Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica
Departamento de Engenharia da Computação e Automação Industrial

Seqüenciamento e Alocação de Operações em *Flow-shops* com Restrições sobre os Recursos Compartilhados e sobre os Prazos de Entrega das Tarefas: Uma Abordagem de Busca Orientada por Restrições.

Autor: Márcio Francisco Dutra e Campos

Orientador: Prof. Dr. Luis Gimeno Latre
(DCA-FEE-UNICAMP)

Co-orientador: Profa. Dra. Maria Tereza M. Rodrigues
(DESQ-FEQ-UNICAMP)

Este exemplar corresponde à redação final da tese defendida por MÁRCIO FRANCISCO DUTRA E CAMPOS e aprovada pela Comissão Julgadora em 03 / 08 / 93.

Orientador

Tese apresentada à Faculdade de Engenharia Elétrica da Unicamp (FEE-UNICAMP) como parte dos requisitos exigidos para a obtenção do título de MESTRE em ENGENHARIA ELÉTRICA.

Campinas, agosto de 1993

Agradecimentos:

Ao Prof. Luis Gimeno Latre pela oportunidade e pela confiança que depositou no meu trabalho;

Ao Prof. Gimeno e à Profa. Maria Teresa pela valiosa orientação e pelos ensinamentos;

Ao Carlos Alberto S. Passos, pelas inúmeras discussões durante a realização deste trabalho;

Aos meus pais, pela compreensão, apoio e incentivo;

Ao Irasmo e à Isabel, pelo apoio indispensável que eles sempre me dão;

Aos meus amigos da FEE, especialmente ao Aleandro S. Macedo e ao Bartolomeu F. Uchôa, pelos momentos agradáveis que me proporcionaram;

Ao CNPQ, pelo apoio financeiro.

Resumo

O *flow-shop*, assim como a grande maioria dos problemas de *scheduling*, é um problema cuja complexidade computacional cresce exponencialmente com a sua dimensão. Para diminuir esta complexidade, geralmente são feitas hipóteses que simplificam o modelo da planta mediante o relaxamento de restrições. Grande parte destas hipóteses dizem respeito aos instantes de tempo em que as tarefas devem ser alocadas e às disponibilidades dos recursos compartilhados. Entrentanto, na indústria de processos químicos, que é a principal área de aplicação do *flow-shop*, estas restrições não podem ser relaxadas, porque o alto custo dos produtos que são produzidos exige que as instalações disponíveis sejam utilizadas da melhor maneira possível. Além do mais, o atendimento rápido aos clientes é o principal objetivo, o qual só é alcançado se as restrições temporais impostas pelo processo produtivo e pela demanda do mercado são obedecidas.

Neste trabalho propõe-se uma estratégia de solução para o problema de *flow-shop* que não admite a relaxação das restrições sobre os recursos compartilhados e sobre os instantes de tempo em que as tarefas devem ser alocadas. A estratégia proposta combina técnicas de Inteligência Artificial (IA), que têm por objetivo satisfazer as restrições do problema, com a técnica *Branch-and-Bound* (BAB) clássica da Pesquisa Operacional (PO), que permite alcançar o objetivo do problema que é minimização do tempo de conclusão das tarefas, através da maximização da utilização dos recursos compartilhados. As técnicas de IA utilizadas são a análise e propagação de restrições que, juntamente com heurísticas de alocação de tarefas, permitem uma redução significativa do espaço de soluções e de busca do problema. O uso conjunto do BAB com propagação de restrições e heurísticas resulta em um algoritmo de busca em árvore eficiente, no sentido de que ele geralmente consegue obter soluções ótimas sem precisar pesquisar um elevado número de nós.

ÍNDICE

NOMENCLATURA

BAB: *Branch-and-Bound*;

BH: Busca Heurística;

BP: Busca em profundidade;

BPAR: Busca em profundidade com análise de restrições;

BPM: Busca pelo melhor

BS: *Backtracking Search*;

C^*: Tempo médio de conclusão das tarefas;

C_{ij}: Tempo de conclusão da tarefa i no processador j;

$C_{in,r}$: instante mais cedo que o perfil de disponibilidade do recurso r permite o processamento da operação (i,j);

C_{max}: Tempo de conclusão máximo das tarefas (*makespan*);

CAN: Conjunto de tarefas candidatas;

CHS: *Constraint Heuristic Search*;

CPI: (*Chemical Process Industries*) Indústria de processos químicos;

CRP: *Constraint Relaxable Problems*;

CSP: *Constraint Satisfaction Problems*;

DD_i: (*due date*) prazo de entrega da tarefa i;

DW: (*Demand Window*) Janela de demanda de uma operação;

E_i: (*earliness*) adiantamento da tarefa i;

EBT: (*Earliest Beginning Time*) instante de início mais cedo;

EFT: (*Earliest Finish Time*) instante de término mais cedo;

f: função objetivo que se deseja otimizar;

f^*: limitante superior da função objetivo;

F^*: Tempo médio de residência das tarefas;

F_i: (*flow time*) tempo de residência da tarefa i na planta;

Fmax: Tempo de residência máximo;

FIS: *Finite Intermediate Storage*;

FS: Problema de *Flow-shop*;

FSP: Problema de *Flow-shop* com seqüências de permutação;

I_{ij}: Instante de início da tarefa i no processador j;

IA: Inteligência artificial;

J: Conjunto de tarefas externas não seqüenciadas;

JS: Problema de *Job-shop*;

K: Conjunto de tarefas seqüenciadas;

L^*: Defasagem média das tarefas;

Li: (*lateness*) defasagem da tarefa i;

LB: (*Lower-Bound*) Limitante inferior;

LBT: (*Latest Beginning Time*) último instante de início;

LFT: (*Latest Finish Time*) último instante de término;

M: Número de processadores;

MIS: *Mixed Intermediate Storage*;

MILP: *Mixed Integer Linear Program*

N: Número de tarefas;

NIS: *No Intermediate Storage*;

NS: Número de nós sondados;

OSP: *Over-subscribed Problems*;

PO: Pesquisa operacional;

PS: Problema de *Scheduling*;

q_r: disponibilidade temporal do recurso r;

Q_{ij}^r: demanda temporal do recurso r pela operação (i,j);

R: Número de recursos compartilhados;

RT_i: (*ready time*) tempo de pronto da tarefa i;

SC: Número de soluções completas encontradas;

SP: Seqüências de permutação;

SS: Solução simplificada para o *flow-shop* com alocações externas;

T^*: Atraso médio das tarefas;

T_i: (*tardiness*) atraso da tarefa i;

T_{ij}: intervalo de tempo disponível para processar as operações i e j, supondo que i será processada antes de j;

T_{max}: Atraso máximo das tarefas;

TF: Tempo de conclusão máximo das tarefas;

TP_{ij}: Tempo de processamento da tarefa i no processador j;

TT: Tempo total de CPU (em segundos);

U: Conjunto de tarefas não seqüenciadas;

UIS: *Unlimited Intermediate Storage*;

UB: (*Upper-Bound*) Limitante superior;

W^*: Tempo médio de espera pelas tarefas;

W_{ij}: (*waiting time*) tempo que processador j espera pela tarefa i;

ZW: *Zero Wait*

Capítulo 1

Introdução.

1.1- Objetivo e Motivação do Trabalho.

Objetivo:

O objetivo deste trabalho é estudar o problema de seqüenciamento e alocação temporal de recursos (*scheduling*) em sistemas de produção do tipo *flow-shop*, sujeitos a restrições sobre os instantes em que as tarefas devem ser processadas e sobre as disponibilidades de recursos compartilhados. É proposto um algoritmo do tipo *Branch-and-Bound* que utiliza estas restrições para orientar e guiar a busca pela solução ótima do problema.

Motivação:

O *flow-shop*, assim como a grande maioria dos problemas de *scheduling*, é um problema cuja complexidade computacional cresce exponencialmente com a sua dimensão. Para diminuir esta complexidade, geralmente são feitas hipóteses que simplificam o modelo da planta mediante o relaxamento de restrições. Grande parte destas hipóteses dizem respeito aos instantes de tempo em que as tarefas devem ser alocadas e às disponibilidades de recursos compartilhados. No entanto, na indústria de processos químicos, que é a principal área de aplicação do *flow-shop*, estas restrições não podem ser relaxadas. Isto porque existe uma necessidade muito grande de maximizar a utilização das instalações disponíveis dado o alto custo dos produtos produzidos nestas indústrias [Dudek-92]. Além do mais, o atendimento rápido aos clientes é o principal objetivo, o qual só é alcançado se as restrições temporais impostas pelo processo produtivo e pela demanda do mercado são obedecidas.

Existem duas perspectivas de solução de um problema *scheduling*: uma orientada a otimizar algum critério de desempenho da planta (geralmente uma função dos tempos de processamento das operações), e outra orientada a satisfazer as suas restrições. O *Branch-and-Bound* (BAB) [Horowitz-78] é o algoritmo que tem produzido os resultados mais notáveis utilizando a primeira perspectiva. No entanto, este algoritmo é impraticável para problemas de dimensões grandes ou mesmo para problemas de dimensões pequenas onde o objetivo é satisfazer um elevado número de restrições. Nestes casos a melhor estratégia é utilizar um algoritmo de busca heurística guiada por restrições (*constraint heuristic search* (CHS) [Fox-83]). Esta estratégia, por sua vez, não é eficiente quando se deseja otimizar algum critério de desempenho.

A motivação deste trabalho, portanto, está na solução de um problema prático, que tem por objetivos otimizar o desempenho da planta e satisfazer as suas restrições, utilizando uma estratégia que

combina uma ferramenta sofisticada de otimização (o BAB) com uma ferramenta adequada para a satisfação de restrições (CHS). A utilização conjunta destas duas ferramentas visa resolver o problema de forma eficiente, pois uma complementa a outra.

1.2- O *Flow-shop* na Indústria Química.

Embora o modo de operação contínuo prevaleça e seja desejável em muitos processos químicos, o processamento em batelada (*batch processing*) é ainda muito importante e tem um lugar assegurado na indústria de processos químicos (CPI). A principal característica das plantas em batelada é sua flexibilidade para produzir múltiplos produtos através do compartilhamento dos equipamentos. No entanto, para que essa flexibilidade possa ser bem aproveitada são necessárias ferramentas de planejamento sofisticadas que permitam tirar o máximo proveito dos recursos compartilhados.

O processamento em batelada é economicamente desejável na produção de uma pequena quantidade de produtos de alto custo ou de uma grande quantidade de produtos que possuem uma mesma rota de produção. O seu uso ou não é influenciado ainda por forças de mercado, tal como a demanda sazonal por produtos, e por problemas operacionais, tal como a necessidade freqüente de limpeza de equipamentos [Das-90]. Quanto à demanda, se existem previsões seguras de demanda a longo prazo, então a planta pode ser operada no modo "campanha" (*campaign*), com todos os seus recursos dedicados a um pequeno subconjunto de produtos por longos períodos de tempo (campanhas) [Kondili-93]. Assim o tempo de estabelecimento dos produtos (*changeover*) pode ser minimizado facilitando o gerenciamento e o controle da planta. Além do mais, um padrão cíclico de operações pode ser estabelecido durante cada campanha, com muitas bateladas idênticas do mesmo produto sendo produzidas em seqüência. Por outro lado, se não existe uma previsão confiável de demanda, então a produção da planta é definida pela chegada de pedidos. Isto leva a um horizonte de planejamento pequeno, sobre o qual nenhum padrão de operações pode ser estabelecido, dando origem ao problema de *scheduling* de curto prazo.

Com respeito ao planejamento da planta, sobre um dado horizonte de tempo, faz-se a distinção entre planejamento de alto-nível e planejamento de baixo-nível.

O planejamento de alto-nível decide o número de bateladas de produtos diferentes que precisam ser produzidas para atender as necessidades da produção, o que consiste em especificar os seguintes elementos: i) o conjunto N de tarefas a serem processadas; ii) o conjunto M de unidades de processamento disponíveis; iii) os tempos de processamento de cada operação de cada tarefa; iv) os tempos de transferência de cada tarefa para cada processador; v) os tempos de preparação das unidades de processamento; vi) o critério de desempenho que será otimizado; e vii) um conjunto de regras que governam o processo de produção, incluindo a ordem na qual as operações devem ser realizadas para cada tarefa e a política de armazenagem intermediária utilizada.

De posse destes dados, o planejamento de baixo-nível determina o instante em que cada tarefa deve ser realizada de forma a otimizar o critério de desempenho desejado pois, uma vez que o tempo e os recursos são compartilhados pelas tarefas, é necessário alocar as operações de forma a utilizar as instalações da maneira mais eficiente possível . Os critérios de desempenho geralmente utilizados são: i)minimização do tempo de conclusão total das tarefas (*makespan*); ii) minimização do tempo médio de residência das tarefas na planta; e iii) minimização do atraso máximo ou do atraso médio das tarefas. Estes critérios sofrem a influência das restrições do processo e da política de armazenagem intermediária adotada. O problema geralmente é considerado determinístico e estático.

Na indústria química prevalecem os sistemas com unidades de processamento seriais, como por exemplo na indústria de alimentos, farmacêutica e de polímeros. No caso mais simples, que será objeto de estudo deste trabalho, cada estágio da planta possui uma única unidade de processamento, que é capaz de processar uma única tarefa por vez, impossibilitando o processamento paralelo de tarefas. O fluxo das tarefas através dos diferentes estágios da planta só pode ocorrer em série e, uma vez iniciado o processamento de um produto (tarefa) por um equipamento, este fica dedicado àquele produto até que o seu processamento seja completado (processamento não preemptivo).

As plantas não-contínuas podem se dividir em:
-plantas multiproduto: são aquelas plantas em que em que todos os produtos têm a mesma seqüência de operações (*flow-shop*), geralmente porque eles pertencem a uma mesma família;
-plantas multipropósito: são aquelas em que produtos diferentes podem ter seqüências de processamento diferentes (*job-shop*).

Grande parte das plantas não-contínuas têm uma estrutura do tipo *flow-shop*. De fato, a indústria química tem mostrado ser a principal área de aplicação do *flow-shop*, e a responsável por grande parte dos desenvolvimentos recentes nesta área. A estrutura *job-shop* é geralmente reservada para aqueles casos especiais em que se deseja processar poucos produtos e que envolvem procedimentos de síntese que são ou complexos, ou incertos, ou que possuem baixa produção ou demanda incerta [Rodrigues-92]. Existem também indústrias em que o processamento paralelo de tarefas é utilizado [Musier-88, Kuryan-88, Rajendran-90], cada estágio da planta podendo conter mais de uma unidade de processamento do mesmo tipo.

A Figura 1.1 mostra a configuração básica de uma planta não-contínua multiproduto serial típica de uma CPI. Como mostra a figura, ela consiste de uma série de unidades em batelada e unidades semi-contínuas com unidades de armazenagem separando algumas delas. As unidades em batelada são aquelas que operam em regime descontínuo e são compostas de reatores, tanques, secadores, etc. As unidades semi-contínuas são aquelas que operam em regime contínuo mas com freqüentes paradas, e são compostas de bombas, trocadores de calor, filtros, etc.

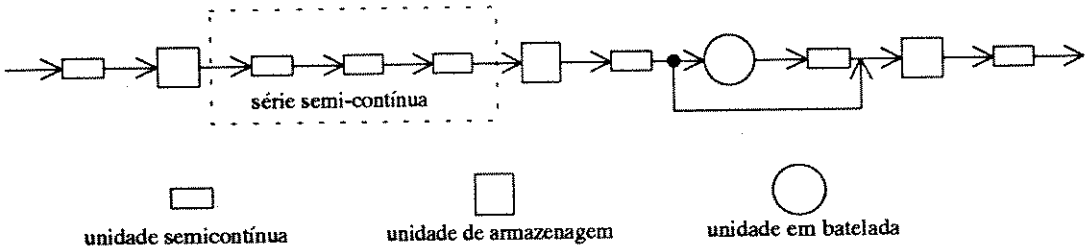


Figura 1.1- Processo serial não-contínuo.

As unidades semi-contínuas são freqüentemente dispostas em série para operarem simultaneamente. Assim, uma batelada pode ser bombeada de um tanque para um cristalizador, deste para um filtro e finalmente para outro tanque. Desta forma, três unidades semi-contínuas (bomba, cristalizador e filtro) operam simultaneamente em série. Uma série de unidades semi-contínuas como esta, sem nenhuma armazenagem intermediária entre elas é, do ponto de vista de seqüenciamento, equivalente a uma única unidade semi-contínua operando na menor das taxas das unidades individuais (isto porque o material deve fluir continuamente através de todas unidades da série [Rajagopalan-89]).

1.3- Organização do Trabalho.

Este trabalho está organizado da seguinte forma:

-no capítulo 2 faz-se uma descrição geral do problema de *scheduling*, abordando os seguintes aspectos: a caracterização dos componentes do problema; os objetivos do *scheduling*; as classificações dos problemas de *scheduling*; os tipos de restrições presentes e as hipóteses feitas para simplificar o problema. Apresenta-se o problema de *flow-shop* dando ênfase à sua aplicação na indústria de processos químicos, considerando restrições sobre os instantes de alocação das tarefas e sobre a disponibilidade dos recursos compartilhados;

-no capítulo 3 faz-se uma pequena revisão bibliográfica das técnicas utilizadas para resolver os problemas de *scheduling*. É dada ênfase a uma técnica da Inteligência Artificial denominada *Constraint Satisfaction Problems*, que busca a solução do problema de *scheduling* focalizando as sua restrições, e a uma técnica da Pesquisa Operacional denominada *Branch-and-Bound*, que busca o solução do problema focalizando os seus objetivos. Propõe-se o uso conjunto destas duas técnicas para solucionar o problema de *flow-shop* apresentado no capítulo 2;

-no capítulo 4 formaliza-se o problema de *flow-shop* com restrições sobre os instantes de alocação das tarefas e sobre os recursos compartilhados. Propõe-se um algoritmo do tipo BAB com busca-em-profundidade que procura a solução ótima do problema considerando não só o seu objetivo como também a satisfação das suas restrições. O algoritmo usa vários conceitos apresentados no capítulo 3,

como análise de restrições baseada em janelas de tempo das operações, criticalidade de tarefas e maximização da utilização dos recursos compartilhados.

-no capítulo 5 realiza-se um conjunto de simulações que ilustram o desempenho do algoritmo. O algoritmo é aplicado a três situações diferentes: 1) em problemas de *flow-shop* com restrições sobre o instante de alocação das tarefas e sobre recursos compartilhados, 2) em problemas de *flow-shop* com restrições apenas sobre recursos compartilhados e 3) em problemas de *flow-shop* com restrições sobre o uso de processadores.

-finalmente apresenta-se as conclusões relativas ao trabalho e faz-se sugestões para trabalhos futuros.

Capítulo 2

Caracterização do Problema de *Scheduling*.

2.1- Introdução.

Neste capítulo faz-se uma descrição geral do problema de *scheduling*. Os componentes principais do problema - produtos, processadores e recursos - são caracterizados em termos de suas necessidades, capacidades e disponibilidades, respectivamente.

Os problemas de *scheduling* são classificados de acordo com: a estrutura de processamento e critérios de otimização utilizados, a origem e instantes de chegada dos pedidos, os tipos de restrições presentes e, os estados inicial e final da planta. Estes atributos permitem identificar as características significativas dos diversos problemas, o que contribui para a sua solução eficiente.

Dentro deste contexto apresenta-se o problema de *flow-shop*, cuja principal área de aplicação é a indústria de processos químicos. Devido a complexidade computacional deste problema, são feitas na literatura corrente algumas hipóteses simplificadoras com relação às tarefas, aos processadores e aos tempos de processamento. Entretanto, o problema resultante, embora mais simples, fica distante dos problemas encontrados na prática. Propõe-se, então, um problema de *flow-shop* mais realista, contendo restrições que normalmente não são consideradas na literatura, que são as restrições sobre o instante de alocação das tarefas e restrições sobre os recursos compartilhados.

2.2- Descrição Geral do Problema de *Scheduling* (PS).

Define-se o problema de *scheduling* (PS) [Baker-74] como a alocação temporal de recursos para realizar um conjunto de tarefas, que são as atividades que se deseja executar a partir dos recursos disponíveis.

O *scheduling* corresponde à fase de operação e execução das tarefas. Antes desta fase certamente terão existido uma ou várias outras fases, que aqui podem se caracterizar de planejamento, onde foram definidas as tarefas passíveis de serem executadas e o conjunto de recursos alocados para a sua execução. A fronteira entre planejamento e *scheduling* não é precisa [Rickel-88] e em geral estas duas fases são interdependentes.

A princípio, considera-se que o PS parte de um conjunto de recursos definido e de um conjunto de tarefas que são realizáveis com aqueles recursos. Num meio industrial isto corresponde a i) um conjunto de processadores (máquinas); ii) os recursos, denominados recursos comuns, utilizados

durante o seu funcionamento (matérias primas, utilidades, mão-de-obra, etc); e, iii) os produtos que podem ser fabricados (tarefas) a partir dos meios definidos em i) e ii).

O processamento de uma tarefa em um processador é definido como uma operação.

2.2.1- Caracterização dos Componentes do Problema.

Os processadores são caracterizados pela sua função, capacidade, quantidade e, se for o caso, pelo consumo de recursos inerente ao seu funcionamento. Quanto à função, os processadores podem ter a mesma função, ter a mesma função mas com velocidades diferentes ou ter função e velocidades diferentes [Coffman-76].

Os recursos comuns são caracterizados pelo tipo e disponibilidade (dada em unidades de recurso por unidade de tempo- ur/ut). Nas plantas industriais, de acordo com a sua disponibilidade, os recursos comuns podem se dividir em: consumíveis, renováveis e substituíveis. Os recursos consumíveis são aqueles para os quais existe um estoque inicial disponível, que pode ou não ser renovado em intervalos de tempo, e que são consumidos de acordo com as necessidades do processo. O capital é um exemplo típico deste tipo de recurso. Os recursos renováveis são aqueles que estão disponíveis em quantidade limitada em cada unidade de tempo, mas sem qualquer limitação de tempo para seu uso, como p.ex. a energia elétrica e o vapor, e são denominados recursos compartilhados. Os processadores também são recursos renováveis, cuja disponibilidade é igual a 1ur/ut. Os recursos substituíveis são aqueles que ao serem consumidos geram outros recursos, que também são chamados de substituíveis. Por exemplo, capital pode ser convertido em combustível. Uma descrição detalhada destes tipos de recursos pode ser encontrada em [Currie-91, Patterson-89, Toker-91].

