura 5.5 apresenta os gráficos de ocupação de cada máquina pelas tarefas, do problema com 10 tarefas e 3 máquinas, levando em conta as sequências obtidas por meio do algoritmo E/TCEPLEX e das heurísticas E/TTabu, E/TBE e E/THC.

Tabela 5.16 – Comparação dos resultados dos dois problemas

	Problema 10 tarefas 3 máquinas	Tempo de processamento. (segundos)	Problema 30 tarefas 5 máquinas	Tempo de processamento (segundos)
E/TCPLEX	7	11330	2382	5400
E/TTabu	120	1	2411	12
E/TBE	120	429	-	-
E/THC	214	0	3971	0

Figura 5.5 – Gráficos de ocupação das máquinas - E/TCEPLEX, E/TTabu, E/TBE e E/THC.

Obs.: “N” significa que a máquina está ociosa.

5.3 – Resultado do Algoritmo e das Heurísticas para Exemplos de Problemas Diversos

Além dos dois problemas já vistos, mais uma série de exemplos de problemas diversos foi gerada, para análise de eficiência das propostas. Estes problemas foram gerados tendo como base os dados propostos por que eram encontrados no endereço (site) da Internet <http://msemga.ms.ic.ac.uk/info.html> (atualmente não estão mais disponíveis neste site, como já comentado), para problemas com uma única máquina, data de entrega comum e que também punem tarefas que são entregues com atrasos ou adiantamentos, já que esta é uma característica importante do problema objeto do presente trabalho.

5.3.1 – Problemas do tipo máquina única e data comum de entrega - E/TCPLEX, E/TTabu, E/TBE e Heurística de Feldmann e Biskup

Na tabela 5.17 são mostrados os resultados de alguns problemas do tipo máquina única e data de entrega comum, para os quais são conhecidos os resultados obtidos por Feldmann e Biskup (2003), que propuseram heurísticas específicas, sendo assim mais eficientes para este tipo de problema. A heurística E/TBE, proposto neste trabalho, chega à solução ótima para este tipo de problema, porém o tempo de processamento cresce muito rapidamente com o número de tarefas e a heurística somente foi aplicada para problemas com 10 tarefas. Somente para problemas com uma máquina é que as heurísticas E/TTabu e E/TBE e E/TCPLEX tem um mesmo ótimo a perseguir, ou seja, tem um mesmo espaço de busca. O tempo de processamento da heurística E/TTabu são bem baixos, mas os resultados das heurísticas propostas por Feldmann e Biskup (2003) são melhores.

A heurística E/THC não foi utilizada para essa classe de problema, pois nesta heurística as sequências são construídas para cada máquina e quando se tem apenas uma máquina, algumas sequências ficam iguais às sequências construídas como solução inicial na heurística E/TTabu e as que não são iguais não são possíveis de calcular, já que esta heurística

não foi desenvolvida para esta classe de problema pois já existe na literatura algoritmos e heurísticas mais eficientes.

Na tabela 5.17, para o algoritmo E/TCPLEX, são mostrados dois resultados para cada problema. Para primeira alternativa (I) foi utilizado um tempo de processamento igual a 10 minutos para todos os problemas; para a segunda alternativa (II) ampliou-se bastante o tempo de processamento, fixando-o em 1 hora para os problemas com 10 tarefas, 2 horas para os problemas com 20 tarefas e 5 horas para os de 50 tarefas.

Para as heurísticas os tempos de processamentos foram:

- Para a heurística E/TBE: 7 minutos e 41 segundos; 7 minutos e 55 segundos e 7 minutos e 4 segundos para os problemas com 10 tarefas;
- Para a heurística E/TTabu: os problemas com 10 tarefas ficaram entre 5 e 7 segundos usando 50mil iterações, os problemas com 20 tarefas ficaram entre 12 e 15 segundos usando 50mil iterações e os problemas com 50 tarefas ficaram entre 13 e 16 segundos usando 5mil iterações.

Na tabela 5.17 é possível observar que, mesmo com altos tempos de processamento, os resultados com o E/TCPLEX para problemas com uma máquina não são bons. Eventualmente seria possível reduzir esse tempo de processamento particularizando a modelagem matemática para tal caso; este, porém, não é o objetivo dessa pesquisa.

Considerando que, no caso de problemas com uma única máquina e 10 tarefas, a heurística de busca exaustiva E/TBE gera a solução ótima do problema real, conclui-se que o limitante inferior para a função objetivo obtido na solução do algoritmo E/TCPLEX, é fraco, mesmo para tempos de processamento iguais a 1 hora. Assim, pelo menos nesses casos, o limitante inferior do CPLEX não é um bom parâmetro para avaliar a qualidade das soluções geradas pelo algoritmo e heurísticas.

Tabela 5.17 – Comparação do algoritmo E/TCPLEX, das heurísticas E/TTabu e E/TBE e com uma heurística específica proposta por Feldmann e Biskup (2003)

Prob.	Ntar	E/TTabu	E/TBE	Feldmann	E/TCPLEX	E/TCPLEX		
		F.O	F.O	F.O	F.O I	Limite I CPLEX	F.O II	Limite II CPLEX
1	10	1943	1936	1936	2007	81	1977	393
2	10	1051	1001	1046	1058	370	1012	599
3	10	1770	1586	1586	1672	272	1611	397
4	20	4789		4431	7076	11	4979	38
5	20	5715		4897	6558	26	5485	44
6	20	6834		6331	8632	55	7811.	102
7	50	49167		42363	70598	0	55763	9
8	50	21848		19279	39945	11	27153	17
9	50	38675		37641	55547	0	54671	14

Para esses problemas com uma única máquina, mesmo com os tempos de processamento mais longos, os resultados obtidos com o algoritmo E/TCPLEX são piores, exceto num caso, que aqueles obtidos por Feldmann e Biskup (2003) e essa desvantagem aumenta com o número de tarefas.

Para os problemas com 10 tarefas os resultados da heurística E/TTabu foram bons comparado com o E/TBE, que também busca uma sequência de prioridades; para os problemas com mais tarefas a heurística E/TTabu gerou, exceto num caso, melhores resultados que o algoritmo E/TCPLEX.

5.3.2 – Problemas com 10 tarefas e datas de entrega diferentes - Algoritmo E/TCPLEX, e as heurísticas E/TTabu, E/TBE e E/THC

A tabela 5.18 mostra os resultados obtidos para problemas com 10 tarefas, com 1 e 3 máquinas e datas de entrega diferentes, utilizando-se as heurísticas E/TTabu, E/THC, E/TBE e o algoritmo E/TCPLEX.

Para esta tabela os tempos de processamento são os seguintes:

- Para o algoritmo E/TCPLEX foi usado um tempo de processamento de 1 hora;
- Para a heurística E/TBE os tempos ficaram entre 7 e 8 minutos;
- Para a heurística E/TTabu os tempos ficaram entre 5 e 7 segundos, usando 50mil iterações;
- Para a heurística E/THC os tempos ficaram muito próximos do zero.

Tabela 5.18 - Comparação do algoritmo E/TCPLEX com as heurísticas E/TTabu, E/THC e E/TBE para problemas com 10 tarefas com datas de entrega não comuns.

Prob.	Nmaq	E/TTabu	E/TBE	E/THC	E/TCPLEX	Limitante CPLEX
		F.O	F.O	F.O	F.O	
1	1	1138	1134	1162	1541	83
2	1	323	243	347	247	8
3	1	976	970	986	1140	178
4	3	446	446	317	162	0
5	3	125	125	166	58	30
6	3	138	129	133	49	37
7	3	133	101	227	71	30
8	3	356	336	839	27	27
9	3	401	401	649	389	239
10	3	302	298	384	298	295

Da análise dos resultados da tabela 5.18, são pertinentes os seguintes comentários:

1. Para o problema 8, com 3 máquinas e 10 tarefas, o algoritmo E/TCPLEX encontrou uma solução ótima, para a qual a função objetivo é 27. Enquanto isto, a heurística E/TBE encontrou uma solução ótima dentro do subconjunto reduzido para a qual o valor da função objetivo é 336. Portanto, conclui-se que, para alguns problemas, a heurística E/TTabu pode gerar soluções bem pobres para o problema real. Um caminho para superar essa limitação da heurística E/TTabu é aproveitar a solução gerada por esta heurística como ponto de partida para o E/TCPLEX ou *Local Branching*, criando, assim, procedimentos combinados para resolução do problema. Observe-se que, no caso do problema 10 os valores da função objetivo para soluções ótimas do E/TBE e do E/TCPLEX são praticamente iguais (para dizer que são iguais precisaria saber que o limite 295 subiria para 298).

2. O desempenho relativo do algoritmo E/TCPLEX frente aos das heurísticas, é mais fraco para problemas com máquina única. Nos 3 exemplos com máquina única a solução gerada pelo E/TCPLEX perde para a solução obtida pelo E/TTabu em dois deles. Para problemas com 3 máquinas, o desempenho do E/TCPLEX foi significativamente melhor que a heurística E/TTabu, exceto nos problemas 9 e 10 em que a vantagem do E/TCPLEX foi bem pequena.

3. Em termos de tempo de computação, o desempenho do E/TCPLEX também foi melhor para problemas com 3 máquinas. Para os problemas 8 e 10, a solução ótima ou quase ótima foi obtida em 5 e 7 minutos respectivamente; para o problema 6, a solução com função objetivo igual a 49 foi encontrada após 7 minutos de processamento, mas somente após 20 minutos, o valor do limitante inferior passou a 48.

4. O desempenho da heurística E/THC em relação ao E/TCPLEX foi pior para os problemas com 3 máquinas; na maioria deles a desvantagem é bem grande. Dado que o tempo de processamento dessa heurística construtiva é relativamente pequeno, pode ser interessante investir esforços para aperfeiçoar a estratégia utilizada para construção dessa heurística.

Para os problemas 9 e 10 os dados de entrada foram gerados de modo que os instantes de início e fim das janelas de tempo de todas as tarefas fossem zero; deste modo todas as tarefas teriam punição por atraso e esta punições ficou sendo 1 para todas as tarefas, assim o problema ficaria sendo minimizar a somatória dos tempos de conclusão das tarefas $Min\left(\sum_{i=1}^N c_i\right)$, que é uma medida de desempenho sem nenhuma importância na prática, é só a análise de m problemas mais simples. Para estes dois problemas o algoritmo E/TCPLEX teve bons resultados mesmo na estimativa dos limitantes inferiores.

O problema 3 foi executado pelo E/TCPLEX durante 71895,2 segundos (quase 20 horas) para se verificar se com um esforço computacional maior o resultado melhora significativamente. Este resultado é apresentado na tabela 5.19.

Tabela 5.19 - Resultados do CPLEX para o problema 3 da tabela 5.18

Tempo Segundos	Função Objetivo		
0.0	2066	Tempo total [s]	: 71895.2
0.1	1818	Melhor Limitante (Inferior)	: 930.70
3.0	1604	Nós da Arvore Branch & Cut	: 1739232
3.3	1582	Valor da função objetivo	: 1130
4.0	1459	Restrições	: 397
8.6	1430	Variáveis (todas)	: 253
9.6	1280	Binárias	: 90
26.1	1250		
26.2	1242		
96.3	1186		
201.5	1184		
238.0	1178		
526.9	1154		
2802.3	1152		
3215.7	1146		
3545.1	1140		
28027.3	1138		
28762.8	1130		

O valor da função objetivo da melhor solução melhorou muito pouco; passou de 1140 obtido em 3545,1 segundos, para 1130 obtido em 28762,8 segundos.

5.3.3 – Problemas diversos - Algoritmo E/TCPLEX e Heurísticas E/TTabu e E/THC

A tabela 5.20 que apresenta uma comparação das heurísticas E/TTabu, E/THC e E/TCPLEX para problemas com 20, 50 e 100 tarefas.

Para esta tabela os tempos de processamento foram os seguintes:

- Para problemas com 20 tarefas: por volta de 15 segundos para a heurísticas E/TTabu fazendo 50 mil iterações, 12 “Milisegundos” para o algoritmo E/THC e 10 minutos para o E/TCPLEX;
- Para os problemas com 50 tarefas: por volta de 40 segundos para a heurística E/Tabu fazendo 50 mil iterações, 20 “Milisegundos” para a heurística E/THC e 10 minutos para o algoritmo E/TCPLEX;
- Para os problemas com 100 tarefas: 1 minuto para a heurística E/TTabu fazendo 50 mil iterações, 35 “Milisegundos” para a heurística E/THC e 10 minutos para o algoritmo E/TCPLEX.

Para essa tabela foi estipulado um tempo de processamento relativamente baixo, já que são problemas com 20, 50 e 100 tarefas, para comparar os resultados do algoritmo e das heurísticas com tempos compatíveis. Na tabela 5.22 os tempos de processamento serão aumentados.

Tabela 5.20 - Comparação E/TCPLEX, E/TTabu e E/THC - problemas com 20, 50 e 100 tarefas.