Os produtos são caracterizados pelo volume de matéria-prima necessário à sua realização e pelo processo de fabricação. Este processo de fabricação, para fins de *scheduling*, deve descrever cada uma das operações realizadas em cada um dos processadores em termos de i) duração, ii) demanda de recursos e iii) outras características da operação que se realizarão sobre o produto de entrada que sejam relevantes para o problema de encadeamento de operações (p.ex., instabilidade do produto).

Modelo da Planta:

Dado que as técnicas de *scheduling* tem como objetivo obter uma solução do problema trabalhando sobre um modelo matemático do sistema real, o primeiro passo, e fundamental, será sempre o de identificar os aspectos relevantes a serem incluídos no modelo. Estes aspectos "relevantes" dependem só da precisão com que se quer resolver o problema e determinam a precisão requerida para o modelo. No caso do PS, a questão da escala de tempo tem esta característica: deve-se definir de início a duração mínima dos eventos, abaixo da qual estes não precisam ser individualizados, ou seja, podem ser

agregados a outros eventos/operações consecutivas ou precedentes sem perda significativa de qualidade da solução. Obviamente a solução não é tão simples; p.ex., a duração de uma operação pode ser pequena mas o seu consumo de recursos comuns pode ser muito grande, causando um impacto grande sobre a solução final.

Existem três modelos básicos para representar a estrutura fundamental de um problema de *scheduling*: o modelo algébrico, o modelo de programação matemática e o modelo de redes (ou grafos disjuntivos). O modelo algébrico é o mais abstrato e usa a teoria de conjuntos e conceitos de álgebra moderna; o modelo de programação matemática é mais acessível pois usa a terminologia e as idéias da teoria de otimização; o modelo de redes, no entanto, é o mais fácil de se usar no desenvolvimento de algoritmos e estratégias de solução e, por isto, tem sido o modelo escolhido em muitos tratamentos recentes de problemas de *scheduling* [Adams-88, Carlier-89, Karabati-92, Van Laarhoven-92]. Maiores detalhes destes modelos podem ser encontrados em [Rogers-91].

2.2.2- O Objetivo do *Scheduling*.

O problema de *scheduling* aparece porque se quer otimizar de alguma forma a capacidade instalada de produção, ou seja, tem-se o objetivo de maximizar o benefício econômico obtido com a instalação disponível. Este benefício resulta da utilização máxima da planta e da fabricação dos produtos requeridos pelo mercado. Este último aspecto se traduz no PS no objetivo de fabricar determinada quantidade de produto até um determinado instante de tempo. A quantidade de produto será transformada num certo número de repetições de uma tarefa, com produção unitária definida pelos equipamentos (corridas de um conversor de aço, bateladas de um reator, etc.). O processamento daquela tarefa, ou de uma das repetições, terá assim um tempo especificado para ser concluído ou um horizonte de tempo dentro do qual o processamento deve ser efetuado.

Quando os únicos recursos utilizados são os processadores, ou os recursos comuns possuem disponibilidade ilimitada, o PS é transformado num problema de seqüenciamento, ou seja, trata-se de obter a seqüência com que as operações de cada tarefa devem ser executadas em cada processador de forma a otimizar o desempenho da planta. O problema de alocação, ou seja, a definição do instante início da execução de cada operação é trivial, pois será o instante de tempo em que o processador for liberado pela operação que o ocupa, ou o instante em que a operação precedente for finalizada no processador anterior. Se existem limitações em qualquer outro tipo de recurso utilizado, como por exemplo transporte, armazenagem ou recursos compartilhados por vários processadores, o tempo inicial dependerá também da disponibilidade destes recursos. Esta disponibilidade, por sua vez, é função das operações que estão sendo executadas simultaneamente em outros processadores (utilidades, mão-de-obra, etc.) ou de decisões de alocação tomadas anteriormente (transporte ocupado pelo término de uma operação anterior). Neste caso não é possível a separação do problema em seqüenciamento e alocação temporal, tratando-se de um problema de *scheduling* propriamente dito.

A construção de um uma solução factível (*schedule*) é portanto um problema de tomada de decisão, que pode se dividir em [Erschler-89]:

-decisões de atribuição, que seleciona de um conjunto de recursos aqueles necessários ao processamento de uma tarefa. Estas decisões estão relacionadas à disponibilidade dos recursos e fazem parte da fase de planejamento;

-decisões de alocação e seqüenciamento (ou decisões de *scheduling*), que alocam e seqüenciam as operações que estão em conflito pela utilização dos recursos.

A solução ótima de um PS é extremamente onerosa do ponto de vista computacional. Isto leva a se procurar soluções "razoáveis", freqüentemente através de heurísticas que impliquem num tempo de cálculo aceitável. O ponto importante é que não se trata de gastar menos tempo em todas as decisões de *scheduling*, mas em concentrar este tempo nas decisões mais relevantes, aquelas que terão um impacto maior sobre a solução. O problema deve ser analisado de forma a evidenciar estas características mais relevantes. Este problema se coloca junto com o problema de escolher o nível de abstração com que vai se modelar a planta, que foi comentado anteriormente. Poderá ser interessante resolver o PS com vários níveis de abstração, os níveis superiores trabalhando com modelos mais agregados limitando o espaço de busca em níveis inferiores [Rickel-88, Ungar-90, Passos-93].

A forma mais comum de mostrar o resultado do *scheduling* é a Carta de Gantt. Esta carta é uma representação gráfica temporal do *schedule* que mostra os instantes de processamento de cada tarefa em cada processador (Figura 2.1). Com esta representação é possível avaliar alguns critérios de desempenho como o tempo total de fluxo das tarefas, o tempo ocioso dos processadores e o tempo de espera de uma tarefa. A carta, no entanto, é apenas um mecanismo de auxílio ao programador da produção (*scheduler*), já que ela não permite por si só a construção e revisão do *schedule*. Por exemplo, ela não é capaz de mostrar as inter-conexões (ou inter-dependências) entre as diversas tarefas. Assim não se é possível deduzir o efeito que o atraso de uma tarefa pode causar nas demais.

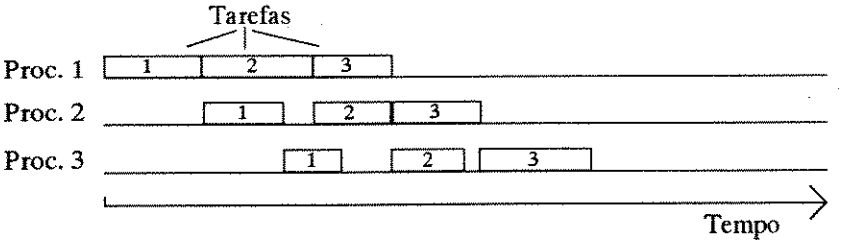


Figura 2.1- Carta de Gantt para um PS com 3 processadores e 3 tarefas.

2.3- Classificação dos Problemas de *Scheduling*.

O objetivo de classificar um problema [Baumgartner-91] é o de identificar as suas características significativas, pois elas contribuirão para a sua solução eficiente, e determinar as relações entre os diversos problemas, pois a solução de um problema pode indicar a solução para um problema correlato. Por outro lado, se um problema não tem solução, isto pode indicar que um problema correlato também não tem.

Existem diversos critérios na literatura para organizar e classificar os PS [Baumgartner-91, Graves-81, Smith-92]. Dentre estes critérios serão analisados alguns com o objetivo de permitir a discussão das técnicas de soluções aplicáveis. Os critérios, analisados nas seções abaixo, são: a estrutura de processamento; o critério de otimização; os tipos de restrições presentes; a origem dos pedidos; os instantes de chegada dos pedidos; e os estados inicial e final da planta.

2.3.1- Estruturas de Processamento.

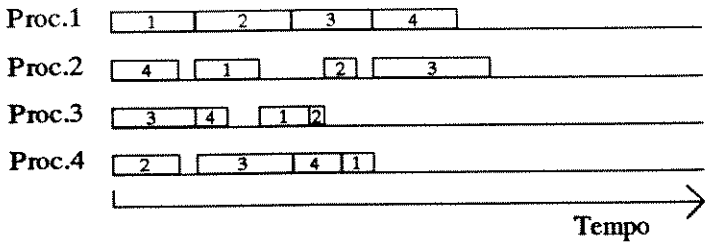
Uma primeira diferenciação entre plantas industriais, do ponto de vista de *scheduling*, é entre plantas que operam em regime de produção contínuo e descontínuo. De fato, é mais correto falar em unidades do que em plantas, porque em uma planta industrial podem coexistir unidades com diferentes formas de operação.

Em uma unidade contínua o PS só aparece se devem ser feitas mudanças nas características dos produtos que são processados continuamente pela unidade, ou, inversamente, se para não haver mudanças nos produtos finais, devem ser implementadas modificações nas condições de processamento de forma a eliminar os efeitos das perturbações. O PS em uma unidade contínua é, ao mesmo tempo, mais simples e mais complexo do que em uma unidade descontínua. Mais simples porque a unidade tem menos graus de liberdade, já que, de fato, ela pode ser considerada como um único processador; e mais complexa porque a "operação" executada neste "único processador" será em geral mais complexa, com um tempo de processamento grande e, mais ainda, com um tempo de resposta grande. As mudanças na operação desta unidade terão sem dúvida um impacto grande sobre o desempenho global da planta. As unidades que operam em modo contínuo mas com freqüentes paradas são denominadas semi-contínuas.

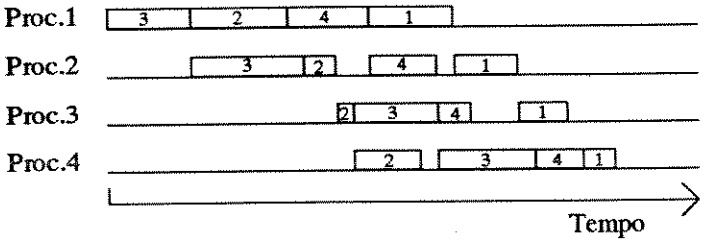
As unidades descontínuas, em termos de estrutura de processamento, podem ser classificadas em *flow-shops*, *job-shops* e *open-shops*. A Figura 2.2 mostra, através da carta de Gantt, as diferenças entre o *job-shop* (JS), o *flow-shop* (FS) e o *flow-shop* com seqüências de permutação (FSP) para uma planta com quatro tarefas e quatro processadores.

Um *flow-shop* é caracterizado por N tarefas e M processadores, sendo que todas as tarefas passam por todos os processadores na mesma ordem. Um caso particular de *flow-shop*, especialmente

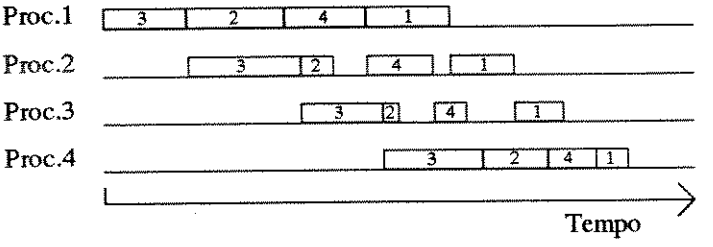
importante na indústria química (onde os processadores estão interligados fisicamente e as N tarefas, produtos finais ou intermediários, pertencem a uma mesma família), é o *flow-shop* com seqüências de permutação, o qual exige que a solução do problema de *scheduling* seja tal que a ordem em cada processador das operações pertencentes as diversas tarefas seja a mesma. Isto faz com que a solução do problema possa ser dada em termos de seqüência de tarefas, não sendo necessário especificar a seqüência de operações para cada processador (porque é a mesma para todos eles).



(a) *Job-shop* (JS)



(b) *Flow-shop* (FS)



(c) *Flow-shop* com seqüências de permutação (FSP)

Figura 2.2- Estruturas de processamento de um PS.

Um *job-shop* é caracterizado por N tarefas e M processadores, cada tarefa podendo utilizar um subconjunto específico de processadores com rotas diferentes para cada uma delas. A multiplicidade de rotas faz com que o transporte de material entre processadores tenha que ser fácil, o que não é geralmente o caso na indústria química. Se o *job-shop* não especifica as rotas das tarefas, então o que se tem é uma estrutura mais geral denominada *open-shop*. No *open-shop* a determinação da rota das tarefas faz parte do processo de solução do problema [French-82, Potts-92].

O *flow-shop*, o *job-shop* e o *open-shop* são problemas de múltiplos estágios, cada estágio correspondendo a um dos M processadores de tipos diferentes. Além do mais, em qualquer uma destas estruturas pode existir mais do que um processador do mesmo tipo, em cada estágio, para a execução de uma operação. Fala-se neste caso de *job-shop (flow-shop)* generalizado [Baker-74, Kuriyan-87].

2.3.2- Critérios de Otimização.

O objetivo do *scheduling* é a otimização do desempenho econômico da planta. Seria desejável que o problema pudesse ser formulado matematicamente como o de otimização de um critério com restrições sobre as variáveis do sistema. Em geral esta não é a situação e o que se tem é um problema de otimização multiobjetivo.

Os seguintes termos são utilizados para definir os critérios de desempenho mais encontrados na literatura:

-TP_{ij}: tempo de processamento da tarefa i (i=1,2,...,N) no processador j (j=1,2,...,M);

-RT_i: (*ready time* ou tempo de pronto), instante em que a tarefa i pode iniciar o seu processamento;

-DD_i: (*due date* ou prazo de entrega), instante em que a tarefa i deve estar terminada;

-C_{ij}: (*completion time* ou tempo de conclusão), instante em que a tarefa i é concluída no processador j. Assim, C_{iM} é o instante em que a tarefa i é concluída e pode deixar a planta;

-F_i: (*flow time* ou tempo de residência), quantidade de tempo em que a tarefa i permanece na planta, ou seja, $F_i = C_{iM}-RT_i$;

-L_i: (*lateness* ou defasagem), quantidade de tempo que excede a data desejada para o término da tarefa, ou seja, $L_i = C_{iM}-DD_i$;

-T_i: (*tardiness* ou atraso) é definido como o máximo entre L_i e 0.

-E_i: (*earliness*) é definido como o abs(mínimo entre L_i e 0).

-W_{ij}: (*waiting time* ou tempo de espera) quantidade de tempo que o processador j espera a tarefa i após sua conclusão no processador (j-1), ou seja, $W_{ij} = (C_{ij} - TP_{ij}) - C_{i(j-1)}$.

-W_i: (*total waiting time* ou tempo total de espera), quantidade total de tempo que a planta espera por uma tarefa, ou seja,

$$W_i = \sum_{j=1}^{M} W_{ij}$$

A partir destes indicadores, definem-se na literatura os seguintes critérios de desempenho para a planta:

a) Tempo médio de residência (F*): $$F^* = \frac{1}{N}\sum_{i=1}^{N} F_i$$

b) Tempo de conclusão médio (C*): $C^* = \frac{1}{N}\sum_{i=1}^{N} C_{iM}$

c) Atraso médio (T*): $T^* = \frac{1}{N}\sum_{i=1}^{N} T_i$

d) Defasagem média (L*): $L^* = \frac{1}{N}\sum_{i=1}^{N} L_i$

e) Tempo total médio de espera por uma tarefa (W*): $W^* = \frac{1}{N}\sum_{i=1}^{N} W_i$

f) Tempo de residência máximo (F_{max}): $F_{max} = \max_{1 \le i \le N} \{F_i\}$

g) *Makespan* (C_{max}): $C_{max} = \max_{1 \le i \le N} \{C_{iM}\}$

h) Atraso máximo (T_{max}): $T_{max} = \max_{1 \le i \le N} \{T_i\}$

Todas estas medidas de desempenho da planta são chamadas de regulares porque:
i) o objetivo do seqüenciamento é minimizar $Z=f(C_{1M},C_{2M},...,C_{NM})$,
ii) Z aumenta apenas se o tempo de término das tarefas também aumenta.

Quando a função objetivo não é uma função crescente dos tempos de conclusão das tarefas, ou seja, não é uma medida de desempenho regular, o problema de otimização torna-se mais difícil. Este é o caso quando o custo associado ao sistema inclui aspectos do seguinte tipo:
a) número de tarefas atrasadas;
b) número médio de tarefas no sistema;
c) evitar tarefas adiantadas, porque pode implicar em custos de estocagem;
d) tempo médio que uma tarefa espera pela liberação de um processador (*mean queue time*);
e) custo operacional do sistema;
f) custos (multas) decorrentes da violação dos prazos de entrega das tarefas.

Alguns destes fatores podem não ter uma representação matemática em termos de tempo de conclusão das tarefas e será difícil, se não impossível, agregá-los a um critério de custo temporal como os descritos anteriormente. Uma alternativa para este problema é incluir alguns fatores no critério de otimização e tratar os outros como restrições a serem satisfeitas pela solução do problema.

As seqüências ótimas para uma dada função objetivo não são necessariamente ótimas para outras funções. Entretanto, existe uma classe de funções que levam à mesma seqüência ótima. De fato,

em um problema de seqüenciamento, uma seqüência ótima para uma das funções objetivo do conjunto {W*,C*,F*,L*} é também ótima para as outras funções deste conjunto [Bellman 82].

2.3.3- Tipos de Restrições Presentes.

Em qualquer problema real de *scheduling* estarão presentes um conjunto de restrições que devem ser satisfeitas. Como é enfatizado em [Rickel-88], é extremamente importante, para limitar o espaço de busca das soluções possíveis, identificar, representar e utilizar correta e eficientemente estas restrições.

Em [Fox-83] citam-se cinco problemas básicos quando se trabalha com restrições: o relaxamento, a importância relativa, a interação entre elas, as condições sob as quais as restrições se aplicam e a sua geração e propagação. Alguns destes problemas serão vistos com detalhes nos capítulos 3 e 4.

Dentre as restrições normalmente encontradas tem-se: i) restrições físicas, como restrições sobre a quantidade de recursos comuns disponíveis, restrições em forma de relações de precedência entre operações, tempos de preparação de processadores, capacidade de armazenagem, etc; e, ii) restrições impostas pela forma como a planta é operada, como por exemplo: restrições sobre datas de entrega de pedidos, restrições sobre estoques intermediários, etc.

Nem todas as restrições terão a mesma importância, algumas deverão ser satisfeitas sempre, (*hard constraints* - restrições rígidas ou restrições de factibilidade), enquanto que outras poderão ser relaxadas (*soft constraints* - restrições flexíveis), normalmente com um certo custo. As restrições de factibilidade que reduzem a priori o espaço de soluções factíveis são denominadas estáticas, e aquelas que geram conflitos no instante em que deve ser tomada uma decisão de alocação são denominadas dinâmicas.

Uma restrição estática sempre presente nos PS são as relações de precedência entre operações de uma mesma tarefa, representando o encadeamento entre as fases sucessivas do processo de fabricação do produto/tarefa (seqüência tecnológica da tarefa). Este tipo de restrição complica a formulação matemática do modelo da planta mas, como já foi dito, tem o efeito de diminuir a dimensão do espaço de soluções do problema.

Em um PS real geralmente existirão restrições dinâmicas, p.ex., restrições sobre o volume disponível de recursos compartilhados. Como exemplos típicos de recursos compartilhados tem-se utilidades e mão-de-obra. Como já foi dito anteriormente, os processadores também são recursos compartilhados. A diferença entre os processadores e os demais recursos compartilhados pelas tarefas e operações é que estes últimos são necessários quando o processador é utilizado, e a utilização do processador só é possível, mesmo estando livre, se o demanda de recursos necessários está disponível

ininterruptamente. Em outras palavras, quando existem restrições sobre os recursos compartilhados a utilização de um processador livre depende do consumo destes recursos pelas operações que neste instante de tempo estão sendo executadas em outros processadores.

Quando se consideram restrições sobre a oferta de recursos compartilhados, os problemas de *scheduling* podem ser divididos em duas classes [Smith-92]: *constraint relaxable problems* (CRP) e *over-subscribed problems* (OSP). Esta divisão se baseia na possibilidade de relaxar ou não as restrições sobre o prazo de entrega das tarefas e as restrições sobre o limite dos recursos.

Nos *constraint relaxable problems* uma destas duas restrições pode ser relaxada, servindo apenas como indicador para a discriminação de duas ou mais soluções de mesmo custo. Dependendo de qual restrição pode ser relaxada, os problemas desta classe podem se dividir em dois grupos [Harhalakis-89]:

1- *scheduling* com restrição rígida sobre o tempo, onde o tempo de pronto e o prazo de entrega das tarefas, determinados por algum processo de análise temporal ou impostos externamente pelo usuário, não podem ser violados. No entanto o consumo de um recurso num certo intervalo de tempo pode ser maior que a sua disponibilidade, o que será resolvido com o relaxamento da oferta do recurso, mediante pagamento de uma multa, ou enviando pedidos para serem processados em outras unidades da indústria.

2- *scheduling* com restrição rígida sobre os recursos, onde a demanda de cada recurso não pode exceder a oferta em nenhuma circunstância. As restrições de tempo podem ser relaxadas e o problema então é o de minimizar o tempo total necessário satisfazendo as ofertas dos recursos. Sempre que a demanda por recursos em um intervalo de tempo não puder ser satisfeita, ocorrerá imediatamente um aumento no tempo total do *scheduling*.

Nos *over-subscribed problems*, nem as restrições sobre o tempo, nem as restrições sobre os recursos podem ser relaxadas. No horizonte de tempo finito em que deve se fazer o *scheduling* poderá haver mais demanda por recursos do que a disponibilidade destes. O objetivo neste tipo de problema é realizar o maior número possível de tarefas maximizando a utilização dos recursos compartilhados.

A utilização dos recursos renováveis e consumíveis podem ainda afetar o tempo de processamento das operações, dando origem aos problemas de compromisso tempo-custo (*time-cost tradeoff problems*) [Baker-74]. Nestes problemas os recursos podem ser alocados a uma operação em qualquer quantidade, sendo que uma maior alocação tem o efeito de reduzir o seu tempo de processamento. Embora os tempos de processamento das operações não sejam mais fixos, o programador da produção tem informação completa e controle sobre eles, e o problema portanto é ainda considerado determinístico.