Prob.	Ntar	Nmaq	BuscaTabu	E/THC	E/TCPLEX	Limitante
			F.O	F.O	F.O	CPLEX
1	20	1	1490	2034	2345	0
2	20	1	2643	3335	3182	0
3	20	1	6057	5555	6235	0
4	20	4	2281	4065	5229	47
5	20	4	1975	2438	1592	0
6	20	4	2991	3769	2535	0
7	20	4	2171	2219	1021	0
8	20	4	1356	1390	1178	0
9	20	4	1505	1373	1750	0
10	20	4	1056	1375	1340	345
11	20	4	1144	1453	1324	368
12	20	4	1554	2016	2199	398
13	20	4	1878	2637	1716	130
14	20	4	1261	2088	1425	0
15	50	1	28739	26765	23518	0
16	50	1	16397	17846	23859	0
17	50	1	21779	23271	26404	0
18	50	5	17605	31684	19206	1006
19	50	5	17081	29892	15684	1102
20	50	5	14497	14691	70808	0
21	50	5	9308	10361	10511	0
22	50	5	8154	10241	14251	0
23	50	5	8745	8383	16528	0
24	100	5	78944	80370	201321	0
25	100	5	62197	51477	173024	0

Novamente para os problemas com uma única máquina, os resultados do algoritmo E/TCPLEX não foram bons, mas aumentando o número de máquinas o desempenho do algoritmo melhora, tanto que para os problemas 5, 6, 7, 8, 13, 15 e 19 os melhores resultados foram obtidos com esse algoritmo. Nesta tabela os problemas 10, 11, 12, 18 e 19 foram gerados da mesma forma que os problemas 9 e 10 da tabela 5.18, ou seja, a medida de desempenho passa a ser $\text{Min}\left(\sum_{i=1}^N c_i\right)$ e para o problema 19 com 50 tarefas e 5 máquinas o resultado do E/TCPLEX foi melhor que os resultados do E/TTabu e E/THC. A estimativa de limitante inferior nos problemas 18 e 19 foram melhores.

A heurística E/THC, que gera abordagem aproximada do problema real, teve para os problemas 3, 9, 15 e 25, resultados melhores do que a heurística E/TTabu, com isto se ratifica a o comentário anterior de que é interessante investir esforços para aperfeiçoar esta heurística.

Aumentando o tempo de processamento, do algoritmo E/TCPLEX, para 1 hora para resolução dos problemas 9 e 12, foi obtido resultados maiores do que os resultados da heurística E/THC. Os valores da função objetivo da melhor solução, mesmo aumentando o tempo de processamento ficaram piores, passaram de 1750 para 1576 e de 2199 para 2086 respectivamente, sendo que a estimativa de limitante inferior para o problema 9 continuou zero e a do problema 12 passou de 398 para 401.

Nos exemplos de problemas com 100 tarefas, vistos na tabela 5.20, com 10 minutos de tempo de processamento, utilizando o algoritmo E/TCPLEX, só foi encontrada uma solução inteira, cabe mencionar que mesmo aumentando o tempo de processamento, a solução obtida com o algoritmo E/TCPLEX é pior que as encontradas pelas heurísticas E/TTabu e E/THC.

Aumentando o tempo de processamento do problema 25 para 6 horas (21600 segundos) foram obtidos os resultados mostrados na tabela 5.21; há somente três resultados (uma solução inicial e duas melhorias); já para o problema 20 com 50 tarefas foi encontrado um número maior de soluções, passando o tempo de processamento para 6 horas. Os resultados encontrados são mostrados na tabela 5.22.

Tabela 5.21 – Resultados do CPLEX para o problema 25 da tabela 5.20

Tempo	Função		
Segundos	Objetivo		
		Tempo Total	: 21601.5
		Melhor Limitante (Inferior)	: 0.00
3647.6	173024	Nós da Arvore Branch & Cut	: 1301
10402.8	106098	Valor da função objetivo	: 87227
16571.2	87227	Restrições	: 44694
		Variáveis (todas)	: 25726
		Binárias	: 12286

Tabela 5.22 – Resultados do CPLEX para o problema 20 da tabela 5.20

Tempo Segundos	Função Objetivo	
36.5	105425	Tempo total [s] :21600.1
119.8	103618	Melhor Limitante (Inferior) : 0.00
173.1	95845	Nós da Arvore Branch & Cut : 34400
339.4	70808	Valor da função objetivo : 23055
956.4	69116	Restrições : 11551
1050.5	54469	Variáveis (todas) : 6693
2038.9	47149	Binarias : 3054
5533.0	44624	
6023.5	40426	
8515.8	38361	
10009.9	35336	
16362.2	33173	
16487.0	30988	
16608.9	28811	
16731.0	26405	
17031.8	25878	
17148.1	23055	

Novamente, cabe mencionar que mesmo aumentando o tempo de processamento, a solução obtida com o algoritmo E/TCPLEX é pior que as encontradas pelas heurísticas E/TTabu e E/THC.

Considerando que, para a maioria dos problemas mostrados nas tabelas 5.18 e 5.20, os resultados com o E/TCPLEX não foram bons comparados com os resultados da heurística E/TTabu, aproveitou-se para aumentar também o tempo de processamento dessa heurística.

Para a heurística E/TTabu foram utilizadas 500mil iterações para os problemas com 10, 20 e 50 tarefas resultando em um tempo médio de processamento de 70, 200 e 1800 segundos respectivamente. Para os problemas com 100 tarefas foram usados 250mil iterações e o tempo médio de processamento foi de 7200 segundos. Para a heurística E/TCPLEX os tempos foram: 3600 segundos para os problemas com 10 tarefas, 7200 segundos para os problemas com 20 tarefas, 21600 segundos para os problemas com 50 e 100 tarefas.

Na tabela 5.23, são apresentados os resultados obtidos a partir da heurística E/TTabu e do algoritmo E/TCPLEX, com os tempos de processamento das tabelas 5.18 e 5.20, chamando-os de E/TTabu I e E/TCPLEX I; e com os tempos maiores chamando-os de

E/TTabu II e E/TCPLEX II. As estimativas de limitantes inferiores são aquelas obtidas para tempos maiores.

Tabela 5.23 - Comparação E/TTabu e E/TCPLEX com tempo de processamento maior

Prob./Tab	Ntar	Nmaq	E/TTabu I	E/TTabu II	E/TCPLEX I	E/TCPLEX II	Limitante Inferior
1/18	10	1	1138	1138	1541	1138	218,49
2/18	10	1	323	323	247	247	24
4/18	10	3	446	446	162	162	0
5/18	10	3	125	125	58	58	35
7/18	10	3	133	131	71	71	70
9/18	10	3	401	401	389	389	261
1/20	20	1	1490	1490	2345	1905	0
2/20	20	1	2643	2643	3182	2824	0
3/20	20	1	6057	6057	6235	6112	0
4/20	20	4	2281	2281	5229	2991	47
9/20	20	4	1505	1505	1750	1456	0
10/20	20	4	1056	1056	1340	1324	350
11/20	20	4	1144	1144	1324	1320	386
12/20	20	4	1554	1554	2199	1865	399
14/20	20	4	1261	1261	1425	1234	0
16/20	50	1	16397	11337	23859	18340	2
17/20	50	1	21779	13809	26404	21354	0
20/20	50	5	14497	12716	70808	12280	0
22/20	50	5	8154	6192	14251	11313	0
23/20	50	5	8745	8070	16528	7289	0
24/20	100	5	78944	44259	201321	87227	0
25/20	100	5	62197	47677	173024	91304	0

Obs.: Prob./Tab => 1/18 => é o problema 1 da tabela 18

Para a heurística E/TTabu, com o aumento de tempo de processamento não houve melhoria dos resultados para problemas com até 20 tarefas, mas para problemas com maior número de tarefas os resultados melhoraram bastante. No caso do algoritmo E/TCPLEX, houve melhoria da solução para a maioria dos problemas, principalmente para as instâncias maiores;

mesmo assim, para essas instâncias a solução gerada pelo E/TCPLEX é pior do que a gerada pelo E/TTabu.

As estimativas de limitantes inferiores, exceto num caso, continuaram muito abaixo das soluções geradas pelo E/TCPLEX e E/TTabu.

5.4 – Propostas de Melhoria

A modelagem usada para o algoritmo E/TCPLEX tem grande potencial, mas exige tempo de processamento muito alto para instâncias maiores de problemas, em razão da natureza combinatória. Assim, no presente trabalho também são propostas mais duas alternativas de resolução com intuito de melhorar a eficiência em termos de tempo de processamento e de qualidade dos resultados.

5.4.1 – Interação do E/TCPLEX com o E/TTabu e E/THC

Uma alternativa para ser estudada, proposta neste trabalho, seria usar uma interação das heurísticas E/TTabu ou E/THC com o E/TCPLEX, ou seja, usar a solução da heurística E/TTabu ou E/THC como um ponto de partida para o algoritmo E/TCPLEX.

O fornecimento de uma solução inicial para o E/TCPLEX não garante, porém, que o desempenho do algoritmo seja melhor. Por exemplo, no caso do problema 4 da tabela 5.18, usando a solução da heurística E/THC, como ponto de partida, o algoritmo E/TCPLEX gera uma solução viável com função objetivo igual a 262; durante 2 horas e meia de processamento não consegue solução melhor e, com 6 horas de processamento o valor da função objetivo para a melhor solução encontrada é 143; sem ponto de partida o E/TCPLEX com 10 minutos já havia conseguido um valor de função objetivo de 162 e com 6 horas de processamento este valor se manteve.

A seguir serão apresentados, na tabela 5.24, os resultados obtidos com o algoritmo E/TCPLEX, para alguns problemas da tabela 5.20, usando os resultados das heurísticas E/TTabu, chamando-os de CPTABU e E/THC, chamando de CPHC, com tempos de processamento de 10 minutos e 6 horas. Os resultados das heurísticas E/THC e E/TTabu utilizados como ponto de partida são os mostrados na tabela 5.20 para comparação dos resultados obtidos com os resultados da tabela 5.23.

Tabela 5.24 – Resultados do E/TCPLEX a partir de solução inicial – (Problemas da tabela 5.20)

		10 min	6 horas	6 horas		10 min	6 horas	6 horas	Tab. 5.23
Prob.	E/TTabu	CPTABU	CPTABU	Limitante	E/THC	CPHC	CPHC	Limitante	E/TCPLEX
4/20	2281	2207	2207	47	4065	3888	3888	47	2991
10/20	1056	1051	1031	401	1375	1339	1006	409	1324
11/20	1144	1133	1133	410	1453	1123	1121	434	1320
12/20	1554	1554	1551	402	2016	1839	1654	453	1865
22/20	8154	7310	7310	0	10241	7310	5615	0	11313
24/20	78944	37687	37687	0	80370	37687	37687	0	87227
25/20	62197	47257	47257	0	51477	47257	47257	0	91304

O problema 4, da tabela acima, sem considerar ponto de partida e sendo processado por 10 minutos, gerou uma função objetivo com valor de 5229 e sendo processado por 6 horas o valor chegou a 2991. Com o CPTABU, este mesmo problema, gerou um valor de função objetivo de 2207 com 10 minutos de processamento e este valor se manteve após 6 horas de processamento. Este valor significa uma grande melhoria comparando com o E/TCPLEX e uma pequena melhoria em relação ao ponto de partida do E/TTabu obtido com menor tempo de processamento. Para os problemas 10, 11 e 22, usar como ponto de partida o resultado do E/THC foi melhor, mesmo sendo o valor de função objetivo do E/THC maior que o do E/TTabu. As estimativas de limites inferiores continuaram muito baixas.

Para os problemas 24 e 25, com 100 tarefas e 5 máquinas, os resultados foram iguais para os dois procedimentos e não se modificaram quando o tempo de processamento passou de 10 minutos para 6 horas. São problemas cuja resolução apresenta grande dificuldade computacional.

Comparando os resultados das tabelas 5.23 e 5.24, é possível destacar: os problemas 4/20, 10/20, 11/20 e 12/20 para 20 tarefas, onde o E/TTabu não obteve melhoria aumentando o

tempo de processamento, obtiveram melhores valores de função objetivo que o E/TCPLEX na tabela 5.23 e na tabela 5.24 o CPTABU melhorou um pouco os resultados do E/TTabu; os problemas 22/20, 24/20 e 25/20 para 50, 100 e 100 tarefas respectivamente, o E/TTabu obteve boa melhoria aumentando o tempo de processamento e continuou com melhores resultados que os do E/TCEPLEX na tabela 5.23 e na tabela 5.24 o CPTABU não obteve nenhuma melhoria nesses problemas, são problemas com muitas tarefas.

Ainda na tabela 5.24, o CPHC obteve boa melhoria considerando os resultados do E/THC.

5.4.2 – Interação da heurística *Local Branching* com o E/TTabu e E/THC

Outra alternativa de procedimento combinado, proposta neste trabalho, é a utilização dos resultados das heurísticas E/TTabu e E/THC como soluções iniciais da heurística *Local Branching*, que foi vista no capítulo 2.

Na tabela 5.25 são apresentados alguns resultados encontrados para os mesmos problemas da tabela 5.20, utilizando como ponto de partida para a heurística *Local Branching* os resultados das heurísticas E/TTabu e E/THC chamando-as de LBTabu e LBHC respectivamente, também com tempos de 10 minutos e 6 horas.