2.3.4- Origem dos Pedidos.

Em [Graves-81] propõe-se separar os PS, em função da origem dos pedidos de produção que a planta recebe, em: i) *closed-shop*, quando as tarefas a serem executadas têm a sua origem na política de manutenção de estoques finais, com o atendimento dos pedidos feito a partir do estoque; e, ii) *open-shop*, quando os pedidos definem diretamente as tarefas a serem executadas, não existindo estoque. Os casos reais situam-se entre estes dois extremos.

Nota-se que na classificação dos PS o termo *open-shop* é utilizado para especificar a estrutura de processamento e também a origem dos pedidos, não havendo nenhuma relação entre elas. Isto não causa nenhuma confusão porque o contexto sempre permite distinguir a que se refere o termo.

2.3.5- Instantes de Chegada dos Pedidos.

Os pedidos de produção que a planta recebe, qualquer que seja a sua origem, podem ser considerados como perfeitamente conhecidos ou conhecidos apenas em termos probabilísticos, falando-se então em PS determinísticos ou estocásticos. No que se refere ao conhecimento do horizonte de tempo para o qual se está resolvendo o PS, existem duas possibilidades: i) ou os pedidos são todos conhecidos *a priori* (*scheduling* estático ou preditivo); ii) ou aceita-se que o sistema de *scheduling* seja capaz de incorporar pedidos novos a um schedule que já está em implementação modificando-o (*scheduling* reativo ou dinâmico), o que significa dizer que são tomadas decisões em tempo real.

2.3.6- Estados Inicial e Final da Planta.

Em [Rodammer-88] enfatiza-se que o problema real de *scheduling* é sempre um problema de *re-scheduling*, onde os estados inicial e/ou final freqüentemente estão definidos. O primeiro porque a planta está funcionando e tem que ser capaz de incorporar novas tarefas (o problema anteriormente citado de *scheduling* reativo), o segundo porque freqüentemente se tem tarefas de execução cíclica com datas definidas. Esta situação pode ser generalizada para o caso onde existem tarefas ou operações com instantes de execução pré-definidos por outros níveis de decisão superiores ao *scheduling* ou por questões operacionais.

2.4- O Problema de *Flow-shop* na Indústria Química.

O problema de *scheduling* em *flow-shops* com seqüências de permutação é o problema multiestágio mais simples. Consiste em determinar a seqüência de um conjunto de N tarefas em um conjunto de M processadores, considerando as seguintes restrições: i) os processadores estão dispostos

em série; ii) as tarefas devem passar pelos processadores na mesma seqüência; e, iii) os processadores recebem as tarefas na mesma ordem. Mesmo com estas restrições a dimensão e a complexidade do problema são grandes e, para simplifica-lo, algumas hipóteses são feitas com relação às tarefas, aos processadores e aos tempos de processamento [Bellman-82]:

H1- Hipóteses sobre as tarefas:

H1.1- todas as tarefas estão prontas para processamento em t=0. Por definição o horizonte de tempo em que se realizará o schedule começa em t=0;

H1.2- nenhuma operação pode ser processada por mais de um processador;

H1.3- uma operação, uma vez iniciada, não pode ser interrompida por um certo intervalo de tempo para retomar posteriormente o seu processamento;

H1.4- não existe nenhuma ordem de prioridade entre as tarefas, todas têm o mesmo grau de importância;

H1.5- uma tarefa pode esperar pela liberação de um processador, ou seja, esta prevista a armazenagem intermediária;

H1.6- as tarefas podem ter um prazo de entrega especificado;

H1.7- cada tarefa deve passar por cada processador uma única vez.

H2- Hipóteses sobre os processadores:

H2.1- todos os processadores estão disponíveis em t=0 para todas as tarefas. Não ocorre quebra ou reparo de qualquer processador;

H2.2- existe um único processador de cada tipo (ou seja, cada estágio possui um único processador) e eles são independentes;

H2.3- cada processador processa no máximo uma única operação de cada vez.

H3- Hipóteses sobre os tempos de processamento:

H3.1- os tempos de processamento de cada operação são constantes, não importando a ordem de processamento;

H3.2- os tempos de processamento incluem os tempos de preparação dos processadores e os tempos de transferência das respectivas operações, uma vez que estes são considerados pequenos em relação àqueles.

Em geral, considera-se o *makespan* como critério de otimização, porque embora seja um critério difícil de ser otimizado, ele ainda é um dos mais simples. Soma-se a isto o fato de que o *makespan* possui outras características benéficas, além da minimização do tempo gasto para realizar todas as tarefas, como p.ex. a maximização da utilização dos processadores em uma linha de produção [Dudek-92].

Considerar válidas apenas as seqüências de permutação (SP) é uma restrição geralmente feita com o objetivo de simplificar o FS. Se o critério de otimização é o *makespan*, para $M \leq 3$, existe pelo

menos uma solução ótima para o FS na qual a seqüência das tarefas é uma SP [Bellman-82]. Para M≥4 não existe a garantia de a solução ótima seja uma SP, principalmente se os tempos de processamento das operações são muito diferentes [Tandon-91]. Na indústria química elas são utilizadas, não só porque o problema torna-se mais simples, mas porque [Rodrigues-92]:
- as seqüências de permutação reduzem a necessidade de armazenagem intermediária e o tempo total gasto na preparação dos processadores;
- quando os tempos de preparação dos processadores e os tempos de transferência das tarefas não são desprezíveis frente aos tempos de processamento, uma boa seqüência provavelmente será uma seqüência de permutação.

A hipótese H1.6 é considerada uma restrição flexível, que pode ser relaxada, e os critérios de otimização mais importantes neste caso são aqueles baseados no atraso das tarefas (*lateness*) [French-82].

A modificação das hipóteses acima e/ou a adição de novas hipóteses levam a diferentes tipos de problemas de *flow-shop*, tais como o FS proporcional [Adenso-Diaz-92], o FS ordenado [Dudek-92],o FS com processador dominante [Nouweland-92], o FS com seqüências de permutação equilibrado [Karabati-92], o FS repetitivo [Ramazani-93], etc, que não serão abordados neste trabalho.

2.4.1- Complexidade e Relevância do Problema.

Encontrar uma solução factível para o problema citado acima é relativamente fácil, dado que existe um número muito grande de soluções que preservam as seqüências tecnológicas das tarefas. No caso do *flow-shop* o número de seqüências factíveis é $(N!)^M$ e, considerando seqüências de permutação, (N!). A grande dificuldade é determinar uma seqüência ou um conjunto específico das seqüências de processamento factíveis que sejam ótimas. O único caminho que se conhece para fazer isto, para N e M grandes, é calcular o custo associado (de acordo com a função objetivo que se quer otimizar) de uma grande parte das soluções factíveis, usando enumeração completa ou parcial [French-82].

No entanto, o problema do *flow-shop* (para ser mais exato, a busca pela solução ótima em qualquer PS) pertence à classe dos problemas NP-completos (problemas não polinomiais [Garey-79]), e, como tal, o tempo necessário para resolvê-lo cresce exponencialmente com a dimensão do problema, mesmo quando se consideram seqüências de permutação. Por exemplo, o *flow-shop* com M>2 e qualquer critério de desempenho regular diferente do *makespan*, e com M>3 e o *makespan* como critério de desempenho são problemas NP-completos. Mesmo para um número pequeno de tarefas (2 ou 3) o *flow-shop*, o *job-shop* e o *open-shop* são NP-completos para critérios de desempenho regulares [Sotskov-91]. A explosão combinatorial resultante impede a enumeração explícita (completa) do espaço de soluções do problema e, por isso, as técnicas de enumeração implícita como o BAB [Belman-82] e o *simulated annealing* [Kirkpatrick-83, Das-90, Tandon-91] têm se mostrado bastante eficazes na obtenção de soluções ótimas e sub-ótimas.

Para o caso M=2, utilizando o *makespan* como critério de otimização, existe um algoritmo para obter a solução ótima para o *flow-shop*, que é o algoritmo de Johnson [French-82]. A principal característica deste algoritmo é que ele é polinomial, o que levou vários pesquisadores a buscar, sem sucesso, uma generalização deste algoritmo para M>2. Para o caso M=3 existem algumas heurísticas que reduzem o problema a um equivalente com M=2, possibilitando a sua solução pelo algoritmo de Johnson [Dannenbring-77].

Quando se considera recursos compartilhados além dos processadores o problema torna-se ainda mais complexo [Coffman-76] pois, como será visto na seção 3.3.4, não há uma relação de recorrência capaz de calcular os instantes de início das operações baseado apenas na seqüência das tarefas e nos tempos de processamento das operações. Isto faz com que as tarefas tenham que ser consideradas em termos de suas operações, para garantir a alocação factível destas. O problema deixa então de ser um problema apenas de seqüenciamento (dimensão N!) e passa a ser também um problema de *scheduling* (dimensão $N!^M$).

Alguns autores [MacKay-88, Dudek-91] consideram o problema, com as hipóteses feitas acima, uma simplificação muito grande dos problemas de programação de produção encontrados na prática, uma vez que elas desprezam diversas características do problema real. Entretanto o problema é muito importante e interessante porque ele captura a complexidade computacional do problema central que é o seqüenciamento de tarefas em processadores, a despeito de quaisquer outros fatores.

2.4.2- Políticas de Armazenagem Intermediária.

Um fator importante na construção de um *schedule* é o armazenamento dos produtos intermediários entre as unidades sucessivas da planta, ou seja, a política de armazenagem intermediária utilizada. A utilização de armazenagem intermediária é importante porque ela [Leinsten-90]:

- aumenta a produtividade: a disponibilidade de tanques de armazenagem permite que um processador seja liberado para outras tarefas quando o processador seguinte estiver ocupado;
- facilita *changeovers*;
- permite que equipamentos adjacentes trabalhem a taxas de produção diferentes (em processos semi-contínuos).

Existem cinco políticas básicas de processamento [Pokny-91, Knorf-85. Kurian-89, Rajagopalan-89]:

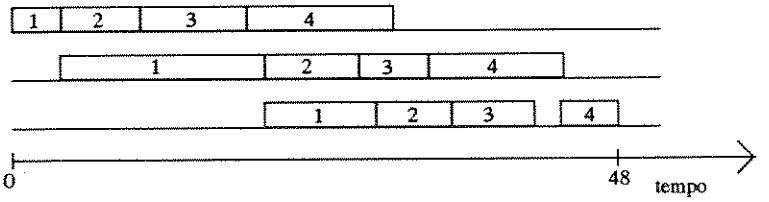
1- *Unlimited Intermediate Storage* (UIS): é a política mais flexível, nela um número ilimitado de tarefas podem ser armazenadas entre dois estágios consecutivos da planta;

2- *Finite Intermediate Storage* (FIS): a armazenagem intermediária é fixa e limitada;

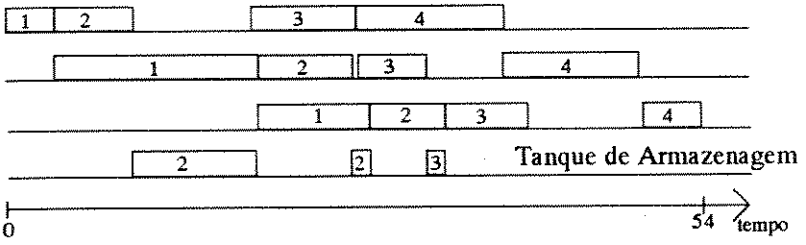
3- *No Intermediate Storage* (NIS): não existe armazenagem intermediária; ao se completar o processamento de uma tarefa em um estágio, esta permanece temporariamente na unidade que acabou de processá-la até que o próximo estágio seja liberado;
4- *Zero Wait* (ZW): não existe armazenagem intermediária; assim que uma tarefa é processada por um estágio, ela deve ser transferida imediatamente para o próximo estágio.
5- Armazenagem Dedicada: a armazenagem é específica de algum estágio de processamento e não é compartilhada por produtos intermediários.

A política de armazenagem pode variar entre os pares de estágios consecutivos, alguns tendo capacidade limitada e outros nenhuma capacidade. Assim, as quatro políticas acima podem se combinar dando origem à política ***Mixed Intermediate Storage*** (MIS). A diferença entre as políticas NIS e ZW é geralmente ignorada na literatura de Engenharia Industrial, sendo que os dois termos são utilizados para designar o tipo ZW de armazenagem intermediária.

A escolha de uma determinada política de armazenagem intermediária determina a quantidade de produtos intermediários que podem ser armazenadas temporariamente entre dois estágios consecutivos da planta, e isto tem um reflexo imediato sobre o tempo total de fluxo das tarefas (*makespan*). Relações de recorrência para o cálculo do tempo de conclusão das tarefas para o *flow-shop* FIS, NIS e ZW podem ser encontradas em [Baker-74] e para o caso MIS em [Rajagopalan-89]. A Figura 2.3 mostra o efeito da armazenagem intermediária sobre o tempo conclusão das tarefas em um *flow-shop* com seqüências de permutação.



(a) Carta de Gantt para o *flow-shop* UIS



(b) *Flow-shop* com um tanque de armazenagem compartilhado entre estágios

Figura 2.3 - Efeito da armazenagem intermediária sobre o tempo de conclusão das tarefas.

Embora a utilização de armazenagem intermediária forneça maior grau de liberdade de ação na construção do *schedule*, eliminando a necessidade de sincronismo entre processadores [Conway-88], este maior grau de liberdade geralmente não é utilizado na solução do problema de *scheduling*, servindo apenas como meio de melhorar ou factibilizar a seqüência final obtida quando existem períodos ociosos ou problemas com a demanda de recursos compartilhados.

2.4.3- *Flow-shop* com Alocações Externas.

O problema que será tratado neste trabalho é o problema de *flow-shop* com restrições sobre recursos compartilhados e sobre os instantes de tempo que as tarefas devem ser alocadas. Estes dois tipos de restrições diminuem o grau de liberdade de tomada de decisão na construção de um *schedule*, porque eles provocam interações entre operações de tarefas diferentes. A oferta limitada de um recurso compartilhado pode proibir a realização simultânea de duas operações, enquanto que os limites de tempo podem proibir a realização de uma operação após outra. Vê-se portanto que estas restrições interagem "antagonisticamente" [Smith-92], ou seja, a satisfação de uma pode causar a violação da outra, o que aumenta a complexidade da busca por uma solução ótima. Deseja-se uma seqüência de permutação que, além de satisfazer estas restrições, minimize o *makespan* da planta dentro da política de armazenagem intermediária ilimitada.

O que se pretende é tratar o problema de *flow-shop* de maneira mais realista, considerando que as tarefas podem não só ter prazos de entrega bem definidos como também tempo de pronto diferente de zero. Diversos tipos de restrições podem ser modeladas como restrições de tempo sobre as tarefas, isto é, existem várias situações em que as tarefas são condicionadas a acontecer dentro de um determinado intervalo de tempo, seja devido a restrições tecnológicas como falta de recurso para processamento de uma operação ou necessidade de manutenção de processadores, seja devido a restrições externas como atendimento a clientes preferenciais.

Neste trabalho define-se como **alocação externa** a atribuição de um intervalo de tempo dentro do qual uma tarefa deve ser programada (processada), ou seja a atribuição de um tempo de pronto e de um prazo de entrega para a tarefa.

Geralmente, na literatura de *scheduling*, as alocações externas e os limites sobre os recursos compartilhados são considerados restrições flexíveis, que podem ser relaxadas, o que normalmente é feito com um único objetivo: simplificar o problema. Neste trabalho considera-se que estas restrições não podem ser relaxadas, sob pena de chegar a uma solução final infactível ou pobre do ponto de vista econômico.

A armazenagem intermediária, embora seja um recurso compartilhado, não é considerada como tal porque os recursos compartilhados como energia elétrica e vapor possuem perfis de demanda

especificados pela receita tecnológica de execução das tarefas (são variáveis estáticas), enquanto que na armazenagem intermediária a demanda é gerada por necessidades surgidas no decorrer da programação da produção (são variáveis dinâmicas). Em outras palavras, a demanda pela armazenagem intermediária é gerada pela não disponibilidade de processadores devido à assincronia natural na execução das operações das diferentes tarefas, a necessidade de armazenagem no instante t dependendo então das decisões de alocação tomadas no intervalo [0,t) [Steffen-86]. Como o problema é considerado do ponto de vista determinístico e estático, onde todos os dados necessários para resolvê-lo são conhecidos a priori e com exatidão, as restrições sobre armazenagem intermediária não serão tratadas, ficando como proposta de trabalhos futuros.

No algoritmo que será proposto no capítulo 4, as restrições sobre o instante de alocação das tarefas e as restrições sobre os recursos compartilhados serão tomadas como restrições a serem satisfeitas pelo processo. Além do mais, a maximização da utilização dos recursos servirá de base para a minimização do *makespan*.

2.5- Conclusão.

Neste capítulo apresentou-se as características gerais dos problemas de *scheduling*, que são problemas de otimização complexos em que o custo computacional para se chegar a uma solução ótima cresce exponencialmente com a dimensão do problema. Dentre estes problemas destaca-se o *flow-shop* com seqüências de permutação, cuja principal área de aplicação é a indústria de processos químicos.

Para diminuir a complexidade do *flow-shop* geralmente fazem-se hipóteses simplificadoras com relação às tarefas, aos processadores e aos tempo de processamento das operações. Estas hipóteses fazem do problema resultante uma simplificação muito grande dos problemas de programação de produção encontrados na prática.

Propõe-se então resolver o problema de *flow-shop* com seqüências de permutação incorporando dois tipos de restrições que são bastante relevantes na indústria de processos químicos, quais sejam: restrições sobre os recursos compartilhados e restrições sobre os instantes de alocação das tarefas (alocações externas). Vários tipos de restrições encontradas na prática podem ser modeladas como uma destas duas restrições. O objetivo é dar um caráter mais realista ao problema.

Capítulo 3

Metodologias e Técnicas Utilizadas no Problema de *Scheduling* Aplicáveis ao *Flow-shop* com Recursos Compartilhados e Alocações Externas.

3.1- Introdução.

Neste capítulo discute-se sucintamente as técnicas utilizadas na resolução do problema de *scheduling*. Concentra-se a atenção nas técnicas aplicáveis ao problema específico de que trata este trabalho, ou seja, o problema do *flow-shop* com seqüências de permutação, restrições sobre os recursos compartilhados e alocações externas de tarefas.

Na seção 3.2 apresenta-se uma das classificações existentes na literatura para as técnicas aplicadas ao problema de *scheduling* geral.

As técnicas orientadas à satisfação de restrições são descritas na seção 3.3, enfatizando as metodologias existentes para propagação de restrições e o tratamento dos recursos compartilhados.

3.2- Classificação das Técnicas de *Scheduling*.

Na literatura encontram-se diversas classificações das técnicas utilizadas para solucionar os PS (p.ex. [Erschler-86, Rickel-88, Rodammer-88]). Dentre elas resume-se a seguir a classificação feita por Rodammer [Rodammer-88], que é bastante abrangente. Nesta referência citam-se sete abordagens: prática industrial; seqüenciamento de máquinas e teoria de *scheduling*; *scheduling* de projetos com restrições sobre os recursos; teoria de controle; simulação de sistemas a eventos discretos; otimização estocástica; e, inteligência artificial.

Segue-se uma breve descrição de cada uma destas abordagens:

Prática Industrial

Em geral, na prática industrial, o *scheduling* é feito manualmente construindo a carta de Gantt. O conhecimento da planta e a experiência são as ferramentas principais. Como auxílio neste processo utilizam-se bases de dados sobre matérias primas, pedidos e produtos intermediários.

Utilizam-se algumas ferramentas que automatizam algumas fases da processo de geração do *schedule*. Dentre elas cabe citar:

MRP:

MRP (*Manufacturing Resource Planning*): Para um horizonte dado de planejamento determina: i) as quantidades de matérias primas e produtos intermediários necessários para uma certa quantidade de produto final e ii) os tempos em que estes elementos tem que estar disponíveis em função do prazo de entrega do produto final. Determina necessidades sem considerar a capacidade instalada da planta, podendo o resultado estar sobre ou sub-dimensionado;

JIT:

JIT (*Just In Time*): Propõe a redução de estoques dos itens de entrada e dos produtos intermediários. Não propõe uma técnica para geração do *scheduling*;

OPT:

OPT (*Optimized Production Timetables*): Trata-se de um pacote fechado que ao que tudo indica seleciona um *schedule* candidato e o simula para determinar os engargalamentos, ou seja, os recursos críticos. A partir daí a seqüência final é gerada partindo dos engargalamentos utilizando processos heurísticos.

Seqüenciamento de Máquinas e Teoria de *Scheduling*

O problema de *scheduling* tem sido estudado, sob certas hipóteses simplificadoras, (citadas no item 2.4) seguindo duas linhas: a programação matemática e a busca em árvore. As técnicas utilizadas são descritas em [Baker-74, French-82, Bellman-82, Coffman-76], mas o problema de otimização resultante, para qualquer problema real, tem dimensão enorme impossibilitando a utilização prática destas técnicas.

Como conseqüência tem-se desenvolvido diversos processos heurísticos de decisão para a escolha da próxima tarefa a ser introduzida na planta (*flow-shop*) ou da próxima operação a ser processada numa máquina (*job-shop*), baseadas no estado das tarefas, operações e planta. Em [Panwalkar-77] citam-se 113 regras heurísticas para a programação de tarefas. As mais utilizadas no caso de *flow-shops* são descritas em [Dannenbring-77]. Estes processos heurísticos de decisão considerando apenas o próximo evento (*single stage heuristics*) são também conhecidos como regras de despacho (*dispatch rules*) e *loading rules* [Grant-86, Rickel-88].

***Scheduling* de Projetos com Restrições sobre Recursos**

Rodammer descreve esta abordagem como a tentativa de utilizar no seqüenciamento de tarefas as ferramentas desenvolvidas na área de *scheduling* de projetos com restrições sobre os recursos. Essas ferramentas utilizam técnicas de programação matemática (p.ex. o trabalho recente descrito em [Demenlemeester-92] que utiliza a técnica *Branch-and-Bound*) e de inteligência artificial (p.ex. o método de busca heurística descrita em [Erschler-89]). O autor sugere que os *softwares* comerciais disponíveis para o

scheduling de projetos e algumas heurísticas desenvolvidas nesta área podem ser úteis no sequenciamento de máquinas.