Tabela 5.25 – Resultados da heurística *Local Branching* – (problemas tabela 5.20)

		10 min	6 horas	6 horas		10 min	6 horas	6 horas	Tab. 5.23
Prob.	E/TTabu	LBTabu	LBTabu	Limite	E/THC	LBHC	LBHC	Limite	E/TCPLEX
4/20	2281	2123	2020	45	4065	3796	2031	47	2991
10/20	1056	1051	1027	391	1375	1105	1006	419	1324
11/20	1144	1133	1133	343	1453	1162	1082	420	1320
12/20	1554	1554	1531	394	2016	1766	1553	438	1865
22/20	8154	7310	5035	0	10241	6818	5540	0	11313
24/20	78944	37687	37687	0	8037	37687	35804	0	87227
25/20	62197	47257	47257	0	51477	47257	47256	0	91304

Os problemas que consideram a heurística LBHC foram processados com a nova versão do *Local Branching* implementada no CPLEX versão 11.0. As diferenças entre as duas versões do *Local Branching* podem afetar as comparações.

Um outro ponto a considerar é que para o *Local Branching* são usados parâmetros de entrada como: tempo de processamento em cada vizinhança de busca, tamanho da vizinhança de busca e seria interessante uma análise do efeito desses parâmetros.

No geral a melhoria conseguida em relação aos resultados iniciais das heurísticas E/TTTabu e E/THC foram boas.

A tabela 5.26 apresenta uma comparação entre as heurísticas LBTabu e LBHC com os algoritmos CPTabu e CPHC para 6 horas de processamento.

Pela tabela 5.26, pode-se observar que houve ligeira vantagem para os resultados da heurística *Local Branching*, lembrando que a heurística LBHC usou a implementação feita na versão 11 do pacote “CPLEX”.

Tabela 5.26 – Comparação - LBTabu , LBHC, CPTabu e CPHC

Prob.	CPTabu	LBTabu	CPHC	LBHC *
4	2207	2020	3888	2031
10	1031	1027	1006	1006
11	1133	1133	1121	1082
12	1551	1531	1654	1553
22	7310	5035	5615	5540
24	37687	37687	37687	35804
25	47257	47257	47257	47257

5.5 – Disposições Finais do Capítulo

Na tabela 5.17 foram apresentados 10 resultados comparando o E/TTabu com uma heurística específica, proposta por Feldmann e Biskup (2003). No entanto uma comparação entre as duas heurísticas foi feita para 80 problemas propostos por Feldmann e Biskup (2003). Houve empate em 8 problemas, a heurística proposta por Feldmann e Biskup (2003) venceu 44 problemas e perdeu nos 28 problemas restantes. O desvio médio entre os valores da função objetivo foi 8,49% a favor da heurística proposta por Feldmann e Biskup (2003).

Para os problemas que consideram apenas uma única máquina, o algoritmo E/TCPLEX e as heurísticas E/TTabu e E/TBE perseguem o mesmo ótimo, sendo que o E/TBE enumera todas as soluções ótimas, mas em termos de tempo de processamento a heurística E/TTabu tem grande vantagem..

O algoritmo E/TCPLEX teve desempenho pior, em termos de função objetivo, que as heurísticas nos problemas que consideraram uma única máquina em todas as instâncias de problemas propostos com baixo tempo de processamento, mas o seu desempenho melhorou bastante aumentando o tempo, como no caso do problema 1/18 da tabela 5.23. Aumentando-se o número de máquinas o desempenho do E/TCPLEX melhora como, por exemplo, o problema 20/20 da tabela 5.23.

Quanto a heurística construtiva E/THC, que gera uma sequência de tarefas para cada máquina, admitindo inserção de tempo ocioso, cabe mencionar que ela conduziu a resultados melhores que os resultados do E/TTabu e E/TCPLEX no problema 4 da tabela 5.18 e nos problemas 3, 23 e 25 da tabela 5.20. Mas na maioria dos casos seus resultados foram piores. No entanto esta heurística é a que tem menor tempo de processamento.

Com o aumento de tempo de processamento pode haver melhoria acentuada dos resultados do algoritmo E/TCPLEX, mas mesmo assim seu desempenho pode continuar inferior aos das heurísticas E/TTabu e E/THC, como por exemplo, no problema 12/20 da tabela 5.23.

Desta forma a opção de se usar os resultados das heurísticas E/TTabu e E/THC como ponto de partida pode ser interessante.

A opção de se usar os resultados das heurísticas E/TTabu e E/THC como ponto de partida para o algoritmo E/TCPLEX e heurística *Local Branching*, conduziu em melhorias para os resultados dos problemas. Porém, não foi possível avaliar a qualidade da solução obtida, pois as estimativas de limitante inferior do pacote CPLEX se mostraram fracas. Por exemplo, no problema com 10 tarefas e 3 máquinas da tabela 5.4, a solução com valor 7 foi encontrada em 776 segundos, mas a estimativa de limite inferior só chegou a 6,95 depois de 11330 segundos, sendo que ficou no valor zero por mais de 1 hora.

Claro que o número de problemas executados pelo algoritmo E/TCPLEX e pela heurística *Local Branching* não são suficientes para avaliar qual deles é melhor para o problema cerne deste trabalho; a intenção foi apresentar alternativas de resolução.

Finalmente, apesar de haver problemas para os quais a solução ótima do “problema aproximado” é muito pior que a solução ótima do problema real, mesmo assim a utilização das heurísticas se torna interessante em termos de tempo de processamento, pela possibilidade de se atingir soluções melhores para tempos razoáveis em instâncias maiores e pela alternativa de uso de seus resultados como ponto de partida para o E/TCPLEX e o *Local Branching*.

Uma última bateria de testes foi feita para avaliar a qualidade de uma hipótese adotada para gerar as soluções que são consideradas pelas heurísticas E/TTabu e E/TBE.

De acordo com a hipótese adotada, ao terminar o processamento de uma tarefa em uma dada máquina, a próxima tarefa é escolhida, dentre as tarefas que estejam na fila daquela dada máquina, de acordo com a sequência de prioridades que caracteriza a solução e não com a posição na fila. A comparação entre os resultados da tabela 5.27 está ligado aos problemas da tabela 5. 18 para o E/TTabu e E/TBE e tabela 5.20 apenas para o E/TTabu.

Tabela 5.27 – Comparação das heurísticas E/TTabu e E/TBE para problemas com 10 tarefas, admitindo e não admitindo prioridade no atendimento das tarefas.

Prob/Tab.	Nmaq	E/TTabu	E/TTabu	E/TBE	E/TBE
		c/prioridade	1° da fila	c/prioridade	1° da fila
		F.O	F.O	F.O	F.O
1/18	1	1138	1470	1134	1446
2/18	1	323	283	243	263
3/18	1	976	1195	970	967
4/18	3	446	446	446	446
5/18	3	125	319	125	301
6/18	3	138	307	129	303
7/18	3	133	247	101	247
8/18	3	356	484	336	452
9/18	3	401	659	401	439
10/18	3	302	442	298	378
1/20	1	1490	1549		
2/20	1	2643	1933		
3/20	1	6057	6091		
4/20	4	2281	3408		
5/20	4	1975	2383		
6/20	4	2991	3762		
7/20	4	2171	3271		
8/20	4	1356	1572		
9/20	4	1505	1850		
10/20	4	1056	1083		
11/20	4	1144	1264		
12/20	4	1554	1744		
13/20	4	1878	2069		
14/20	4	1261	1886		
15/20	5	28739	31765		
16/20	5	16397	18976		
17/20	5	21779	25343		
18/20	5	17605	18901		
19/20	5	17081	19065		
20/20	5	14497	13765		
21/20	5	9308	10276		
22/20	5	8154	7901		
23/20	5	8745	9321		
24/20	5	62197	65843		
25/20	5	55261	59187		

Obs.: Prob./Tab => 1/18 => é o problema 1 da tabela 18

Do exame dos resultados da tabela 5.27, constata-se que, para o conjunto dos problemas resolvidos, a hipótese de atendimento de acordo com a sequência de prioridades conduziu à soluções melhores que aquelas obtidas com a hipótese de atendimento de acordo com o instante de chegada na fila e, por isto, sugere-se que ela seja mantida.

CAPÍTULO 6

CONCLUSÕES E SUGESTÕES

Neste capítulo são apresentadas as principais conclusões da pesquisa desenvolvida sobre programação de tarefas em ambiente *flow shop* e são feitas sugestões para a continuidade do trabalho. Na seção inicial faz-se um breve resumo sobre o tema da pesquisa e a forma como ele foi tratado.

6.1 – Breve Resumo da Pesquisa Desenvolvida

O objetivo deste estudo é o problema de programação de tarefas em ambiente *flow shop* com algumas características particulares que, juntas, o tornam diferente dos problemas usuais. Há N tarefas a serem processadas por M máquinas diferentes e cada máquina tem um roteiro específico ao longo da oficina (*shop*), não passando necessariamente por toda as máquinas. Cada tarefa deve ser concluída dentro de um respectivo intervalo de tempo, designado por janela de tempo, e há punições por atraso e adiantamento no término da tarefa. A qualidade da programação é medida pela soma das penalidades por adiantamentos e atrasos. Trata-se de um problema combinatório, pertencente à classe NP-Difícil, para o qual, no limite, há $(N!)^M$ alternativas. O objetivo do presente trabalho é apresentar procedimentos para resolução desse problema de programação de tarefas.

Na pesquisa bibliográfica sobre problemas de programação de tarefas foram identificadas duas classes de procedimento para resolução:

1. Elaboração do modelo matemático para representação do problema e emprego de técnicas de programação linear mista inteira para obtenção da solução ótima;
2. Desenvolvimento de procedimentos heurísticos para obtenção de soluções aproximadas.

O desenvolvimento da pesquisa seguiu os caminhos acima mencionados, abrindo-se duas frentes de trabalho, que, ao fim, se juntam para construir procedimentos combinados.

O modelo matemático proposto para representar o problema, parcialmente credor de contribuições de Rubén, Funda e Thijs (2006), tem características originais no tipo de função objetivo, e na sua modelagem, e nas restrições para evitar a formação de ciclos de tarefas em uma dada máquina. Para a resolução do modelo matemático, foi utilizado o solver CPLEX.

No desenvolvimento de procedimentos aproximados para resolução do problema, o trabalho teve duas vertentes. Numa delas, foi elaborada uma heurística construtiva, a partir de regras empíricas utilizadas em problemas mais simples, para determinar, para cada máquina, a sequência de atendimento das tarefas. Na outra vertente, optou-se por explorar um subconjunto das soluções do problema. Em lugar de se especificar M sequências de atendimento, uma para cada máquina, os dois procedimentos propostos determinam uma sequência de prioridades para as tarefas. Em cada máquina, ao se concluir o atendimento de uma tarefa seleciona-se da fila a tarefa de acordo com a sequência de prioridades adotada.

Para examinar soluções caracterizadas por uma sequência, foram desenvolvidas duas heurísticas: E/TBE e E/TTabu. A primeira é uma heurística de busca exaustiva que encontra, para problemas de pequeno porte, a solução ótima dentro do subconjunto reduzido de alternativas. A segunda heurística, elaborada de acordo com os conceitos básicos da “meta-heurística” Busca Tabu, se propõe a encontrar uma solução de boa qualidade dentro do subconjunto reduzido de alternativas para uma ampla faixa de porte de problema. A primeira heurística serve para avaliar, para instâncias pequenas do problema, a qualidade relativa da solução gerada pela segunda heurística, dentro do subconjunto reduzido de alternativas. Ainda no caso de instâncias pequenas, a solução da primeira heurística quando comparada à solução ótima do modelo matemático pode revelar as limitações decorrentes da redução do espaço de busca.

Por fim, foram examinados alguns procedimentos combinados, utilizando uma solução inicial gerada por procedimentos aproximados e a estrutura matemática do problema, quer utilizando a estrutura geral de ramificação ou uma heurística de ramificação local.

Foram realizadas algumas séries de ensaios computacionais com problemas diversificados para avaliar o desempenho dos procedimentos propostos, tanto em termos de qualidade de solução quanto do tempo de processamento. As principais conclusões são relatadas

na seção 6.2 e, da análise dos resultados encontrados, surgem sugestões para a continuidade da pesquisa, apresentadas na seção 6.3.

6.2 – Conclusões

O problema de programação de tarefas em ambiente *flow shop* tratado neste trabalho é da classe NP-Difícil, sendo bastante difícil o desenvolvimento de técnicas que sejam eficientes, tanto do ponto de vista da qualidade da solução, quanto do ponto de vista do tempo de processamento, para resolução de instancias maiores. Pelas características particulares do problema tratado, pela modelagem matemática proposta e pelos algoritmos e heurísticas apresentados, o presente trabalho traz contribuições para o estudo de problemas de programação de tarefas em ambiente *flow shop*.

A aplicação do modelo matemático, proposto neste trabalho, está limitada a instâncias menores do problema. Para instâncias maiores, mesmo com longos tempos de processamento, é difícil avaliar a qualidade da solução, pois a distância entre o valor de função objetivo da melhor solução encontrada e a estimativa de limite inferior indicado pelo solver é muito grande. O problema da tabela 5.4, com 10 tarefas e 3 máquinas ilustra esta deficiência do solver para a estimativa do limitante inferior. Com 776 segundos de processamento foi obtida uma solução para a qual a função objetivo assumiu valor 7. Neste momento, o valor do limitante inferior era zero. Após 11330 segundos, o limitante inferior assumiu valor 6,95 e não foi encontrada nenhuma solução melhor.