Teoria de Controle

O PS pode ser visto como um problema de controle onde: i) as entradas controladas são as decisões de seqüenciamento e alocação, ii) as perturbações são falhas em processadores ou variações nos recursos disponíveis em geral, iii) o estado da planta é definido pelos estoques, ocupação dos processadores e disponibilidade de recursos em geral e iv) o objetivo é determinar as entradas controladas que levam a que o estado tenha a evolução desejada no tempo satisfazendo as restrições. A formulação do problema de controle parece bem adequada ao problema de *scheduling*, o problema é que as técnicas existentes para a síntese do controle aplicam-se a sistemas contínuos ou discretos no tempo, mas ainda estão em fase inicial quando se trata de sistemas de eventos discretos (DEFD-*Discrete Event Dynamic Systems*).

Simulação de Sistemas a Eventos Discretos

Uma simulação da planta é freqüentemente utilizada para avaliar o desempenho de decisões de *scheduling* tomadas através de processos heurísticos, ou seja, para teste. Dado que a maioria dos problemas de *scheduling* reais são complexos e a obtenção da estratégia ótima é difícil, pode-se imaginar a utilização de simulações de uma forma interativa para a geração do *schedule*. Nesta situação o usuário (re)define as decisões de *scheduling* e testa o seu desempenho sobre a simulação num processo interativo.

A vantagem inerente a uma técnica de simulação é a possibilidade de introduzir todos os aspectos relevantes do problema real. Uma das principais desvantagens é a dificuldade de extrapolar resultados.

Otimização Estocástica

Pode ser vista como ferramentas de programação matemática como teoria de filas, *reliability theory*, *lot sizing techniques* e *inventory theory*. Uma restrição comum é que trabalham com modelos altamente simplificados.

Inteligência Artificial

Com o objetivo de dar maior flexibilidade aos modelos de *scheduling*, surgiram nos últimos anos novas técnicas baseadas em inteligência artificial (IA). A maioria destas técnicas constroem uma árvore de busca para ajudar a encontrar a solução do problema. Dependendo da aplicação, a busca pela árvore pode ser feita em profundidade, em largura, ou em alguma combinação desta duas formas [Nilsson-

80, Pearl-84]. Exemplos de alguns métodos de busca utilizados por estes sistemas são o *beam-search* [Fox-83, Ungar-90], o *best-first-search* [Smith-90, Ow-89] e o *depth-first-search* [Bensana-88, Bel-89]. O tamanho da árvore de busca pode ser reduzido utilizando algumas das seguintes abordagens [Kusiak-87, Dattero-89]: hierárquica, oportunista e guiada-por-restrições (*constraint-directed*).

Em uma abordagem hierárquica [Sacerdoti-74, Steffen-86, Wilkins-87, Ungar-90, Burke-91] o problema original é decomposto em vários subproblemas menores e mais simples e com nível de detalhamento crescente. Cada subproblema é uma abstração, que ignora certos detalhes do problema original, e o conjunto de subproblemas forma uma hierarquia de abstrações (*hierarchy of abstractions*) [Sacerdoti-74]. O problema é então resolvido em um nível de abstração mais alto, onde detalhes pouco importantes não são considerados, e depois a solução encontrada é compatibilizada com os níveis de abstração mais baixos, com maior nível de detalhamento. Isto pode resultar em uma considerável redução do espaço de busca.

Na abordagem oportunista [Smith-85, Sadeh-89, Smith-90] as tomadas de decisão são direcionadas para aquelas ações que se mostram as mais promissoras em termos do estado atual da planta. Esta abordagem cria "regiões de certeza" no espaço de busca, e somente as soluções contidas nestas regiões são pesquisadas. É uma abordagem que tem se mostrado bastante eficiente na solução de problemas de *scheduling* via a utilização de heurísticas [Smith-85].

A abordagem guiada-por-restrições (*constraint-directed*) [Atabakhsh-91] é uma técnica de busca heurística na qual o conhecimento que se tem da planta é representado por restrições que limitam e guiam a busca por uma solução factível. Esta abordagem será discutida na seção 3.3.

3.3- Técnicas Orientadas à Satisfação de Restrições Aplicáveis ao Problema do *Flow-shop* com Recursos Compartilhados e Alocações Externas.

O problema de *scheduling*, como já foi visto, está sempre sujeito a um conjunto de objetivos (minimização do *makespan*, maximização da utilização dos recursos, etc) e restrições (seqüência tecnológica das tarefas, preferências do usuário, alocação externa de tarefas, escassez de recursos compartilhados, etc). A literatura de *scheduling* mostra que resolver o problema considerando ao mesmo tempo os objetivos da produção e as restrições do processo é uma tarefa árdua, e por isso as técnicas disponíveis geralmente contemplam apenas um destes dois aspectos.

As técnicas clássicas de *scheduling* como a programação dinâmica, a programação inteira, o *branch-and-bound*, etc, denominadas técnicas de Pesquisa Operacional (PO), são técnicas voltadas para a otimização de um critério de desempenho e que geralmente só consideram como restrições rígidas as seqüências tecnológicas das tarefas. Geralmente as alocações externas de tarefas são consideradas

restrições flexíveis, que podem ser relaxadas mediante o pagamento de uma multa, e a escassez de recursos só é considerada após a solução do problema, numa fase de factibilização.

Na literatura de Engenharia Química existem alguns trabalhos que tentam modelar o PS como um MILP (*mixed integer linear program*). Isto é feito em [Kondili-93], onde o PS é formulado como um MILP contemplando, entre outras, as seguintes restrições: armazenagem intermediária do tipo MIS; limitação dos recursos compartilhados; alocações externas; estrutura de processamento genérica; etc. O problema com este tipo de formulação é que o número de variáveis binárias necessárias é muito elevado, mesmo para representar um problema de pequena dimensão, o que compromete a utilização prática deste tipo de abordagem [Shah-93].

A abordagem mais promissora ao problema em questão parece ser a busca de soluções orientada para a satisfação de restrições (*Constrained Based Search*) [Fox-83], que pode ser vista dentro do contexto maior da área de Inteligência Artificial denominada *Constraint Based Reasoning*, *Constraint Satisfaction Problems*, etc. A idéia básica é resolver o problema de otimização com restrições através de uma técnica de busca onde o processo de enumeração inerente seja limitado pelas restrições presentes.

O primeiro sistema de *scheduling* desenvolvido com o objetivo de satisfazer restrições (ISIS) foi proposto por Fox [Fox-83]. A partir deste trabalho foram desenvolvidos vários outros sistemas que combinam técnicas de alocações heurísticas com a propagação das restrições resultantes destas alocações, p.ex. OPIS [Smith-90], CSS [Ow-88], CORTES [Sadeh-89], ISPS [Keng-88], OPAL [Bensana-88, Bel-89].

3.3.1- Processo de Busca com *Backtracking*.

O processo de busca numa técnica orientada à satisfação das restrições será descrito a seguir no contexto de um problema de satisfação de restrições (*Constraint Satisfaction Problem* - CSP) [Atabakhsh-91].

Um CSP consiste de um conjunto de n variáveis, X_1, X_2, ..., X_n, a cada uma delas podendo ser atribuído um valor v_i, dentro de seu domínio pré-definido, D_1, D_2, ..., D_n, respectivamente. A atribuição de um valor $v_i \in D_i$ a uma variável X_i é chamada de uma instanciação de X_i. As relações entre as variáveis são especificadas por um conjunto de restrições consistentes, que irão limitar as suas possíveis instanciações. Para resolver um CSP é preciso instanciar todas as variáveis satisfazendo todas as restrições. Isto implica na capacidade de enumerar todas as instanciações possíveis das variáveis, o que faz do CSP um problema NP-completo [Garey-79].

Um paradigma geral para solucionar um CSP é o algoritmo de busca em árvore denominado *Backtracking Search* (BS) [Pearl-84]. A vantagem de usar um algoritmo de busca é que ele um algoritmo

construtivo, ou seja, a solução do problema é gerada durante o processo de enumeração das instanciações das variáveis. O uso conjunto do CSP e de um algoritmo de busca heurística (BH) na solução de um problema dá origem à técnica denominado *constraint heuristic search* [Fox-83]. Esta técnica é descrita nos parágrafos abaixo.

A BS consiste em atribuir valores para um conjunto de variáveis (no caso do *scheduling*, instantes de início para cada operação) de modo que todas as restrições associadas a estas variáveis sejam satisfeitas. O algoritmo ordena as variáveis e, começando pela primeira, atribui um valor (tomado de um conjunto de valores factíveis) para cada uma na ordem pré-determinada, contando que este valor seja consistente com os valores atribuídos às variáveis anteriores.

Quando a busca se depara com uma variável tal que nenhum de seus possíveis valores é consistente com as atribuições feitas anteriormente (situação conhecida como *dead-end*), ocorre o *backtracking*, ou seja, o valor atribuído a uma ou mais variáveis precedentes é trocado e a busca continua de forma sistemática até que todas as variáveis sejam instanciadas, ou até se chegar à conclusão que não existe nenhuma solução capaz de satisfazer todas as restrições do problema.

Neste tipo de algoritmo, o tamanho do espaço de busca depende em grande parte de dois fatores:

1- a forma como as restrições são representadas no problema. O algoritmo se beneficiará mais das restrições quanto mais explicitamente elas forem representadas, mesmo quando elas resultam da combinação de outras restrições. O objetivo será poder levar em consideração o maior número de restrições no processo de busca de forma a diminuir o tamanho da árvore. Obviamente este objetivo terá que ser balanceado com o esforço computacional necessário para mapear todas as restrições em condições a serem testadas em cada nó da árvore;

2- a técnica utilizada para a expansão de cada nó da árvore. Num extremo pode-se utilizar uma técnica que garanta a obtenção da solução ótima, como p.ex. o *Branch-and-Bound*, o que vai levar a uma árvore normalmente grande. No outro extremo podem-se utilizar heurísticas que levem à determinação de um único nó-filho para cada nó-pai (equivalente a uma regra de despacho). Neste caso a árvore teria um só caminho que provavelmente não seria a solução ótima. Entre estes dois extremos tem-se as técnicas que de uma forma mais ou menos heurística reduzem o espaço de busca, a custa de, eventualmente, perder a otimalidade da solução.

A utilização de alguma técnica heurística para expandir um nó e decidir o caminho que será seguido implica na ordenação dos nós-filhos para, na ocorrência de *backtracking*, escolher um caminho alternativo. Este ordenamento das variáveis (os nós filhos possíveis) terá obviamente influência sobre o tamanho do espaço de busca. Além da expansão do nó, o processo de busca deve definir um valor para

a variável (no caso de PS, o instante de início de processamento da operação) satisfazendo as restrições do problema.

Estes dois fatores podem ser utilizados *a priori* ou dinamicamente durante o processo de busca para melhorar o desempenho da BS. As técnicas projetadas para serem utilizadas *a priori* fazem uso de algoritmos para, a partir das restrições originais, deduzir novas restrições que serão adicionadas ao problema. As técnicas utilizadas dinamicamente para reduzir o espaço de busca podem ser convenientemente classificadas como *lookahead* e *lookback* [Dechter-90].

O *lookahead* é invocado sempre que o algoritmo esta se preparando para atribuir um valor para a próxima variável. Algumas funções que esta técnica pode realizar são:

1- decidir qual será a próxima variável a ser instanciada (quando a ordem não é pré-determinada). Uma heurística freqüentemente utilizada é instanciar primeiro as variáveis mais fortemente restritas (*tigtly constraint*). Agindo assim o sistema evita perder muito tempo construindo soluções parciais que poderão não vir a ser completadas porque algumas variáveis difíceis não foram instanciadas ainda;

2- decidir qual valor será atribuído à variável escolhida. Uma heurística para ordenamento dos valores é a *least-constraining*, que atribui o valor que maximiza o número de opções disponíveis para as instanciações das outras variáveis, ou seja, ela busca restringir o mínimo possível o domínio das outras variáveis para que o solução parcial possa ser terminada sem *backtracking*;

3- propagar restrições. Consiste em verificar as possíveis infactibilidades, num período à frente, resultantes da instanciação de uma variável.

O *lookahead*, ao utilizar o conjunto de variáveis e seus domínios em um mecanismo de propagação de restrições, permite ao sistema detectar conflitos entre as restrições antes que eles ocorram, evitando até certo ponto a explosão combinatorial inerente ao CSP. Se o *lookahead* for realizado com perfeição o sistema poderá encontrar uma solução para o problema sem precisar realizar o *backtracking*. No entanto, como o problema de *scheduling* é NP-completo, acredita-se que é impossível obter tal eficiência [Sycara(b)-91].

O *lookback* é invocado quando o algoritmo encontra um *dead-end* e se prepara para fazer o *backtrack*. Ele é responsável por [Dechter-90]:

1- decidir até que ponto o algoritmo deve retroceder. Analisando as razões do *dead-end* pode ser possível voltar diretamente à fonte do conflito, em lugar de voltar a uma variável prévia no ordenamento. Esta idéia é freqüentemente denominada *dependency-directed backtracking* ou *backjumping*. Quando o algoritmo retrocede à variável imediatamente anterior o que se tem é um *backtracking* cronológico.

2- Armazenar as causas do *dead-end* na forma de novas restrições para evitar que os mesmos conflitos apareçam novamente na busca. Esta idéia é denominada *constrained recording* ou *nogood constraint.*

3.3.2- Propagação de Restrições. [Keng-88]

A propagação de restrições é uma atividade dedutiva que permite decompor um problema sem negligenciar as interações entre os subproblemas resultantes, determinar quais subproblemas são mais urgentes (mais restritos) e focalizar a atenção nestes [Le Pape-92]. A propagação de restrições formula novas restrições analisando as já existentes e detecta inconsistências entre elas.

Em problemas de *scheduling*, a propagação de restrições resultante de uma alocação pode levar à detecção de dois tipos de conflitos [Smith-90, Sycara(a-b)-92]:
- conflitos de tempo: situações em que os limites de tempo, dada as durações das operações de uma tarefa, implicam em violação das restrições de precedência destas operações;
- conflitos de capacidade: situações em que a quantidade de recurso necessária para processar um conjunto de operações excede a capacidade disponível do recurso em alguma unidade de tempo. A propagação de restrições só é capaz de detectar conflitos entre um conjunto de operações alocadas e uma não alocada. A detecção de conflitos entre várias operações não alocadas provavelmente requer tempo exponencial [Sycara(a)-91].

Para que a propagação de restrições seja possível é preciso definir o intervalo de tempo factível para a alocação de cada operação, denominado janela de demanda (*demand window*-DW) da operação. Cada DW é limitado pelo instante de início mais cedo (*earliest beginning time*-EBT) e pelo instante de fim mais tarde (*latest finish time*-LFT) de uma operação. Estes instantes são calculados, a partir do *ready time* (RT) e do *due date* (DD) das tarefas, pelas relações de seqüenciamento 3.1 e 3.2.

$$EBT_{ij} = EBT_{ip} + \sum_{l=p}^{j-1} TP_{il}, \text{ com } p<j \text{ (}\textit{forward propagation}\text{)} \qquad (3.1)$$

$$LFT_{ij} = LFT_{ip} - \sum_{l=p}^{j+1} TP_{il}, \text{ com } p>j \text{ (}\textit{backward propagation}\text{)} \qquad (3.2)$$

onde: $EBT_{i0} = RT_i$, $LFT_{i0} = DD_i$, e p é o processador onde tem início a propogação de restrições.

Apresentam-se a seguir duas técnicas descritas na literatura para propagação de restrições.

3.3.2.1- Propagação Intra-Ordem e Inter-Ordem. [Keng-88]

Em um PS, como p.ex. o *job-shop* ou o *flow-shop*, a alocação de uma tarefa (ou operação) dá origem a dois tipos de propagação de restrições: a propagação intra-ordem e a propagação inter-ordem, onde uma ordem representa o conjunto de operações de uma tarefa. A propagação de restrições pode modificar os instantes iniciais, finais, ou até mesmo recortar um DW. A infactibilidade de uma solução (inconsistência das restrições temporais) é detectada quando uma operação fica com o DW menor que o tempo necessário ao seu processamento.

A propagação intra-ordem surge em conseqüência das relações temporais de precedência entre as diversas operações de uma mesma tarefa, e consiste na modificação dos instantes iniciais (*forward propagation*) das operações sucessoras, e dos instantes finais (*backward propagation*) das operações predecessoras à última operação alocada, de acordo com as relações 3.1 e 3.2, respectivamente.

Há dois tipos de propagação inter-ordem:
i) propagação da indisponibilidade dos processadores: consiste em proibir o intervalo de tempo em que um processador é alocado a uma operação nas janelas de demanda das demais operações que competem pelo uso deste processador (propagação inter-ordem intra-processador),

ii) propagação da indisponibilidade de recursos compartilhados: consisite em proibir os intervalos de tempo das operações em que a demanda por um recurso compartilhado é maior que a disponibilidade deste nestes intervalos (propagação inter-ordem inter-processador). Após a propagação inter-ordem é necessário realizar a propagação intra-ordem dos intervalos de tempo que acabam de ser proibidos.

A propagação de restrições pode recortar as janelas de demanda de algumas operações, e aquelas que não tiverem duração suficiente para processar as suas respectivas operações devem ser canceladas. A modificação ou o cancelamento da janela de tempo de alguma operação da origem a um novo ciclo de propagação de restrições.

As propagações intra-ordem e inter-ordem levam em consideração a disponibilidade de recursos e as rotas das tarefas, ambas conhecidas a *priori*, e são válidas para o *job-shop* e *flow-shop*. Quando se considera um *flow-shop* com seqüências de permutação, estas restrições também precisam ser propagadas. Isto pode ser complicado porque geralmente não se sabe *a priori* em que seqüência as tarefas serão processadas (esse é justamente o problema que se quer resolver). A dificuldade surge exatamente quando a alocação de uma operação em um processador recorta as janelas de outras operações que utilizam o mesmo processador, pois estas últimas operações poderão, a príncipio, serem feitas tanto antes quanto depois da operação que acaba de ser alocada. Quando não ocorre recortes nas janelas de outras operações é porque estas têm, obrigatoriamente, que serem feitas antes ou depois da operação que acaba de ser alocada e, neste caso, a propagação de restrições para preservar as seqüências de permutação é possível. Uma maneira de evitar os problemas com a propagação das

seqüências de permutação é utilizar o procedimento proposto em [Rodrigues-92]: alocar todas as operações de uma mesma tarefa simultaneamente nos seus instantes de início mais cedo, evitando assim o recorte de janelas. Este procedimento é utilizado na proposta do capítulo 4.

Exemplo: Propagação de restrições.

As Figuras 3.1 a 3.4 ilustram o mecanismo de propagação de restrições para um *flow-shop* com 3 tarefas, 3 processadores (P1,P2,P3) e um recurso compartilhado cuja disponibilidade em cada unidade de tempo é igual a 10. Dado um horizonte total de 40 unidades de tempo, RT_i=0 e DD_i=40 para i={1,2,3}, a Figura 3.1 mostra as janelas de tempo das operações calculadas de acordo com as relações (3.1) e (3.2). A Figura 3.2 mostra a alocação da operação o32 no intervalo [10,16] e a propagação desta restrição pelo sistema. A Figura 3.3 mostra o cancelamento de uma janela de tempo resultante da propagação de restrições. A Figura 3.4 mostra a propagação de restrições para manter as seqüências de permutação e, finalmente, a Figura 3.5 mostra a configuração final do sistema após o término do processo.

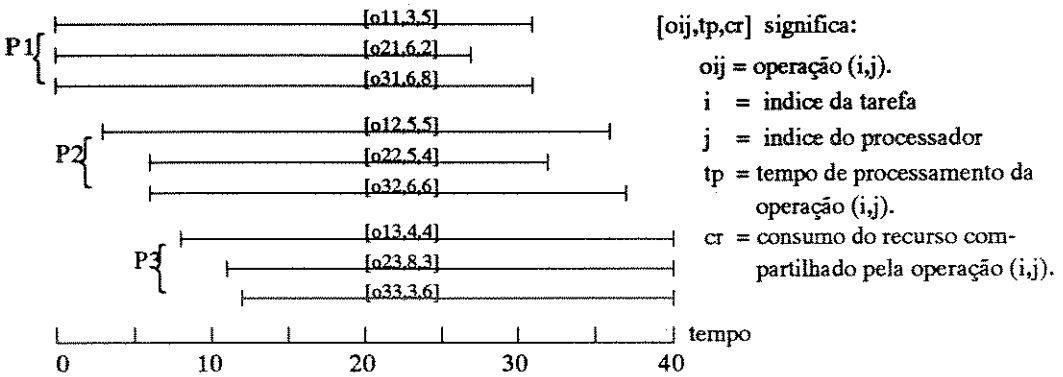


Figura 3.1- Janelas de tempo das operações

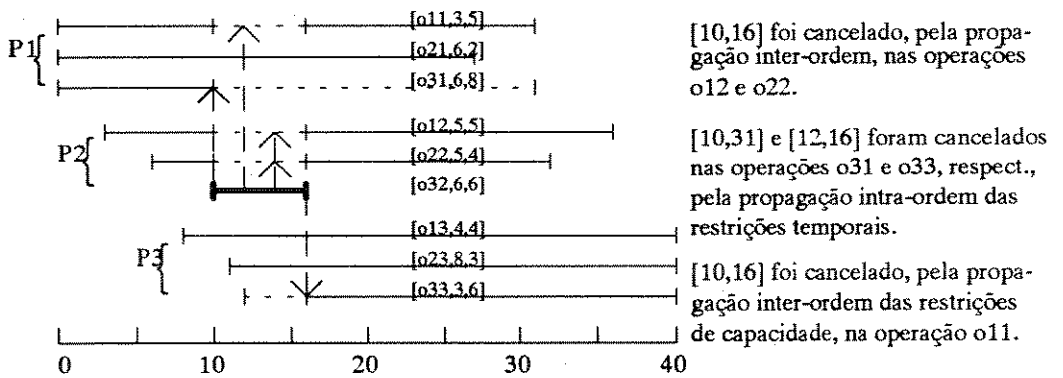
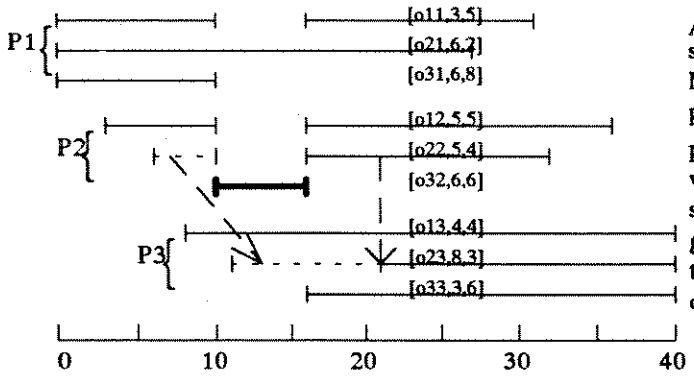
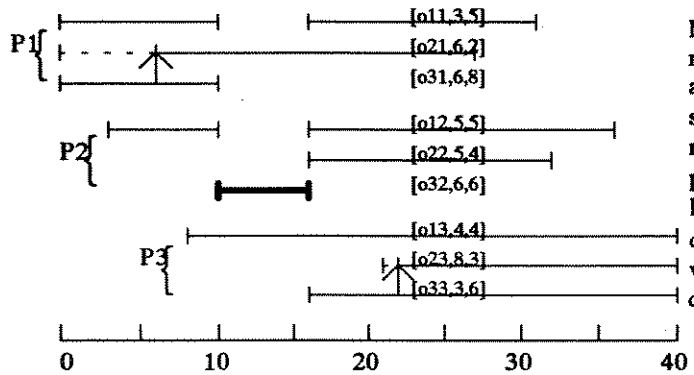


Figura 3.2- Alocação de o32 no intervalo [10,16] e propagação de restrições.



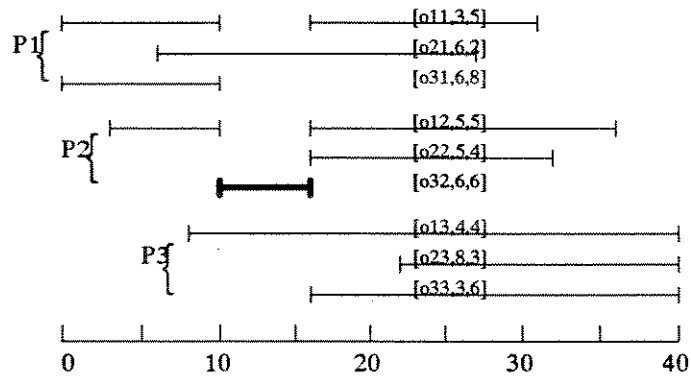
As op. o11, o12 e o22 tiveram as suas janelas de tempo recortadas. Note que o intervalo [6,10] da operação o22 não é suficiente para processá-la. Portanto este intervalo deve ser cancelado, devendo ser realizada uma nova propagação intra-ordem das restrições temporais resultantes deste cancelamento.

Figura 3.3- Cancelamento do intervalo [6,10] de o22 e propagação de restrições.



No processador 2, a tarefa o22 tem necessariamente que ser alocada após a tarefa o32. Para manter as seqüências de permutação, o mesmo deve acontecer nos demais processadores. Portanto o intervalo [0,6] da operação o21 foi cancelado no processador 1 e o intervalo [21,22] da operação o23 foi cancelado no processador 3.

Figura 3.4- Cancelamento de janela para manter as seqüências de permutação.



Como não há mais nenhuma restrição a ser propagada, o processo termina. Para todas as operações ainda não alocadas existe pelo menos uma janela de tempo capaz de satisfazer a sua respectiva operação, a configuração resultante é, portanto, factível.

Figura 3.5- Configuração do sistema após o término da propagação de restrições.

■

3.3.2.2- Determinação das Condições de Factibilidade de uma Solução.

O mecanismo de propagação de restrições visto na seção anterior permite, a partir de restrições já existentes, deduzir novas restrições para o problema trabalhando sobre as janelas de demanda das operações. O objetivo principal é detectar, antes da alocação definitiva de uma operação, se as suas restrições são inconsistentes com as restrições das demais operações, permitindo que o algoritmo evite os caminhos que levam a um *dead-end*. Em adição à propagação de restrições, as janelas de demanda

das operações podem ser utilizadas para deduzir relações de precedência entre elas, como foi demonstrado por Erschler em [Erschler-76]. Na abordagem proposta no capítulo 4 esta técnica será utilizada para evidenciar possíveis violações de relações de precedência (definidas sempre num FSP) resultantes de alocações.

O mecanismo proposto por Erschler, denominado <u>análise de restrições</u>, analisa as restrições de tempo (na forma de janela de demanda das operações) e de disponibilidade de recursos do PS e determina as condições, denominadas <u>condições de factibilidade</u> ou <u>características essenciais</u>, que qualquer solução factível deve obedecer.

Considere uma operação (i,j) e um conjunto de operações $O_{kj}=\{(k_1,j),...,(k_q,j)\}$ que utilizam um processador j. É possível associar uma <u>operação fictícia</u> (k,j) ao conjunto O_{kj}, tal que:

$$TP_{kj} = \sum_{v=1}^{q} TP_{k_v j},$$

$$EBT_{kj} = \underset{v}{MIN}(EBT_{k_v j}),$$

$$LFT_{kj} = \underset{v}{MAX}(LFT_{k_v j}).$$

Como não é possível processar duas operações simultaneamente em um mesmo processador, então as seguintes relações de seqüenciamento são válidas:

A.1) Se $(LFT_{ij}-EBT_{kj} < TP_{kj}+TP_{ij})$ então (i,j) precede $(k_1,j) \vee ... \vee (k_q,j)$;

A.2) Se $(LFT_{kj}-EBT_{ij} < TP_{kj}+TP_{ij})$ então $(k_1,j) \vee ... \vee (k_q,j)$ precede (i,j);
onde $\vee$ é o operador lógico OU.

Uma vez determinado que existe uma relação de seqüenciamento entre um conjunto de operações, as suas janelas de demanda devem ser atualizadas pelas seguintes relações:

B.1) Se (i,j) precede $(k_1,j) \vee ... \vee (k_q,j)$,
Então $LFT_{ij} = MIN\{LFT_{ij}, MAX_{1 \le v \le q}\{LFT_{(kv,j)}-TP_{(kv,j)}\}\}$.

B.2) Se $(k_1,j) \vee ... \vee (k_q,j)$ precede (i,j),
Então $EBT_{ij} = MAX\{EBT_{ij}, MIN_{1 \le v \le q}\{EBT_{(kv,j)}+TP_{(kv,j)}\}\}$.

As relações A e B são denominadas <u>características essenciais</u> ou <u>condições de factibilidade</u>, pois não é possível achar um *schedule* que satisfaça as restrições impostas pelo processo e não satisfaça estas relações.

Exemplo: Análise de restrições.

A Figura 3.6 mostra as janelas de tempo de duas tarefas, A e B, em um dado processador. A tarefa A necessita de 3 ut para ser processada, enquanto B necessita de 2 ut. Como ($LFT_{(B)}$-$EBT_{(A)}$=4) <($TP_{(A)}$+$TP_{(B)}$=5), a tarefa A não pode ser programada antes de B. Conseqüentemente as janelas de A e B devem ser modificas de forma a explicitar esta restrição.

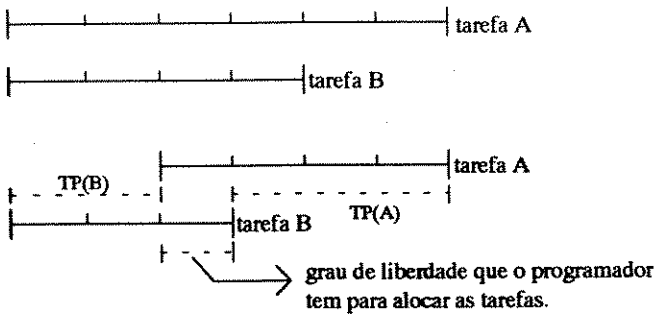


Figura 3.6- Exemplo da análise de restrições.

■

Esta é uma abordagem que não gera uma solução única para o problema, mas restringe o número de soluções factíveis através das condições de factibilidade, deixando um certo grau de liberdade para o *scheduler* tomar as decisões de alocação. É considerada uma técnica de Pesquisa Operacional, porque não faz uso de heurísticas ou conhecimento específico do problema que se quer resolver. Tem sido utilizada como parte de um sistema de *scheduling* baseado em conhecimento, como p.ex. em [Erschler-86, Erschler-89, Bensana-88, Bel-89].

Em [Erschler-86] é proposto a generalização das relações A e B para o caso em que as operações estão em conflito pela utilização de um recurso compartilhado, em lugar de um processador. As relações resultantes, no entanto, não são diretamente aplicáveis pois podem existir várias combinações de operações que, sobrepostas, satisfazem as restrições sobre os recursos (no caso do processador não existe nenhum conjunto de operações que podem ser sobrepostas, e portanto a aplicação das regras é fácil).

3.3.3- Exemplos de Sistemas de *Scheduling* Utilizando a Propagação de Restrições.

Nesta seção são apresentadas algumas técnicas existentes na literatura que utilizam as metodologias de propagação das restrições através das janelas de demanda das operações. Todas elas foram desenvolvidas para o *scheduling* de *job-shops*.

3.3.3.1- Planejamento Sensível a Interações para *Scheduling* de *Job-shop* (ISPS).

Neste sistema, proposto por Keng [Keng-88], o problema original é decomposto em um conjunto de subproblemas, com base no horizonte de tempo disponível (DW) para processar cada operação (*operand-based perspective*). Esta proposta utiliza as janelas de demanda não só para restringir o espaço de busca e manter a consistência das restrições do problema, através da propagação de restrições, mas também para determinar o ordenamento das variáveis e seus possíveis valores, através de heurísticas. O *job-shop scheduling* é tratado como um problema de planejamento composto de 4 módulos:

1- Processo de seleção de rota

Cada tarefa possui um GPO (gráfico de precedência operacional) que contém as suas possíveis rotas e alternativas de processadores em que elas podem ser processadas. O custo de uma rota é igual ao acúmulo dos seus tempos de processamento. As rotas de cada tarefa são então ordenadas de acordo com seu custo e, nos casos de empate, a que tem menor número de operações é selecionada primeiro.

2- Análise de capacidade

Tenta predizer a distribuição de carga em cada processador e faz o balanço de carga de cada um deles. Se a carga de uma processador em algum intervalo de tempo é maior que um determinado limiar (p.ex. 1), então uma rota alternativa é considerada. Entre as rotas alternativas, aquela que reduzir a sobrecarga com menor custo adicional é escolhida.

Para fazer o balanço de carga é preciso primeiro calcular a probabilidade de cada intervalo de tempo de cada DW ser atribuído à sua operação, ou seja, calcular as distribuições de carga nas janelas de demanda das operações. Feito isso, a distribuição de carga em um processador é igual a soma das distribuições de carga de todas as operações que necessitam deste processador.

Exemplo: Cálculo da distribuição de carga em um dw.

Considere uma operação com TP=3 e DW=6. O número de alocações possíveis para uma operação é igual a (DW-TP+1), que neste caso é 4. A Figura 3.7 mostra como é feito a distribuição de carga no dw.

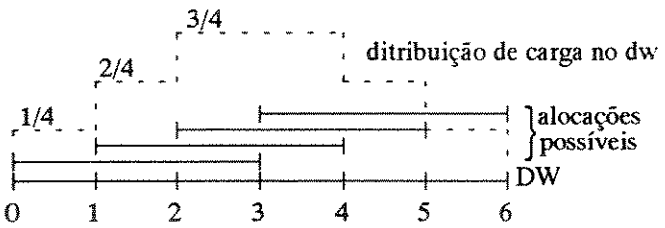


Figura 3.7- Distribuição de carga em um dw.

■

3- Planejamento de recursos

Decide qual é a próxima operação a ser alocada e qual intervalo de tempo será alocado a esta operação. Este módulo usa uma estratégia conhecida como estratégia do menor compromisso (*least commitment strategy* [LCS]), a qual tenta adiar as decisões de alocação o máximo possível, mantendo todas as opções em aberto. O LCS usa duas heurísticas sensíveis às interações dinâmicas entre as operações, para guiar a busca por uma solução factível: *graceful retreat*, que ordena as operações segundo suas criticalidades, selecionando a cada instante a mais crítica; e, *least impact*, que ordena os intervalos de tempo das operações pela sua crucialidade, selecionando o intervalo menos crucial (a criticalidade e a crucialidade são definidas a seguir). Após aplicar estas duas heurísticas faz-se a propagação de restrições, para verificar se a alocação que acaba de ser feita não produz nenhuma inconsistência e para garantir que a próxima alocação será consistente com as anteriores. As interações entre as operações são, portanto, modeladas considerando as restrições, criticalidades das operações e crucialidade dos intervalos de tempo dos processadores.

A criticalidade (crt) de uma operação é igual ao seu tempo de processamento divido pela duração de seu DW, o que significa que quanto mais critica for uma operação menor é o número de opções para sua alocação. Por isso a operação mais critica é selecionada primeiro. Se as tarefas possuem graus de importância diferentes, as criticalidades podem ser multiplicadas por pesos que refletem esta importância. A crucialidade (crc) de cada unidade de tempo de um processador é igual a soma das criticalidades de todas as tarefas que competem por aquela unidade de tempo. A crucialidade de um intervalo de tempo é igual a soma das crucialidades de todas as unidades de tempo que pertencem àquele intervalo.

Exemplo: Cálculo das crucialidades dos intervalos de tempo.

A Figura 3.8 mostra o processo de cálculo das crucialidades das unidades de tempo de um processador P com três operações (o1,o2,o3) com tempos de processamento (3,2,3), representadas pelos seus dw's.

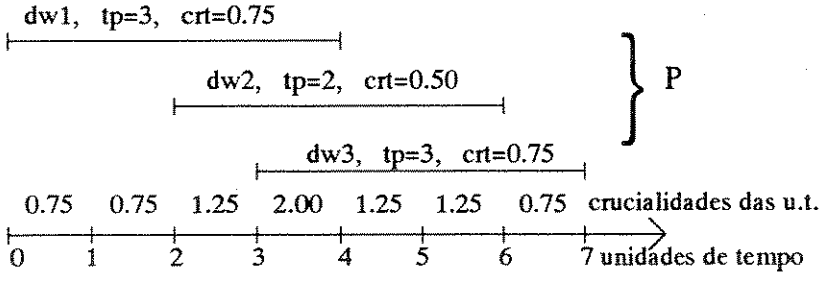


Figura 3.8- Crucialidades das unidades de tempo de um processador P.

A operação 3 pode ser alocada no intervalo [3,6] ou [4,7]. A crucialidade do intervalo [3,6] é $crc_{[3,6]}=crc_{[3,4]}+crc_{[4,5]}+crc_{[5,6]}=2+1.25+1.25=4.5$. A crucialidade do intervalo [4,7] é $crc_{[4,7]}=crc_{[4,5]}+crc_{[5,6]}+crc_{[6,7]}=1.25+1.25+0.75=3.25$. Como o intervalo [4,7] é o menos crucial, a operação 3 é alocada nele.

■

4- Gerenciamento de Conflitos

Se for detectada alguma inconsistência durante a propagação de restrições (ou seja, se o tempo de processamento de uma operação for menor que o seu DW), então o *schedule* deve ser revisto. A forma mais simples de fazer isto é através de um *backtracking* cronológico. No entanto, os autores optaram por não fazer o *backtracking* cronológico (alegando que ele sempre desfaz a última decisão tomada pelo sistema e esta nem sempre é a responsável pela inconsistência) usando duas outras técnicas para gerenciar os conflitos:

i) em primeiro lugar, o *scheduler* busca uma rota alternativa capaz de satisfazer a operação inconsistente no sistema;

ii) não havendo uma rota alternativa, a restrição de prazo de entrega do produto que contém a operação inconsistente é relaxada. O prazo de entrega é relaxado o mínimo suficiente para permitir que a operação em questão seja alocada. Após a alocação as restrições são propagadas. Nesta propagação, o horizonte de tempo das operações que sucediam a operação inconsistente precisam ser recalculadas (se alguma delas estava alocada, a alocação é desfeita). A partir daí o módulo de planejamento de recursos reassume o processo.

3.3.3.2- Sistema Micro-oportunista para *Scheduling job-shop* (CORTES) [Sadeh-89]

Este sistema é basicamente idêntico ao ISPS, com a diferença de que as heurísticas para ordenamento das variáveis e dos valores são um pouco mais elaboradas. Tanto o CORTES quanto o ISPS surgiram em resposta aos problemas de dois sistemas anteriores: o ISIS [Fox-83] e o OPIS [Smith-85].

O ISIS adota uma estratégia de alocação baseada em tarefas, onde o problema de *scheduling* é resolvido identificando a cada iteração a tarefa mais crítica e resolvendo os conflitos em torno daquela tarefa (ou seja, alocando todas as suas operações). O problema desta abordagem é que as interações entre as diferentes tarefas que competem pela utilização de um mesmo recurso são negligenciadas. O OPIS tenta resolver este problema adotando duas estratégias: uma baseada em recursos, que é responsável pela alocação das tarefas que competem pela utilização do recurso identificado como o gargalo do sistema; e outra baseada em tarefas, que é responsável pela alocação das tarefas que competem pela utilização dos demais recursos, sendo que as mais críticas são alocadas primeiro. Esta

estratégia foi denominada oportunista porque as alocações das tarefas são feitas ora utilizando uma estratégia, ora utilizando a outra, dependendo do estado da planta.

O CORTES e o ISPS, ao atribuir janelas para as operações e utilizar heurísticas de alocação baseadas nestas janelas juntamente com um mecanismo de propagação de restrições, permitem que as alocações das operações sejam feitas considerando simultaneamente as tarefas críticas e os gargalos do sistema. Esta é uma estratégia baseada em operações e foi denominada micro-oportunista [Sadeh-89].

Para resolver problema de *scheduling*, o CORTES, como o ISPS, focaliza a atenção nas operações, e as decisões de alocação são tomadas de maneira mais oportunística, utilizando duas heurísticas para guiar a busca:
- heurística de ordenamento das variáveis, que ordena as operações pelo valor de suas criticalidades;
- heurística de ordenamento dos valores, que ordena os intervalos de tempo pelo valor de suas sobrevivências (*survivability*).

Na versão centralizada do sistema (existe uma versão distribuída [Sycara-91]), que é idêntica à proposta de Keng [Keng-88], o *schedule* é construído iterativamente, selecionando a cada instante uma operação a ser alocada e um intervalo de tempo para esta alocação. Sempre que uma nova operação é alocada, novas restrições são formuladas, as quais precisam ser propagadas pelo sistema. Durante a propagação de restrições, se alguma inconsistência for detectada (violação de alguma restrição), o sistema busca uma solução alternativa realizando o *backtracking* cronológico. Do contrário uma nova operação é selecionada e o processo continua até que todas tenham sido alocadas.

Ordenamento das operações e dos intervalos de tempo

Uma operação é considerada crítica quando a sua demanda de recurso entra em conflito com as demandas de outras operações. As criticalidades das operações no ISPS eram calculadas dividindo o tempo de processamento da operação pelo seu DW. No CORTES a criticalidade é definida como sendo o máximo da distribuição de carga no DW. Para efeito de ordenamento das operações, estas duas heurísticas levam ao mesmo resultado. No entanto, no CORTES, as operações são ordenadas não apenas segundo as suas criticalidades, mas também segundo as criticalidades dos processadores (identificando o gargalo do sistema). Isto é feito da seguinte forma: primeiro é feita a distribuição de carga (aqui chamada de demanda agregada) nos processadores; em seguida é identificado o processador e o intervalo de tempo em que a demanda agregada é máxima (chamado de pico de contenção); e, finalmente, a operação que mais contribui para a demanda agregada neste intervalo de tempo é escolhida como a próxima operação a ser alocada.

Uma vez escolhida a próxima operação a ser alocada, é preciso decidir em que intervalo de tempo será feita esta alocação. No ISPS os intervalos de tempo eram ordenados segundo as suas crucialidades. O CORTES usa um conceito semelhante, denominado sobrevivência, porém de custo

computacional maior, para ordenar os intervalos de tempo. A sobrevivência de cada intervalo de tempo é definida como sendo a crucialidade do intervalo de tempo calculado a partir da distribuição de carga do dw da operação em questão dividido pela crucialidade do intervalo de tempo calculado a partir da demanda agregada do processador em questão. O instante de tempo em que a survivabilidade for máxima é escolhido para alocar a operação.

Os autores argumentam que, do ponto de vista do processador, as heurísticas de ordenamento das operações e dos intervalos de tempo podem ser consideradas políticas altruístas, uma vez que ao alocar uma operação elas levam em consideração as outras operações que necessitam daquele processador. No entanto, se esta proposta fosse aplicada a um problema com recursos compartilhados, do ponto de vista de um recurso compartilhado, estas heurísticas seriam até certo ponto egoístas, já que elas não consideram as operações pertencentes a outros processadores e que também estão competindo pela sua utilização. Neste caso seria então necessário generalizar os conceitos de criticalidade de uma operação e de crucialidade de um intervalo de tempo.

3.3.3.3- Sistema Baseado em Conhecimento para *Scheduling* de *Job-shop* (OPAL).

O OPAL [Bensana-88, Bel-89] é um sistema que utiliza três fontes de conhecimento para resolver o problema de *scheduling*:
1- Conhecimento Teórico: obtido da teoria de *scheduling* (análise de restrições [Erschler-76]);
2- Conhecimento Empírico: regras de prioridade e suas influências nos objetivos da produção;
3- Conhecimento Prático: restrições tecnológicas a serem satisfeitas em uma dada aplicação (obtido dos *schedulers*).