Para os problemas com uma máquina, o algoritmo E/TCPLEX e as heurísticas E/TBE e E/TTabu perseguem a mesma solução ótima, sendo que, para os problemas de pequeno porte a heurística E/TBE obtém todas as soluções ótimas. Para essa classe de problemas, o E/TCPLEX teve um desempenho fraco quer quando comparado às heurísticas, quer quando comparado ao seu próprio desempenho em problemas com múltiplas máquinas. Como o foco do trabalho era no problema geral com múltiplas máquinas, não houve preocupação em modificar o modelo para o caso de única máquina.

Uma questão central da pesquisa era avaliar a qualidade da substituição da busca de M sequências de tarefas por uma única sequência de prioridades. Um exemplo foi decisivo para comprovar que o “sub-ótimo” (associado a melhor sequência de prioridades) pode, em alguns casos, estar muito distante da solução ótima. De fato no problema da tabela 5.2 com 10 tarefas e 3 máquinas, enquanto a função objetivo da solução ótima do modelo matemático, obtido pelo algoritmo E/TCPLEX, assume valor 7, a função objetivo da solução ótima obtida pela heurística E/TBE assume valor 120. Sendo assim, não basta avaliar o desempenho da heurística E/TTabu comparando os seus resultados com os da heurística E/TBE (o que somente pode ser feito para instâncias pequenas do problema, e nesses casos, o desempenho relativo do E/TTabu foi bom). É necessário também comparar os seus resultados com algum limitante inferior calculado no conjunto completo de soluções do problema, dado que, para instâncias maiores, é muito difícil de obter a solução ótima pelo algoritmo E/TCPLEX. No entanto, o único limitante inferior utilizado neste estudo – aquele fornecido pelo solver CPLEX ao longo do processamento do modelo matemático, revelou-se inadequado, conforme mencionado acima.

A heurística E/THC é uma heurística construtiva, que constrói as sequências baseada apenas em regras empíricas, desta forma fica com seu desempenho limitado. Porém a solução que esta heurística busca está em um espaço de soluções diferente das heurísticas E/TBE e E/TTabu e desse modo pode conseguir resultados melhores que essas outras heurísticas, como foi o caso em alguns dos problemas propostos. Outra vantagem é o seu desempenho em termos de tempo de processamento, o que faz esta heurística ser uma opção interessante para procedimentos combinados.

Com o intuito de melhorar a qualidade dos resultados, foram examinados procedimentos combinados das heurísticas, explorando a estrutura do modelo matemático, tanto usando o Branch and Cut quanto o *Local Branching*. Como o número de testes realizados foi relativamente pequeno, não é possível estabelecer conclusões relevantes.

A partir dos exemplos resolvidos neste trabalho, lembrando que o número de problemas foi pequeno, pode-se dizer que: para problemas com uma única máquina, de pequena instâncias, uma boa alternativa seria o E/TBE, que busca a solução ótima e as enumera em

tempos melhores que o E/TCPLEX; para instâncias maiores a alternativa seria o E/TTabu, lembrando que existe na literatura heurísticas mais eficientes para esta classe de problema. Para problemas com mais máquinas e pequenas instâncias a alternativa interessante seria o algoritmo E/TCPLEX; para instâncias maiores poderiam ser utilizados os procedimentos combinados utilizando a heurística *Local Branching* com pontos de partida vindos da heurística E/THC que tem os tempos de processamentos menores.

O importante é destacar que o estudo aqui descrito abre frentes para futuras pesquisas sobre problemas de programação de tarefas em ambiente *flow shop*. Algumas sugestões são apresentadas na próxima seção.

6.3 – Sugestões

Para outros trabalhos pode ser sugerido que se invista em implementações mais eficientes de algoritmos e heurísticas para resolução do problema proposto, em termos de tempo de processamento. No presente trabalho foi proposta a utilização da heurística *Local Branching*, que já é consagrado na literatura, mas outras heurísticas podem ser testadas, tais como:

- *Relaxation Induced Neighborhood Search – RINS* que pressupõe o conhecimento de uma solução inteira viável, heurística proposta por Danna, Rotheberg e Le Pape (2005) e abordou um problema de programação de *job shop*. Esta heurística explora o subespaço de solução que use a solução do problema de programação linear relaxado a uma solução viável conhecida. As variáveis que tiverem os mesmos valores em ambas soluções são fixadas. Em seguida, é resolvido um problema de MIP desenvolvido por Danna, Rotheberg e Le Pape. A solução obtida, caso viável quanto integralidade, terá definido um novo limitante para a resolução do problema. Este procedimento é iterado até que a solução ótima seja obtida;
- *Variable Neighborhood Search – VNS* proposta por Mladenovic e Hansen (1997), que combina uma busca local com mudanças sistemáticas da estrutura de vizinhança, de forma a escapar de regiões de ótimo local. Esta heurística já foi utilizada em problemas de

programação, como no trabalho de Roshanaei et al (2009), que aborda o problema de programação de *Job shop* considerando tempos de preparação e, como medida de desempenho, o *makespan*.

Estas heurísticas e mesmo outras podem ser implementadas de forma a, talvez, conseguir se eleger uma delas como sendo mais eficiente para problemas de “sequenciação”.

Outra sugestão seria investir na heurística E/THC para melhorar o ponto de partida para algoritmos e heurísticas que utilizam *Branch and Cut*, já que para alguns problemas os resultados foram bem interessantes e no trabalho de Mendes (2007) a combinação de uma heurística construtiva com o *Branch and Cut* gerou bons resultados.

Tendo em vista que uma das principais dificuldades encontradas foi a falta de um bom limitante inferior para avaliar a qualidade das soluções geradas pelos algoritmos e heurísticas propostos, uma sugestão para a continuidade da pesquisa é o desenvolvimento de procedimentos eficientes para obtenção desse limitante. Eventualmente, tal procedimento possa ser encontrado utilizando a técnica de relaxação Lagrangeana.

BIBLIOGRAFIA

Ashour, S. “**Techine notes in economics and mathematical systems**” New Your Springer-Verlag, 133p, 1972.

Baker, R.K., Scudder,G.D., “**Sequencing with earliness and tardiness penalties: A review**”, Operations Research, vol 38, n. 1, pp. 22-36, 1989.

Baker,R.K., “**Sequencing and Scheduling**” by K.R.Baker, Hanover, 1997.

Bazaraa,M.S.; Sherali, H.D.; Shetty, C.M. “**Nonlinear Programming – theory and algorithms**”, John Wiley & Sons. Inc., N. York, 1993.

Biskup, D.; Cheng, T.C.E. “**Multiple-machine scheduling with earliness, tardiness and cost penalties**”, Computers & Operations Research v. 26, issue 1, pp.45-57, 1999.

Converse, A.O. “**Optimization**”, Holt,Rinehart and Winston, Inc. USA, 1970.

Danna, E; Rothberg, E; Le Pape, C “**Exploring relaxation induced neighborhoods to improve MIP solutions**”, Mathematical Programming, Ser. A, 102, p. 71-90, 2005.