Estas três fontes de conhecimento são utilizadas pelos três módulos descritos abaixo para obter um *schedule*:

1- Módulo de Análise de Restrições (MAR): avalia as conseqüências das restrições de tempo sobre o seqüenciamento das operações. Ela pode gerar novas relações de seqüenciamento entre as operações e propagar estas restrições;

2- Módulo de Tomada de Decisões (MTD): fornece sugestões sobre o seqüenciamento das operações baseado no conhecimento prático e heurístico, quando o MAR não consegue alocá-las;

3- Supervisor: controla o diálogo entre o MAR e o MTD e constrói o *schedule* passo-a-passo de acordo com as decisões tomadas por estes módulos.

O *scheduling* é feito pelo MAR juntamente com um algoritmo de busca do tipo BS (*backtracking Search*). Quando o MAR não é capaz de seqüenciar as operações o BS é chamado para

enumerar as opções de alocação possíveis de acordo com as sugestões fornecidas pelo MTD. O MTD é responsável então pela solução dos conflitos, e possui várias heurísticas para escolher a melhor solução de acordo com o estado atual da planta. Alguns exemplos destas heurísticas são:
-seleção do processador m onde ocorre o maior número de conflitos, e resolução destes conflitos;
-seleção do conflito (i,j) onde i ou j é uma operação de particular interesse;
-seleção do processador onde as operações têm maior folga e resolução dos seus conflitos;
-etc.

Quando o sistema não é capaz de encontrar uma solução que satisfaça as restrições do problema, algumas delas devem ser relaxadas. O OPAL prevê a existência de uma quarta fonte de conhecimento com as preferências de relaxação para diversos casos, uma vez que decidir qual restrição deve ser relaxada pode não ser uma tarefa trivial. No entanto, nas versões apresentadas em [Bensana-88, Bel-89], esta fonte de conhecimento não era implementada.

Comentários:

Os três sistemas apresentados foram desenvolvidos para o *job-shop*. O objetivo dos três é apenas o de encontrar uma solução para o problema que satisfaça a suas restrições, sem se preocupar com a otimização de qualquer critério de desempenho da planta. O ISPS e o CORTES não consideram recursos compartilhados com demanda não unitária, ou seja, outros recursos além dos processadores. Em particular, a generalização do conceito de criticalidade e crucialidade não é evidente.

Para a problema proposto neste trabalho, o *flow-shop* com recursos compartilhados e alocações externas, o interesse destes sistemas está na metodologia de propagação de restrições que restringe as janelas de demanda levando a uma árvore de busca menor e se necessário a um *backtracking*.

3.3.4- *Scheduling* de *Flow-shops* com Recursos Compartilhados.

O problema do seqüenciamento de tarefas em *flow-shop* sob o critério *makespan* e sem restrições sobre os recursos compartilhados tem sido amplamente discutido na literatura [Bellman-82, Conway-76]. Normalmente, o algoritmo BAB é utilizado para encontrar a solução ótima. Na árvore de busca do BAB, cada nó corresponde a uma seqüência parcial de tarefas e não há a necessidade de realizar a busca a nível de operação, uma vez que se consideram seqüências de permutação.

Quando se consideram restrições sobre os recursos compartilhados o problema torna-se mais complexo, uma vez que fica mais difícil obter limitantes inferiores (*lower-bounds*-LB) para as seqüências parciais, como será visto na seção 3.3.4.2. O que se faz normalmente é ignorar estas restrições durante o BAB e, posteriormente, numa fase de factibilização modificar o *schedule* obtido para acomodá-lo ao perfil de oferta dos recursos [Kondili-93]. No entanto, se as restrições sobre os

recursos compartilhados são fortes, este procedimento pode alterar significativamente o custo da solução obtida pelo BAB, perdendo-se a otimalidade perseguida na fase de enumeração. Esta situação está retratada no exemplo abaixo:

Exemplo: Factibilização *a posteriori* [Rodrigues-92].

Problema: seqüenciar 4 tarefas em 3 processadores utilizando um recurso compartilhado com disponibilidade q = 10 (ur/ut). Os tempos de processamento e consumo de recurso de cada operação são dados na Tabela 3.1 abaixo:

TABELA 3.1- Tempos de proc. e cons. de recurso

	tempo proc.			cons. de recurso		
	O.1	O. 2	O.3	O.1	O.2	O.3
T.1	8	9	5	8	6	4
T.2	6	7	7	6	2	4
T.3	7	7	12	7	3	6
T.4	4	7	12	3	4	8

Uma das soluções ótimas para este problema considerando disponibilidade ilimitada do recurso em questão é a seqüência 4123 com *makespan* 47, como mostra a Figura 3.9. A factibilização desta seqüência para limitar o consumo do recurso a 10 ut/ur aumenta o makespan para 72, como mostra a Figura 3.10. A Figura 3.11 mostra o perfil de consumo de recurso para os dois casos. A solução factibilizada esta longe de ser a solução ótima do problema, que é a seqüência 1234 com *makespan* 61 (como será visto na seção 3.3.4.2).

■

Na indústria de processos químicos as utilidades são alguns dos recursos compartilhados cujos perfis de oferta são bem conhecidos. O compartilhamento destes recursos permite a execução simultânea de várias operações e sua utilização eficiente significa obter o máximo da capacidade instalada de produção. Como a relaxação da restrição sobre os recursos e posterior factibilização da solução encontrada não garante a qualidade desta solução, esta pode não ser uma estratégia aceitável do ponto de vista econômico.

Em [Rodrigues-92] foi desenvolvido um algoritmo do tipo BAB para resolver este problema. Uma extensa revisão bibliográfica feita neste trabalho mostra que o algoritmo proposto era o primeiro a resolver o problema de forma ótima, embora houvessem propostas de modelagem e solução do problema tanto na área da pesquisa operacional quanto na área de inteligência artificial. Em [Passos-93] o mesmo problema é tratado de forma hierárquica.

A seção 3.3.4.1 irá discutir brevemente o método BAB que é a base do algoritmo proposto em [Rodrigues-92] e que servirá de base também para o algoritmo que será proposto no capítulo 4. A seção 3.3.4.2 irá apresentar os algoritmos de estimação dos LB dos nós propostos por [Rodrigues-92], que também servirão de base para a proposta do capítulo 4.

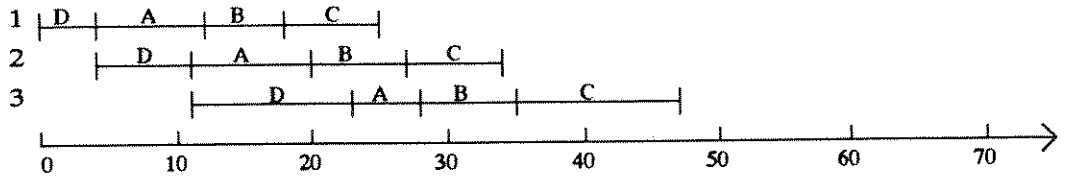


Figura 3.9- Solução sem limite para o recurso compartilhado

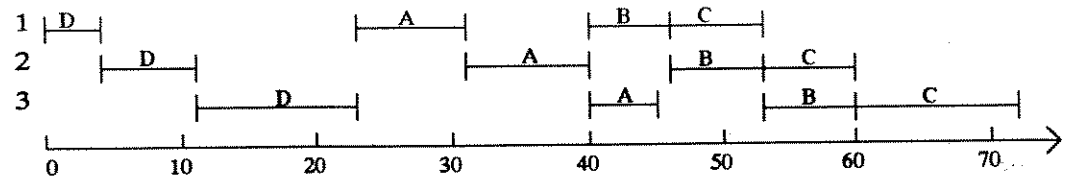


Figura 3.10- Factibilização da solução anterior para uma disponibilidade igual a 10.

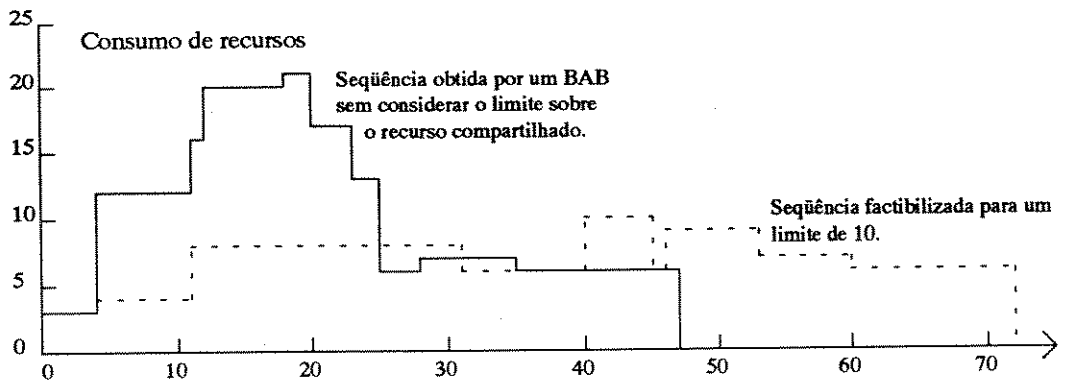


Figura 3.11- Perfil de consumo do recurso compartilhado.

3.3.4.1- O Método *Branch-and-Bound.*

O método BAB é uma técnica de enumeração implícita que procura uma solução ótima examinando sistematicamente subconjuntos de soluções factíveis [Horowitz-78]. Isto é feito em uma estrutura do tipo árvore (árvore de busca ou árvore de soluções), onde os nós representam soluções parciais do problema e os ramos ligações entre nós sucessivos. Para minimizar uma função objetivo **f**, o BAB aplica sucessivamente duas atividades, a busca (*branching*) e o controle (*bounding*), que permitem eliminar os subconjuntos de soluções (nós) que não incluem a solução ótima, até que reste uma única solução para o problema (a solução ótima!). A busca se concentra nas maneiras possíveis de exploração da estrutura em árvore e o controle em selecionar, dentre as várias alternativas de caminhos que podem ser percorridos, aquela que efetivamente será escolhida.

O controle é feito calculando um limitante inferior (*lower-bound* - LB) dos valores de **f** das soluções factíveis contidas em cada nó. O nó com menor LB é então decomposto (explodido) em seus

subconjuntos de soluções factíveis, criando novos nós que serão adicionados à árvore e que posteriormente poderão ser explodidos (denominados **nós abertos**), dependendo dos valores do LB e da forma de busca utilizada. As formas de busca possíveis são três: a busca-em-profundidade (*depth-first*), a busca-pelo-melhor (*best-first*) e a *beam-search*. Os processos de busca e controle podem podem ser agilizados pelo cálculo de um limitante superior (*Upper-Bound*-UB), geralmente definido como o valor mínimo de **f**, denominado **f***, de todas as soluções factíveis completas encontradas até o momento. No início da busca, quando não se conhece ainda nenhuma solução factível completa, **f*** é infinito.

A eliminação dos nós que não contêm a solução ótima é feita excluindo aqueles cujo LB é maior ou igual o limitante superior **f***. O resultado disto é a diminuição do número de soluções que serão efetivamente pesquisadas, ou seja, é a redução do espaço de busca do problema. Ocorrerá *backtracking* na árvore enquanto existir um nó com valor de LB menor que o valor de **f***. Quando o LB de todas as soluções incompletas forem maior ou igual ao **f***, então a solução factível com este **f*** é a solução ótima para o problema. Outra forma de reduzir o tamanho da árvore de busca é utilizar relações de dominância [Baker(b)-75].

A utilização eficiente de um BAB apresenta uma grande limitação, que é o cálculo da função objetivo (ou função de custo) **f** associada a cada nó, ou seja, a estimativa do LB. Esta função é normalmente composta de duas parcelas:
- uma parcela relativa ao custo associado à solução parcial representada no nó, que é, em princípio, facilmente calculada a partir dos dados do problema;
- uma parcela relativa ao custo mínimo necessário para completar a solução parcial do nó. Esta parcela é mais complexa e implica na utilização de funções para estimar o custo mínimo associado ao resto do caminho em direção à solução final. É necessário garantir que o custo total resultante da soma destas duas parcelas não será, em hipótese alguma, reduzido à medida que se caminha na árvore, pois é a monotonicidade da estimativa de **f** que garante a otimalidade da solução encontrada.

Quanto pior for a estimativa do LB em cada nó, maior será a freqüência de *backtracking*. No entanto, quanto melhor for esta estimativa, maior será o seu custo computacional, devendo haver portanto uma relação de compromisso entre a freqüência de *backtracking* e a exatidão da função utilizada para estimar os valores de **f**.

3.3.4.2- BAB para *Flow-shops* com Recursos Compartilhados.

A estimativa do LB das seqüências parciais (nós) é o ponto principal de qualquer BAB. Quando o critério de otimização utilizado é o *makespan*, para calcular os LB, é necessário conhecer os instantes de início possíveis (I) das operações das tarefas ainda não seqüenciadas. Se não existe restrições sobre os recursos compartilhados e a armazenagem intermediária é ilimitada existe uma relação de recorrência simples para calcular estes instantes de início, que é a relação 3.3. Esta relação estabelece que a tarefa *i*

pode iniciar no processador *j* após a tarefa *i* ter sido liberada pelo processador *j-1* e o processador *j* liberado pela tarefa *i-1*. Assim o instante de início das operações, e conseqüentemente o cálculo do LB, só depende das seqüência das tarefas e da duração de suas operações. A grande conseqüência disto tudo é que os procedimentos de seqüenciamento e alocação podem ser separados [Rodrigues-92].

$$I_{(i,j)} = MAX\{C_{(i,j-1)}, C_{(i-1,j)}\} \tag{3.3}$$

Quando os recursos compartilhados são limitados, não é mais possível calcular os instantes de início das operações conhecendo apenas a seqüência das tarefas e a duração de suas operações. Será necessário conhecer o perfil de utilização de cada recurso compartilhado, pois os instantes de início das operações irá depender também de suas disponibilidades, que mudam dinamicamente no decorrer do *scheduling*. Isto faz com que as decisões de alocação devam ser tomadas juntamente com as decisões de seqüenciamento.

O instante de início da tarefa *i* no processador *j* deve então ser calculado da seguinte forma:

$$I^*_{(i,j)} = MAX\{C_{(i,j-1)}, C_{(i-1,j)}, \underset{0 \le r \le R}{MAX}\{C_{in,r}(i,j)\}\} \tag{3.4}$$

onde:

r=1,2,...,R são os recursos compartilhados, e

Cin,r(i,j) é o instante mais cedo que o perfil de disponibilidade do recurso *r* permite o processamento da operação (i,j), que pode ser obtido por propagação de restrições.

Para utilizar o BAB falta ainda definir um procedimento para estimar o limitante inferior (LB) de cada nó monotonicamente.

Cálculo do *Lower-Bound* (LB)

Para calcular o LB, que é uma estimativa do *makespan* de cada nó (ou seqüência parcial) da árvore de busca, considere a seguintes definições:

- cada nó é caracterizado por um conjunto ordenado **K** de tarefas seqüenciadas (representando a seqüência parcial K), um conjunto U de tarefas não seqüenciadas, e pelo seu LB;
- a disponibilidade temporal do recurso *r* é dada por $q^r(t)$ [unidade de recurso/unidade de tempo (ur/ut)];
- a demanda instantânea do recurso *r* pela operação (i,j) é dada por $Q^r_{ij}(t)$ [ur/ut].

O LB será obtido na forma usual pela adição de duas partes: uma relacionada à seqüência parcial K (esta parte é facilmente obtida) e a outra que será uma estimativa do tempo necessário para processar as tarefas do conjunto U.

Algoritmos A e B [Rodrigues-92]

No nó raiz, o LB é definido como o horizonte mínimo necessário para processar todas as tarefas considerando somente a disponibilidade dos recursos. Este tempo T é obtido da seguinte expressão:

$$T = \max_{1 \le r \le R} \{T^r\}, \text{ com } \int_{t=0}^{t=T} q^r(t).dt = \sum_{i=1}^{N} \sum_{i=j}^{M} \int_{\tau=0}^{\tau=tij} Q_{ij}^r(\tau).d\tau \qquad (3.5)$$

que estabelece que o *makespan* inicial T será igual ao tempo necessário para processar todas as tarefas partindo da hipótese que todos os recursos serão completamente utilizados.

Para os nós intermediários a contribuição das tarefas do conjunto U será estimada com base no gráfico apresentado na Figura 3.12. Esta figura é dividida em três regiões:

- a Região I começa em t=0 e termina em t=C(K,1), que é por definição o tempo de conclusão da primeira operação da última tarefa do seqüência parcial K. A primeira operação de qualquer tarefa candidata a fazer parte da seqüência parcial K só poderá começar em t=C(K,1). Como conseqüência disto, os recursos que eventualmente deixaram de ser utilizados em I serão perdidos;

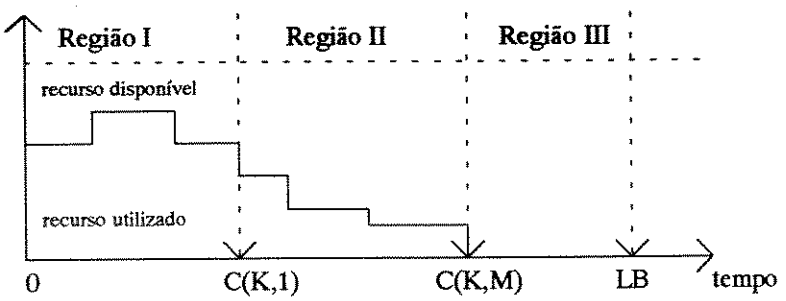


Figura 3.12- Regiões de uma solução parcial.

- a Região II começa em t=C(K,1) e termina em t=C(K,M), o tempo de conclusão da última operação da seqüência parcial K. Nesta região o BAB poderá ainda programar algumas operações, dependendo da disponibilidade dos recursos.

- a Região III vai de t=C(K,M) até t=LB, a estimativa de tempo de conclusão corrente. Esta região está totalmente livre, sem nenhuma tarefa seqüenciada. Para uma seqüência completa, LB=C(K,M).

A proposta do algoritmo A é calcular o LB da seqüência parcial K acrescentando ao LB da seqüência parcial K-1 o tempo necessário par obter a quantidade de recurso perdido na região I. Cada recurso definirá um valor de LB e o máximo entre eles será o LB do nó em questão.

O algoritmo A pode ser melhorado levando em consideração que o instante de início da tarefa que deseja fazer parte da seqüência parcial K (tarefa candidata) pode não ser C(K,1). Isto ocorrerá se os recursos remanescentes na Região II não forem suficientes para processar a primeira operação da tarefa candidata em C(K,1). Esta situação esta ilustrada na Figura 3.13. A região hachurada desta figura representa o consumo de recursos da primeira operação da tarefa candidata. O perfil de consumo desta operação (valor AB constante) não permite o seu início em t=C(K,1), mas em t=t*. A conseqüência disto é que os recursos não utilizados entre C(K,1) e t* serão perdidos. Na Figura 3.13 tem-se ainda t*=Cin,r.

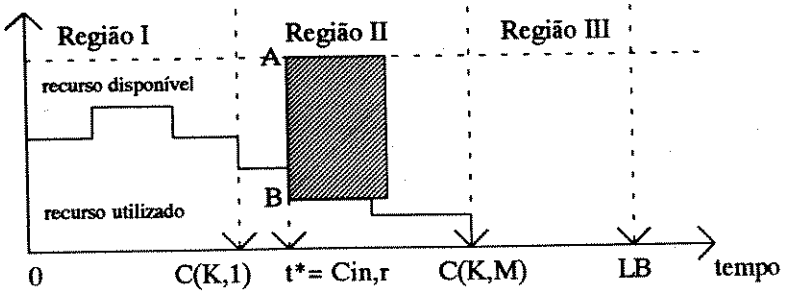


Figura 3.13- Determinação de t* e Cin,r.

No algoritmo B esta informação é usada para obter uma estimativa melhor do LB, de acordo com o mesmo esquema de A. O procedimento consome mais tempo porque: i) para a primeira operação de cada tarefa candidata deve-se obter o t* para cada recurso r; ii) para cada tarefa o maior t* é armazenado, pois ele define a perda de Recursos na região II caso a tarefa seja alocada; iii) a tarefa que provocar o menor acréscimo no LB irá definir o seu valor (este procedimento preserva a monotonicidade da estimativa do LB). A passagem do algoritmo A para o B é a incorporação de uma estratégia do tipo *lookahead*, que melhora a estimativa do *makespan* no nó analisando apenas a primeira operação de cada tarefa.

Algoritmos C1 e C2

Os algoritmos A e B estimam o valor do *makespan* com base apenas na disponibilidade dos recursos. Entretanto, a seqüência tecnológica das tarefas também contribui para a definição do *makespan* e mais, ela é fundamental quando as restrições sobre os recursos não são muito fortes. Os algoritmos C1 e C2 são propostos como modificações dos algoritmos A e B, respectivamente, onde o LB será o máximo entre o valor calculado com base na disponibilidade de recursos e o valor calculado com base no encadeamento das operações.

Existem várias funções para estimar um limitante inferior para o *makespan* com base no encadeamento das operações [Bellman-82]. Dentre elas a *Full Machine Based Bound* (FMBB) [Baker(a)-75] é a de maior aceitação e por esse motivo ela foi a escolhida.

As figuras 3.14 e 3.15 mostram a carta de Gantt e o perfil de consumo do recurso compartilhado para o problema proposto acima, resolvido pelos algoritmos A, B, C1 e C2 (todos levam à mesma solução final, o que muda é o número de nós sondados em função da melhor (C2) ou pior (C1) estimativa do *makespan*)

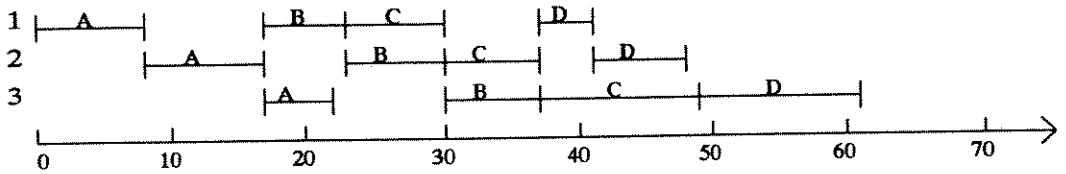


Figura 3.14- Carta de Gantt da seqüência ótima ABCD.

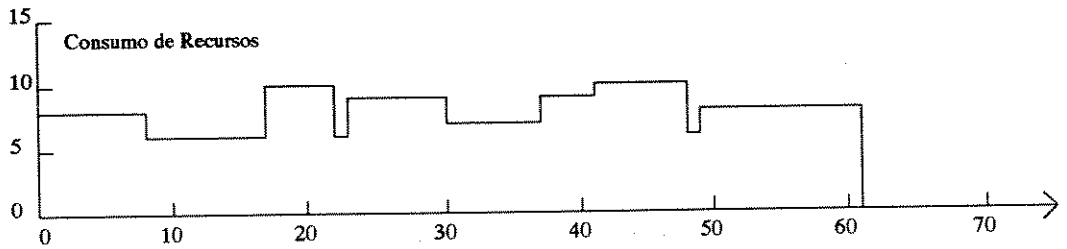


Figura 3.15- Perfil de consumo do recurso compartilhado

Observa-se que a solução encontrada aqui é completamente diferente da solução encontrada na seção 3.3.4, onde o problema era resolvido por um BAB sem considerar o limite do recurso e a solução encontrada era factibilizada para respeitar esta restrição.

3.4- Conclusão.

As técnicas de IA apresentadas neste capítulo foram propostas originalmente para o problema de *job-shop*. Como o *flow-shop* é um caso particular deste, era de se esperar que estas técnicas fossem diretamente aplicáveis a ele, como de fato são. Entretanto, como exposto em [Graves-86], as técnicas de IA são tipicamente dirigidas por restrições, ou seja, o seu objetivo é satisfazer o maior número possível de restrições do problema, falhando quando o objetivo é otimizar algum critério de desempenho. Em

[Rodrigues-92] mostra-se a dificuldade de minimizar o makespan em flow-shops com recursos compartilhados utilizando uma proposta semelhante ao CORTES [Sadeh-89].

As técnicas de PO como o BAB, por sua vez, são tipicamente dirigidas por objetivos, falhando quando se deseja tratar o problema com um elevado número de restrições. No caso do *flow-shop*, as propostas encontradas na literatura geralmente não consideram a possibilidade de alocações definidas externamente ou, quando o fazem, elas são consideradas restrições flexíveis (que podem ser relaxadas mediante um custo). O mesmo acontece com os recursos compartilhados, que são considerados apenas em uma fase de factibilização, que é feita após a solução do problema relaxado. O único algoritmo que se conhece para encontrar a solução ótima do problema de minimização do *makespan* em *flow-shops* com recursos compartilhados é o algoritmo proposto em [Rodrigues-92].

Será proposto no próximo capítulo um algoritmo que tira vantagem das técnicas de IA e de PO para resolver o problema proposto no capítulo 2. Será utilizado um algoritmo BAB para encontrar a solução ótima do problema, que irá trabalhar com janelas de demanda das operações permitindo o uso de análise de restrições, de propagação de restrições e de heurísticas de alocação, como forma de reduzir o espaço de solução e de busca do problema.

Capítulo 4

Seqüenciamento de *Flow-shops* com Tarefas Alocadas Externamente Usando a Abordagem *Branch-and-Bound.*

4.1- Introdução

O objetivo deste capítulo é estudar o problema de *scheduling* em *flow-shops* com tarefas alocadas externamente e restrições sobre os recursos compartilhados. Será desenvolvido um algoritmo, que é a principal contribuição deste trabalho, que utiliza ao mesmo tempo:
i) um algoritmo de busca do tipo BAB, com limitante inferior calculado com base na utilização dos recursos compartilhados, como proposto em [Rodrigues-92];
ii) um algoritmo de análise de restrições, como proposto em [Erschler-76], adaptado ao caso de *flow-shop*, para lidar com as restrições de tempo do problema, na forma de janela de demanda de suas operações;
iii) um mecanismo de propagação de restrições, como proposto em [Keng-88], para lidar com as restrições geradas pela alocação de tarefas e pela análise de restrições.

Os mecanismos de análise e propagação de restrições em ii) e iii) têm como principal objetivo restringir o espaço de busca do algoritmo BAB preservando a otimalidade da solução. Propõe-se também um mecanismo heurístico adicional para reduzir ainda mais a árvore de busca do BAB, utilizando uma extensão do conceito de criticalidade para a situação em que há recursos compartilhados.

Estas técnicas são utilizadas em conjunto com o objetivo de obter solução ótima para problemas de *flow-shop* mais reais do que os que são geralmente tratados na literatura de pesquisa operacional.

4.2- Descrição do Problema

O problema pode ser definido da seguinte forma: são dadas N tarefas, das quais J (J≤N) foram alocadas externamente, M processadores, R recursos compartilhados, armazenagem intermediária ilimitada, e o objetivo é seqüenciar as tarefas de forma a minimizar o *makespan*, considerando seqüências de permutação.

Uma alocação externa consiste na atribuição de um tempo de pronto (*ready time*-RT) e de um prazo de entrega (*duedate*-DD) para uma tarefa, ou seja, na atribuição de uma janela de tempo "conveniente" para ela. Esta janela de tempo deverá sempre ser maior ou igual ao somatório dos tempos de processamento das operações da tarefa. Sendo igual a alocação é denominada **fixa**, caso contrário ela

é denominada **flexível**. As tarefas alocadas externamente são denominadas **tarefas externas**, e as demais **tarefas internas**.

A partir do RT e do DD de uma tarefa é possível construir janelas de tempo para cada uma de suas operações, através da propagação intra-ordem destes limites de tempo (de acordo com a seção 3.3.2). O conceito de alocação fixa e flexível se estende então para cada operação. Se a alocação é fixa, a operação possui um único instante de início factível (*start time*-ST) e, no seqüenciamento, ela terá obrigatoriamente que ser alocada neste instante. Quando a alocação é flexível, ela terá um instante de início mais cedo (*earliest beginning time*-EBT) e um último instante de início factível (*latest beginning time*-LBT). O ST da operação pode ser qualquer valor dentro deste intervalo de tempo, cabendo ao algoritmo que fará o seqüenciamento determinar qual é o melhor. A diferença entre o LBT e o EBT de uma operação é definido como sendo a sua **folga**. O LBT de uma operação é igual ao seu prazo de entrega (*latest finish time*-LFT) menos o seu tempo de processamento.

Algumas situações em que é possível atribuir, *a priori*, janelas para certas tarefas são:

- uma tarefa (ou uma operação) necessita de um recurso que está disponível em um determinado intervalo de tempo;

- existem tarefas que não podem ser iniciadas em t=0, p.ex. porque devem ser satisfeitas restrições de precedência tecnológica entre tarefas;

- existem tarefas com prazo de entrega bem definido, p. ex. para satisfazer as exigências de um cliente ou por uma questão de estabilidade do produto final;

- no horizonte tempo em que se deve realizar o *scheduling* existem períodos em que um ou mais processadores devem permanecer inativos (por exemplo, para manutenção). Neste caso, este período pode ser modelado como uma operação que possui uma alocação fixa, com RT no início e com DD no final do período inativo, e com consumo de recursos igual a zero;

- existem tarefas de execução periódica devido a demanda estável dos produtos.

Para cada planta específica certamente haverá várias outras situações que, baseadas nas restrições impostas pelo processo produtivo, nas exigências dos clientes, na experiência e no conhecimento adquirido na operação da planta, permitirão que sejam definidas janelas para algumas tarefas.

O problema será abordado do ponto de vista determinístico e estático, ou seja, todos os dados necessários para resolvê-lo são conhecidos *a priori* e com exatidão. Além do mais, embora grande parte da literatura de *scheduling* trate o DD como uma restrição flexível (que pode ser relaxada mediante um

certo custo [Baker-74, Bellman-82, Keng-88]), aqui ele será tratado como uma restrição rígida, que deve ser satisfeita por uma questão de factibilidade da solução.

4.3- Solução do Problema Utilizando a Técnica *Branch-and-Bound* (BAB)

O problema de *flow-shop* proposto e solucionado por [Rodrigues-92] consiste em determinar o exato momento em que as operações de cada tarefa devem ser realizadas de forma a minimizar o *makespan*, maximizando a utilização dos recursos compartilhados. O problema aqui proposto, *flow-shop* com alocações externas, adiciona duas novas decisões ao problema anterior:
1- determinar quando as tarefas com RT≠0 devem fazer parte do processo de solução do problema, e
2- determinar quando a alocação de uma tarefa provoca a violação do DD de outra tarefa, já que existem tarefas com prazos de entrega bem definidos.

Neste trabalho é proposto um método para resolver este problema, o qual possui as seguintes características:

- para encontrar a solução ótima, é utilizado um algoritmo de busca do tipo BAB. A forma de busca utilizada é a busca-em-profundidade (BP) porque o método utiliza o limitante superior (*upper-bound* (UB)) quando este está disponível, e a busca-em-profundidade permite, em geral, obter este limitante superior mais rapidamente do que a busca-pelo-melhor (BPM). Esta última forma de busca pode também ser utilizada, com pequenas modificações. O cálculo do limitante inferior (*lower-bound* (LB)) de cada nó é feito buscando minimizar o *makespan* pela maximização da utilização dos recursos compartilhados, através dos algoritmos C1 e C2 propostos em [Rodrigues-92] e descritos na seção 3.3.4.

- para decidir quais tarefas externas precisam ser consideradas durante a execução do BAB, utiliza-se o conceito das regiões I, II e III descrito na seção 3.3.4. Desta forma, na expansão de cada nó da árvore, são consideradas apenas as tarefas externas com instante de início inferior ou igual a C(K,M), ou seja, com início inferior ao tempo de conclusão, na planta, da seqüência parcial K correspondente ao nó que está sendo expandido;

- para permitir a verificação da factibilidade da solução que o BAB está construindo, são definidas janelas de tempo para todas as operações da planta. A atualização destas janelas durante o processo de busca permite uma análise consistente das restrições temporais do problema, visando evitar os caminhos que levam a um *dead-end*. O limitante superior é utilizado como prazo de entrega máximo para as tarefas internas;

- para manter a consistência das janelas das operações durante a busca pela solução ótima, o método proposto faz uso intensivo das técnicas de propagação de restrições intra-ordem e inter-ordem descritas na seção 3.3.2.1, o que resulta na diminuição do espaço de busca do problema.

Resumindo, o algoritmo proposto consiste em um pré-processamento que é aplicado a cada nó de uma árvore de busca, antes de sua explosão, para selecionar, dentre todas as candidatas a fazer parte da seqüência parcial do nó, aquelas que potencialmente levarão a uma seqüência final factível. O objetivo é chegar a um conjunto mínimo de candidatas, que no limite conterá apenas uma tarefa. Isto aproxima o BAB da BS (busca com *backtracking*) utilizada em [Sycara-91, Keng-88], onde cada nó dá origem a um único descendente. A principal diferença reside no fato de que a necessidade de *backtracking* é reduzida, uma vez que a determinação da candidata é feita de forma que os caminhos que levam a infactibilidades são evitados antecipadamente.

A estratégia de selecionar as tarefas candidatas para alocação é semelhante à estratégia proposta em [Smith-92], que seleciona os alvos que devem ser fotografados pelo telescópio Hubble, maximizando a sua utilização e minimizando o número de alvos que em um determinado período deixam de ser fotografados. A diferença é que nesta solução não está embutido nenhum mecanismo de busca, além de se tratar de um caso muito específico e monoprocessador.

A determinação do conjunto de tarefas candidatas, denominado **CAN**, é feito através da análise das restrições temporais originadas i) pelos limites de tempo das tarefas e ii) pela escassez de recursos compartilhados. Para que esta análise seja possível, é necessário atribuir janelas de tempo para todas as operações da planta; e para que ela seja consistente é necessário atualizar as janelas das operações das tarefas ainda não seqüenciadas durante o processo de busca.

Para descrever o método proposto, são analisados de forma separada os três itens abaixo:
- Condições de factibilidade de uma solução para o problema de *flow-shop;*
- Construção e atualização das janelas das operações das tarefas ainda não seqüenciadas;
- Determinação do conjunto de tarefas candidatas.

4.3.1- Condições de Factibilidade de uma Solução para o *Flow-shop*: Análise de Restrições.

As condições de factibilidade propostas por Erschler em [Erschler-76, Erschler-86] e descritas na seção 3.3.2.2 são aplicáveis a qualquer problema de *scheduling* em que possam ser definidas janelas de tempo factíveis para suas operações. Elas podem portanto ser aplicadas ao problema de *flow-shop* com tarefas alocadas externamente e com restrições sobre os recursos compartilhados.

Como o problema é resolvido através de um BAB, o que se propõe é a aplicação das condições de factibilidade às soluções parciais que podem se originar de cada nó da árvore de busca, antes de sua explosão propriamente dita. A verificação destas condições pode dar origem à modificação das janelas de tempo das operações das tarefas candidatas a fazer parte da seqüência (seção 4.3.2), bem como à eliminação, na expansão do nó, de algumas destas tarefas (seção 4.3.3).

Serão verificadas as condições de factibilidade para aquelas operações que utilizam um mesmo processador, e que portanto não podem ser sobrepostas no tempo. Uma vez determinado que, para manter a factibilidade da solução que se esta construindo, a operação (i,m) deve preceder a operação (j,m), então a tarefa i deve obrigatoriamente preceder a tarefa j, a fim de que sejam preservadas as seqüências de permutação. Não serão verificadas as condições de factibilidade para as operações de processadores diferentes (que também podem ter a sua sobreposição temporal proibida pela disponibilidade limitada de recursos compartilhados) porque não é possível tirar nenhuma conclusão imediata, a nível de tarefas, do fato de que (i,m) precede (j,n). No entanto, como será visto na seção 4.3.3.2, através de simulação da utilização dos recursos compartilhados é possível tirar conclusões de seqüenciamento entre tarefas.

Assim , dado um processador m e duas operações (i,m) e (j,m), cada uma podendo ser uma operação real ou fictícia, esta última representando um conjunto de operações (seção 3.3.2.2), define-se os seguintes termos:

-$TP=TP_{im}+TP_{jm}$ = tempo necessário para processar as duas operações;
-$T_{ij}=LFT_{jm}-EBT_{im}$ = janela de tempo disponível para processar (i,m) e (j,m), supondo que i será alocada antes de j;
-$T_{ji}=LFT_{im}-EBT_{jm}$ = janela de tempo disponível para processar (i,m) e (j,m), supondo que j será alocada antes de i.

A partir destas definições e das regras A.1 e A.2 (apresentadas na seção 3.3.2.2) é possível chegar às seguintes regras e conclusões úteis a respeito do nó que se esta analisando:

R1: Se $TP \leq Tij$ e $TP \leq Tji$, Então:

C1- não é possível tirar qualquer conclusão. A princípio qualquer ordenamento das tarefas satisfaz às suas restrições temporais.

R2: Se $TP > Tij$ e $TP > Tji$, Então:

C2- não existe ordenamento das tarefas i e j que satisfaça suas restrições de tempo;

R3: Se $TP > Tji$ e $TP \leq Tij$, Então:

C3- se i e j são tarefas reais, para manter a factibilidade da solução i deve preceder j;

C4- se i é uma tarefa fictícia, para manter a factibilidade da solução, $i1 \vee i2 \vee \ldots \vee in$ deve preceder j;

C5- se j é uma tarefa fictícia, para manter a factibilidade da solução, i deve preceder $j1 \vee j2 \vee \ldots \vee jn$;

onde $\vee$ é o operador lógico OU.

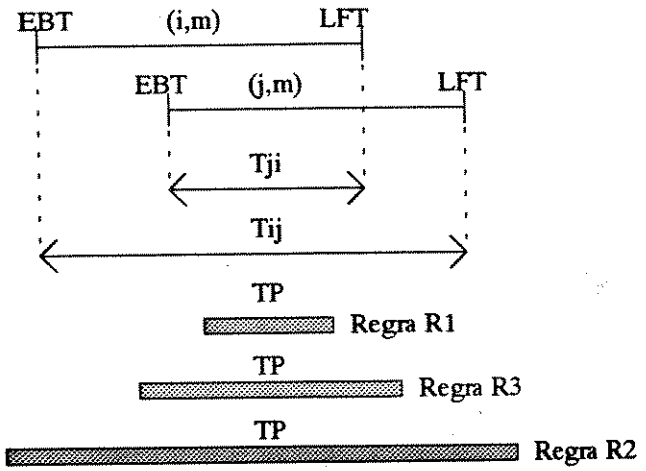


Figura 4.1- Condições de factibilidade de uma solução.

A aplicação destas regras está ilustrada na Figura 4.1. As conclusões resultantes da aplicação destas regras podem conduzir à elaboração de novas restrições temporais que, por sua vez, exigem atualização das janelas de demanda das operações através da propagação de restrições. Isto poderá resultar no atraso da programação de algumas tarefas, como será visto nas próximas seções.

Os termos C1, ..., C5 serão utilizados no decorrer deste trabalho com duplo sentido: ora para designar conclusão de seqüenciamento entre tarefas, ora para designar conflito entre elas.

4.3.2- Definição e Atualização de Janelas para as Operações

A possibilidade de conflito na alocação entre tarefas internas e tarefas externas, e a conseqüente eliminação de tarefas de CAN, é analisada a partir das suas janelas de tempo factíveis, utilizando as regras e conclusões da seção anterior. Para as tarefas externas estas janelas estão definidas desde o início. Para as tarefas internas o RT (tempo de pronto) é por definição zero e o DD (prazo de entrega) é definido como sendo TF, onde:

$$TF = \sum_{i=1}^{N} \sum_{j=1}^{M} TP_{ij}, \qquad (4.1)$$

representa o horizonte de tempo que seria necessário para processar todas as tarefas se, dado a escassez dos recursos compartilhados, não fosse permitida a sobreposição temporal de nenhuma de suas operações. Supõe-se que a definição das janelas externas e que a disponibilidade de recursos permite satisfazer pelo menos este horizonte. Não haveria problema algum em definir o DD das tarefas internas maior do TF ou até mesmo infinito, mas quanto menor este valor melhor dirigida será a busca nas

primeiras iterações. Freqüentemente, em problemas reais, TF será dado externamente como o horizonte de tempo em que se está resolvendo o problema de *scheduling*.

De posse das janelas para as tarefas, tanto internas como externas, pode-se obter facilmente as janelas correspondentes às suas operações, utilizando as relações 3.1 e 3.2. A atualização destas janelas passa a ser então da maior importância, pois ela vai garantir a consistência dos limites de tempo das operações permitindo que a análise de restrições tire conclusões acertadas a respeito da factibilidade da solução que o BAB está construindo.

A atualização das janelas das operações ocorre quando:

i) um nó vai ser explorado. Sempre que um nó vai ser explorado é necessário calcular o ST (que corresponde ao início da janela de tempo factível) de todas as tarefas candidatas a fazer parte da seqüência parcial deste nó. O ST de cada tarefa é, a princípio, MAX{RT,C(K,1)}, ou seja, o máximo entre o RT da tarefa e o instante de término da última tarefa da seqüência parcial K no primeiro processador. No entanto, com foi discutido na seção 3.3.4, devido às limitações nos recursos o início possível de uma tarefa é t*, t*≥C(K,1). Após a atualização dos ST das tarefas, os limites de tempo de cada uma de suas operações são determinados pela relação 3.1 (propagação para frente das restrições intra-ordem). Deve-se notar que o cálculo (propagação de restrições) necessário para determinar t* não é um trabalho adicional no caso em que a tarefa para o qual ele está sendo calculado seja efetivamente uma candidata não eliminada do conjunto **CAN**;

ii) o BAB encontra um novo limitante superior f*. Uma vez que se encontrou uma solução factível com *makespan* igual a f*, qualquer solução que leve a um *makespan* maior do que este não interessa mais, portanto todas as tarefas (internas e externas) ainda não seqüenciadas passam a ter DD=MIN{DD,f*}. O limitante superior passa então a ser uma restrição do problema que não pode ser relaxada, o que irá provocar a redução do espaço de busca do problema. Após a atualização dos DD das tarefas, os limites de tempo de cada uma de suas operações são determinados pela relação 3.2 (propagação para trás das restrições intra-ordem);

iii) a análise de restrições determina que existe alguma relação de seqüenciamento entre as tarefas candidatas. Assim, as conclusões C2, C3 e C4 da seção 4.3.1 são utilizadas para redefinir as janelas das operações, da seguinte forma:

C3- se i deve preceder j no processador m, então

$$EBT_{(j,m)}=MAX\{EBT_{(j,m)},EBT_{(i,m)}+TP_{(i,m)}\};$$

$$LFT_{(i,m)}=MIN\{LFT_{(i,m)},LBT_{(j,m)}\}.$$

C4- se i1 ∨ i2 ∨ ... ∨ in deve preceder j no processador m, então

$$EBT_{(j,m)}=MAX\{EBT_{(j,m)},MIN\{EBT_{(i1,m)}+TP_{(i1,m)},\ldots,EBT_{(in,m)}+TP_{(in,m)}\}\}.$$

C5- se i deve preceder j1 ∨ j2 ∨ ... ∨ jn no processador m, então

$$LFT_{(i,m)}=MIN\{LFT_{(i,m)},MAX\{LBT_{(j1,m)},\ldots,LBT_{(jn,m)}\}\}.$$

Após a determinação dos EBT's e LFT's das tarefas i e j no processador m, é feita a propagação intra-ordem destes novos limites de tempo, de acordo com as relações 3.1 e 3.2.

4.3.3- Determinação do Conjunto de Tarefas Candidatas (CAN).

Para o problema proposto, cada nó *i* da árvore de busca do BAB é caracterizado por um conjunto ordenado **K** de tarefas seqüenciadas, um conjunto **U** de tarefas internas ainda não seqüenciadas, um conjunto **J** de tarefas externas não seqüenciadas, e pelo perfil de consumo de cada um dos recursos por parte das tarefas do conjunto **K**. Para permitir a explosão do nó *i* é preciso determinar, a partir de **U** e **J**, o conjunto **CAN** das tarefas que darão origem aos seus descendentes.