Dantzig, G.B. “**Studies in optimization**”, The Mathematical Association of America (Incorporated), USA, 1974.

Dias, J.S.; Barreto, J.M. “**Algoritmo Genético: Inspiração Biológica na Solução de Problemas – Uma Introdução**”, CASNAV, p.105-128, 1998.

Duarte, J. D. “**Modelo de programação robusta para sistemas de produção job shop**” São Paulo, 2001, Tese de Doutorado, Escola Politécnica da Universidade de São Paulo, São Paulo, 2001.

Fischetti, M.; Lodi, A. “**Local Branching**”, Mathematical Programming ser. B, 98, p.23-47, 2003.

Forsythe, A.J.; Keenan, T.A.; Organick, E.J.; Stenberg, W. “**Computer science: A first course**”, John Wiley & Sons, Inc., New York, 1989.

Glover, F.W.; Laguna, M., “**Tabu Search**”, Kluwer Academic Publishers, Boston, 1997.

Goldberg, D. E. “**Genetic Algorithms in Search, Optimization and Machine Learning**”. Addison-wesley, 1989.

Hax, A.; Candea, D. “**Production and inventory Management**”, Englewood Cliffs, N.J., Prentice-Hall, 1984.

Henriques, L.R.S.A “**Heurísticas para o problema de minimização do atraso total no ambiente *flow shop* com *buffer zero***”, São Paulo, 2005, Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo, São Paulo, 2005.

Ignall, E. ; Schrage, L., “ **Application of the Branch –and- bound technique to some flow shop scheduling problems**”, Operations Research, v. 13, pp 400-412, 1965.

James, R.J.W., “**Using Tabu Search to solve the common due data Early/Tardy machine scheduling problem**” Computers Ops. Res., v.24, n.3, pp.199-208, 1996.

Johnson, S.M., “ **Optimal Two- and Three- stage production schedules**” , Naval Research logistics Quarterly, 1954, v. 1, pp 61-68.

Li, G., “ **Single machine earliness and tardiness scheduling**” , European Journal of Operational Research, n. 96, pp. 546-558, 1997.

Liaw, C. “**A branch and bound algorithm for the single machine earliness and tardiness scheduling problem**” Computers & Operations Research, v. 26, pp. 679-693, 1999.

Knuth, D. “**The Art of Computer Programming –Vol. 1: fundamental Algorithms**”, Addison-Wesley Publ. Co., Reading, Massachusetts, EUA, 1973.

Kochhar, S.; Morris, R.; Wong, W. “**The local search approach to flexible flow line scheduling.**” Engineering Costs and Production Economics. v14 i1. 25-37, 1988.

Koulamas, C. “**Single machine scheduling with time windows and earliness/tardiness penalties**” European Journal of Operational Research, v. 91, issue 1, pp. 190-202, 1996.

Kurs, M.E. ; Askin, R.G. “**Comparing scheduling rules for flexible flow lines**”, International Journal of production Economics, v. 85, pp. 371-388, 2003.

Mladenovic, N.; Hansen, P. “**Variable Neighborhood Search**” Computers & Operation Systems and project management, New York, John Wiley & Sons Inc., 695p, 1997.

Mendes, A.B. “**Programação de frota de apoio a operação offshore sujeita à requisição de múltiplas embarcações para uma mesma tarefa**”, São Paulo, 224p. Tese de doutorado, Escola Politécnica da Universidade de São Paulo – Departamento de Engenharia Naval e Oceânica, 2007.

Morton, T.E.; Pentico, D.W. “**Heuristic scheduling systems with applications to production systems and project management**” by John Wiley & Sons Inc. New York, 1993.

Nagano, M.S. **“Novos Procedimentos de Busca Tabu para problemas de programação de Operações Flow Shop Permutacional.”** São Carlos. 117p., Dissertação (Mestrado), Escola de Engenharia de São Carlos, Universidade de São Paulo, 1995.

Nawaz, M.; Ensore, E.E.; Ham, I. **“A heuristic algorithm for the m- machine, n-job flow-shop sequencing problem”**, OMEGA, The international Journal of Management Science, v.11, pp. 91-95, 1983.

Pacheco, R.F.; Santoro, M.C. **“Proposta de classificação hierarquizada dos modelos de solução para o problema de job shop scheduling”** Gestão & Produção, v. 6, n. 1, pp. 1-78, (1999).

Peal, J. **“Heuristics: Intelligent search strategies for computer problem solving”**, Addison-Wesley Publishing Company, New York, 1984.

Pinedo, M., **“Scheduling – theory, algorithms and systems”** Prentice-Hall, Inc. A Simon & Schuster Company Englewood Cliffs, New Jersey, 1995.

Pochet, Y.; Wolsey, L.A. **“Production planning by mixed integer programming.”** New York: Springer, 499p, 2006. (Springer Series in Operations Research and Financial Engineering).

Oliveira, A. C. **“Uma contribuição para o problema de programação de tarefas em job shop dinâmico não determinístico com datas de entrega”** São Paulo, 2002, Tese de Doutorado Escola Politécnica da Universidade São Paulo, São Paulo, 2002.

Rajendran, C. **“Formulations and heuristics for scheduling in a Kanban flow shop to minimize the sum of weighted flowtime, weighted tardiness and weighted earliness of containers”** International Journal of Research, v. 37, n. 5, pp. 1137-1158, 1999.

Roshanaei, V.; Nadari, B.; Jolai, F.; Khalili, M. “**A variable neighborhood search for job shop scheduling with set up times to minimize makespan**” *Future Generation Computer Systems*, 25, pp.654-661, 2009.

Rubén R; Funda S. S; Thijs U. “**Modeling realistic hybrid flexible flow shop scheduling problems**” *Computers and Operations Research*, v. 35 , Issue 4, pp. 1151-1175, 2008.

Sakuraba, C.S, “**Minimização dos desvios absolutos com data de entrega comum no ambiente *flow shop* com duas máquinas**” São Paulo, 2005, Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo, São Paulo, 2005.

Santoro, M.C. “**Modelo de programação para produção intermitente com composição de produtos variável no tempo**” São Paulo, 1982, Tese de Doutorado, Escola Politécnica Da Universidade de São Paulo, São Paulo, 1982.

Swait Jr, J.D. “**Fundamentos computacionais, algoritmos e estruturas de dados**” Makron, McGraw-Hill, São Paulo, 1991.

Wilamwsky, Y.; Epstein, S.; Dickan, B. “**Optimal common due-date with completion time tolerance**” *Computers Operational Research*, v. 23, n. 12, pp. 1203-1210, 1996.

Zhu, Z; Heady, R.B. “**Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach**” *Computer and Industrial Engineering*, v. 38, issue 2, pp. 297-305, 2000.