Na técnica BAB clássica todas as tarefas não seqüenciadas são igualmente candidatas na expansão do nó, ou seja, na situação que se analisa todas as tarefas do conjunto **U** e **J** seriam candidatas. Entretanto, algumas das tarefas alocadas externamente e ainda não seqüenciadas podem ter que ser iniciadas dentro da Região II onde já houve a alocação de tarefas e, portanto, consumo de recursos. Existe então a possibilidade de que a alocação de alguma tarefa pertencente a **U**, mesmo otimizando o tempo, inviabilize a programação de alguma das tarefas de **J**. Além do mais, como no caso geral não existe nenhum ordenamento temporal entre as tarefas de **J**, estas competem entre si pela utilização do tempo e dos recursos compartilhados, podendo a programação de algumas delas inviabilizar outras. Obviamente esta verificação pode ser feita *a posteriori*, mas isto vai implicar em maior *backtracking* no BAB.

O que se propõe então é a determinação de um conjunto **CAN** inicial e, a partir dos testes definidos na seção 4.3.1, eliminar deste conjunto aquelas tarefas que inviabilizam uma ou mais tarefas de **J**, com base nas janelas de demanda das operações das tarefas ainda não seqüenciadas (seção 4.3.3.1) e no comprometimento de recursos na Região II (seção 4.3.3.2). No que diz respeito às tarefas internas, inicialmente **CAN** conterá todas aquelas que ainda não foram seqüenciadas. Dentre as tarefas externas não seqüenciadas, são incluídas inicialmente em **CAN** aquelas tarefas cujo RT se encontra dentro da região II. Ou seja, para toda tarefa $Ji \in \mathbf{J}$, $Ji \in \mathbf{CAN}$ se $RT(Ji) \leq C(K,M)$

4.3.3.1- Análise de Conflitos no Tempo entre Tarefas (Externas e Internas)

O objetivo é a determinação do conjunto final de tarefas candidatas à expansão do nó. Dispondo das janelas factíveis de todas as operações das tarefas internas e externas é possível aplicar ao conjunto inicial **CAN** as regras que resultam das condições de factibilidade propostas em [Erschler-76], que para o caso do *flow-shop* levam às 5 conclusões da seção 4.3.1. Estas conclusões levam às seguintes ações de eliminação de tarefas de **CAN**:

C1→ não é possível eliminar nenhuma das tarefas analisadas;
C2→ é impossível satisfazer as restrições de todas as tarefas, então a solução parcial que está sendo analisada é infactível e deve ser abandonada, ou seja, o nó é eliminado e ocorre o *backtrack*;
C3→ a tarefa j é eliminada do conjunto de tarefas candidatas;
C4→ a tarefa j é retirada do conjunto CAN, porque sendo a próxima tarefa da seqüência ela causará a perda do DD de pelo menos uma das tarefas de i;
C5→ Embora tenha sido detectado um conflito entre a tarefa i e a tarefa fictícia j, não é possível determinar, sem uma análise adicional, qual ou quais tarefas seriam as responsáveis pelo conflito.

As conclusões C1, C2, C3 e C4 permitem tomar uma decisão segura de eliminar ou não tarefas do conjunto CAN, com a conseqüente eliminação do conflito entre as tarefas. A conclusão C5 é mais difícil de ser analisada, razão pela qual ela será vista com mais detalhes na seção abaixo.

Utilização de Heurísticas para Solução de Conflitos do Tipo C5.

Quanto à conclusão C5, não é possível no nó que se está analisando tomar uma decisão segura de eliminar alguma tarefa. Sabe-se com certeza que o conflito C5 não é causado por uma única tarefa de j, porque se assim fosse isto teria sido detectado pela regra R3:C3, e a tarefa causadora do conflito já teria sido eliminada de CAN. Portanto, o que se sabe com certeza é que todas as tarefas pertencentes à tarefa fictícia não poderão ser alocadas antes da tarefa individual.

Restam então duas duas ações a serem tomadas diante deste conflito:
1- seguir adiante sem tomar qualquer decisão de eliminação de tarefas. Nas próximas iterações o algoritmo terá mais elementos para tomar decisões;
2- eliminar alguma tarefa com base em heurísticas ou preferências do *scheduler*. Algumas possibilidades são:

- o *scheduler* decide qual tarefa deve ser atrasada. Ele pode ter uma lista de preferências que especifica quais tarefas devem ser programadas primeiro em caso de ocorrência deste tipo de conflito entre elas;
- atrasar a tarefa menos crítica. Esta estratégia se baseia na mesma premissa levantada por Keng [Keng-88] e Sadeh [Sadeh-89], que as tarefas mais críticas devem ser programadas primeiro para evitar a perda dos seus *deadlines*;
- atrasar a tarefa que possui maior EBT e, em caso de empate, a menos crítica. Esta estratégia visa diminuir a perda de recursos compartilhados, que resultaria no aumento do *makespan* [Leon-92];
- outras, de acordo com o conhecimento adquirido na operação da planta.

A estratégia adotada aqui será a de atrasar a tarefa menos crítica, sendo que a criticalidade de uma tarefa será calculada por

$$crt(i) = \sum_{1 \le j \le M} \left(\frac{TP_{ij}}{DW_{ij}} * \max_{1 \le r \le R}\left(\frac{Q_{ij}^{r}}{q^{r}}\right)\right). \qquad (4.2)$$

A criticalidade calculada desta forma leva em consideração o tempo disponível para processar a tarefa e os recursos necessários para isto. Assim, para duas operações com a mesma relação (TP/DW), será considerada mais crítica aquela que necessitar de mais recursos para a sua realização. Da mesma forma, duas tarefas com a mesma relação (Q_{ij}/q) será mais crítica aquela que tiver menor janela de tempo disponível para sua alocação.

É bom salientar que a eliminação de tarefas do conjunto **CAN** não significa que estas tarefas não serão mais realizadas, mas sim, que a programação delas será feita em um nível mais abaixo do que o nível presente, ou seja, a programação das tarefas candidatas eliminadas do nó *i* será reconsiderada nos seus descendentes.

Exemplo:

Suponha que no nó i existam 4 tarefas candidatas: A, B, C e D. Se for determinado pela regra R3:C5 que B deve preceder AvCvD, a única certeza que se tem é que todas as seqüências terminadas em B estão proibidas, como mostra a Figura 4.2. Esta informação não provoca uma redução imediata do espaço de busca do problema, porque ela não permite descartar nenhuma das candidatas. No entanto, uma redução significativa poderá ser obtida pela utilização de uma das heurísticas propostas acima para atrasar uma das tarefas, por exemplo, atrasando a tarefa menos crítica C, como mostra a Figura 4.3.

Nota-se, pela Figura 4.3, que a utilização desta heurística elimina ramos da árvore que a princípio conduziriam a soluções factíveis. Entretanto as árvores das figuras 4.2 e 4.3 não representam a árvore completa de busca, a menos que a partir do nó i só faltassem as tarefas A, B, C e D para serem programadas. O caso geral, no entanto, é de que faltam mais tarefas para serem programadas (algumas que acabaram de serem eliminadas de CAN pelas regras R1, R2 e R3, e outras que em *i* ainda não estavam prontas para o seqüenciamento).

Assim, se C, que é a tarefa menos crítica, não é retirada do conjunto de candidatas, então A, B e D, que são tarefas mais críticas, terão que ser programadas depois de C (aumentando as suas criticalidades) e ainda poderão sofrer a concorrência de outras tarefas, o que contribuirá para que os seus DD possam ser violados. Portanto, a estratégia de atrasar a tarefa menos crítica quando se encontra o conflito C5 parece ser bastante razoável, o que será verificado nos resultados apresentados no capítulo 5.

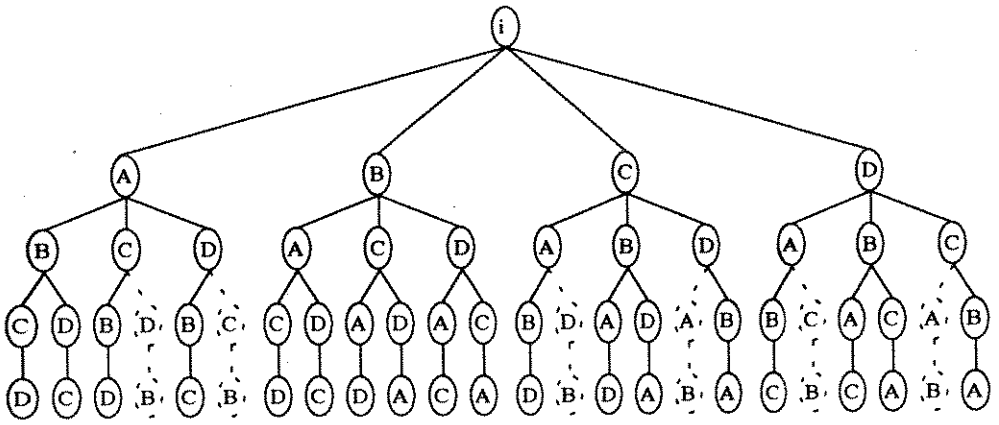


Figura 4.2- B precede AvCvD. As seqüências proibidas estão pontilhadas.

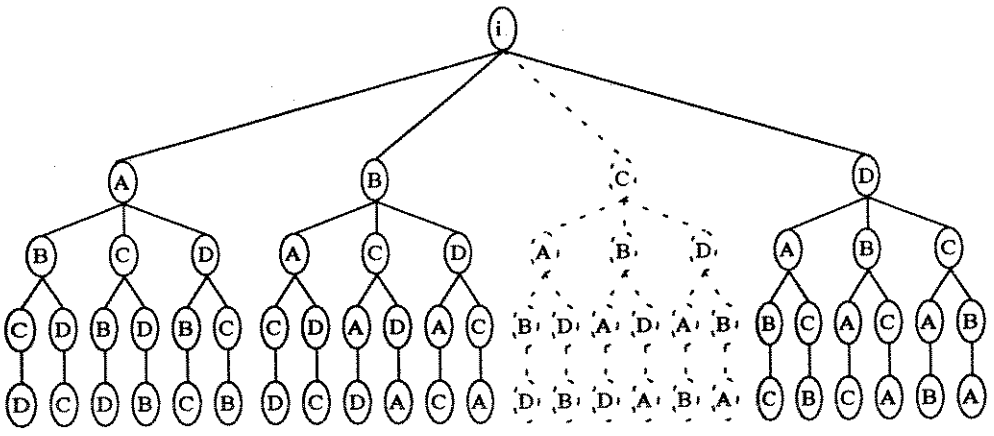


Figura 4.3- C é a tarefa menos crítica e portanto será atrasada.

■

O Algoritmo de Análise de Restrições:

Na técnica proposta, a análise para eliminar tarefas candidatas é feita em duas etapas:

A- Testando a factiblidade de cada par de tarefas, que poderá resultar nas conclusões C1, C2 e C3;

B- Testando a factibilidade de cada tarefa com a tarefa fictícia representando o conjunto das tarefas restantes, que poderá resultar nas conclusões C1, C2, C4 e C5; e, em caso da ocorrência de C5, utilizando heurísticas para resolver conflitos entre tarefas.

As figuras 4.4 e 4.5 mostram os diagramas de bloco para estas duas fases.

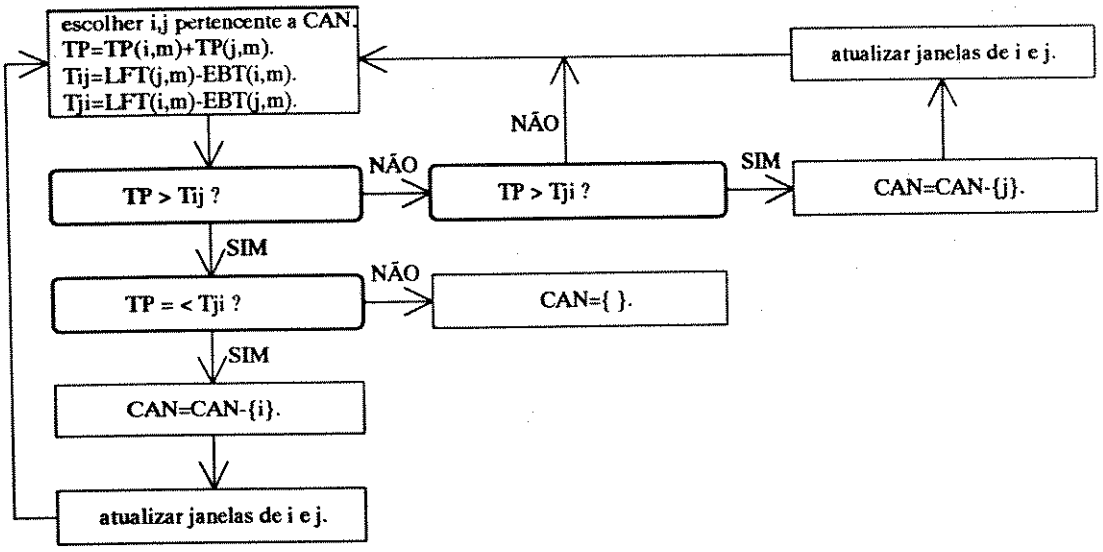


Figura 4.4- Determinação de CAN: Fase A - Análise por par de operações.

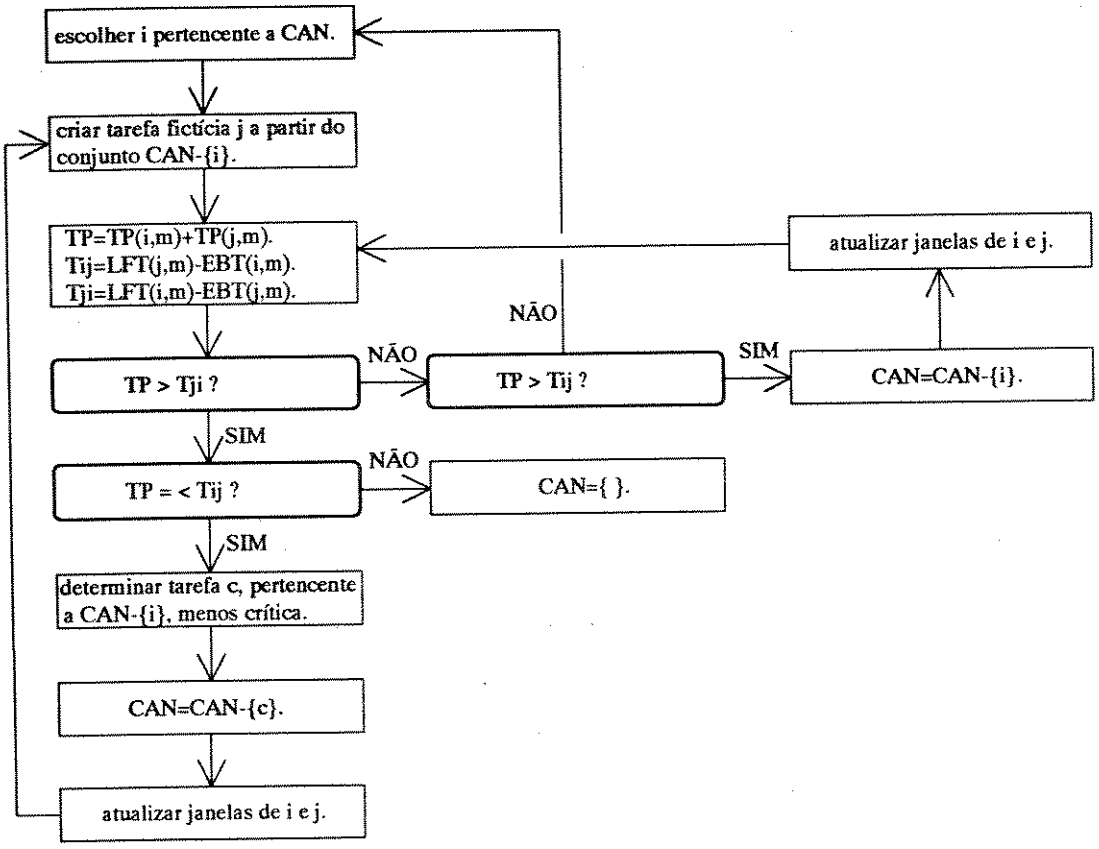


Figura 4.5- Determinação de CAN: Fase B - Análise por conjunto de operações.

4.3.3.2- Resolução de Conflitos no Tempo Originados pelos Restrições sobre os Recursos Compartilhados

A análise de restrições pode permitir que uma tarefa que infactibiliza outra seja candidata, isto porque ele não considera diretamente o consumo de recursos compartilhados (o faz, indiretamente, na propagação de restrições). A Figura 4.6 mostra duas tarefas em dois processadores que compartilham de um recurso comum r, cuja disponibilidade é 10 ur/ut. Os tempos de processamento (tp) das operações e seu consumo do recurso r (cr) são mostrados na Figura 4.6. Por simplicidade, o perfil de consumo do recurso compartilhado não é mostrado. A análise de restrições não é capaz de identificar qualquer relação de seqüenciamento entre estas duas tarefas e, portanto, A e B são igualmente candidatas. No entanto ao se alocar B antes de A e fazer a propagação das restrições resultantes desta alocação chega-se a conclusão de que B infactibiliza A, como mostra a Figura 4.7.

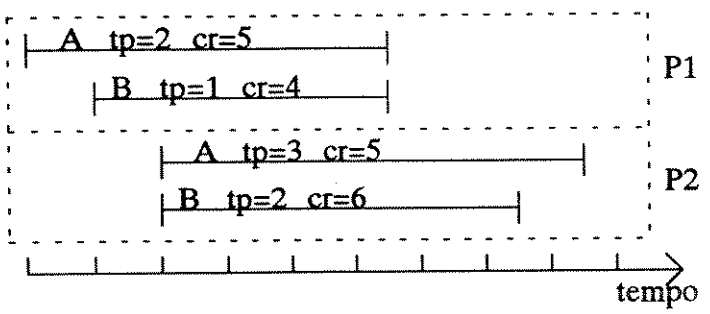


Figura 4.6- Tarefas candidatas A e B.

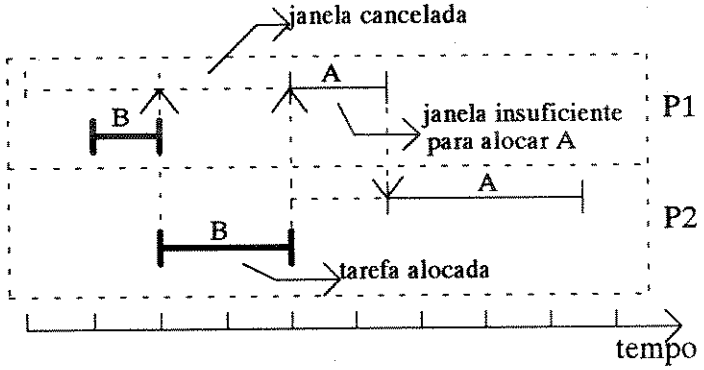


Figura 4.7- Violação do DD de A pela alocação de B.

O que se propõe então é que, após a análise de restrições temporais que determina o conjunto de candidatas a fazer parte da seqüência parcial que se está pesquisando, a explosão do nó seja simulada para verificar se, devido a limitação de recursos, alguma tarefa $t \in CAN$ infactibiliza alguma tarefa ainda não seqüenciada. Caso isto aconteça, a tarefa causadora da infactibilidade é eliminada de CAN.

A Figura 4.8 mostra o diagrama de blocos para a simulação.

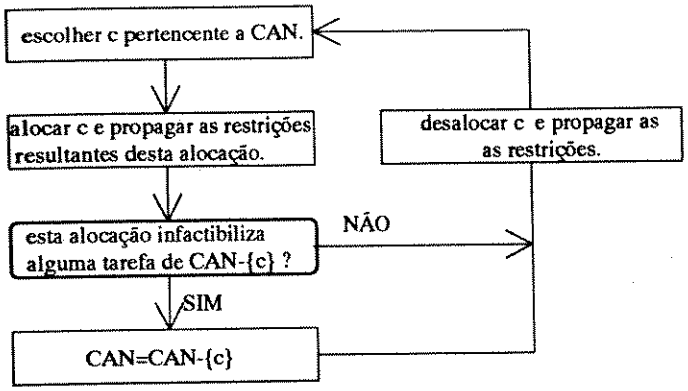


Figura 4.8- Determinação de CAN: Simulação.

Esta simulação poderia fazer parte da análise de restrições, da seguinte forma: sempre que se fosse testar a factibilidade de uma tarefa t com uma outra tarefa ou conjunto de tarefas, antes seria feita a alocação de t e a propagação das restrições resultantes desta alocação. Após a análise de restrições, t seria desalocada, as restrições propagadas, e o processo se repetiria para as demais tarefas que fossem testadas. No entanto, como a alocação e propagação de restrições é um processo bastante trabalhoso (na verdade representa grande parte do esforço computacional do algoritmo), é preferível primeiro tentar reduzir o conjunto de tarefas candidatas considerando apenas as restrições temporais e só depois verificar se os recursos podem causar alguma infactibilidade.

4.3.4- O Algoritmo Completo

A Figura 4.9 mostra o algoritmo BAB com Busca em Profundidade e Análise de Restrições (BAB-BPAR) mostrando os seus componentes principais: a busca-em-profundidade pela solução ótima; a determinação do conjunto de tarefas candidatas a fazer parte da seqüência parcial do nó que se está analisando; e a atualização das janelas das operações durante o processo de busca.

Observações relativas à busca-em-profundidade:

1- a partir do momento que se escolhe um ramo da árvore, ele é explorado até se chegar a uma solução completa ou a um *dead-end*. Portanto, como uma solução é construída passo-a-passo, a atualização do perfil de consumo dos recursos compartilhados (que é um processo que demanda muito tempo) fica facilitado, pois a cada passo tem-se somente que adicionar a este perfil o consumo de recursos por parte das operações da última tarefa que foi alocada (quando o algoritmo prossegue normalmente), ou retirar os consumos de recursos da última tarefa da seqüência (quando da ocorrência de *backtrack*). No caso da busca-pelo-melhor, como pode-se estar mudando de ramo a todo momento, várias vezes o perfil precisa ser quase que totalmente reconstruído acarretando um aumento significativo de tempo de simulação;

2- ocorre o *backtrack* quando um ramo é totalmente explorado ou quando se chega a um *dead-end*. Estando no nível n de um ramo da árvore, o *backtrack* consiste em voltar ao nível n-1, escolher o nó aberto com menor valor de LB e prosseguir na busca. Caso não exista nenhum nó aberto no nível n-1, volta-se ao nível n-2, e assim sucessivamente.

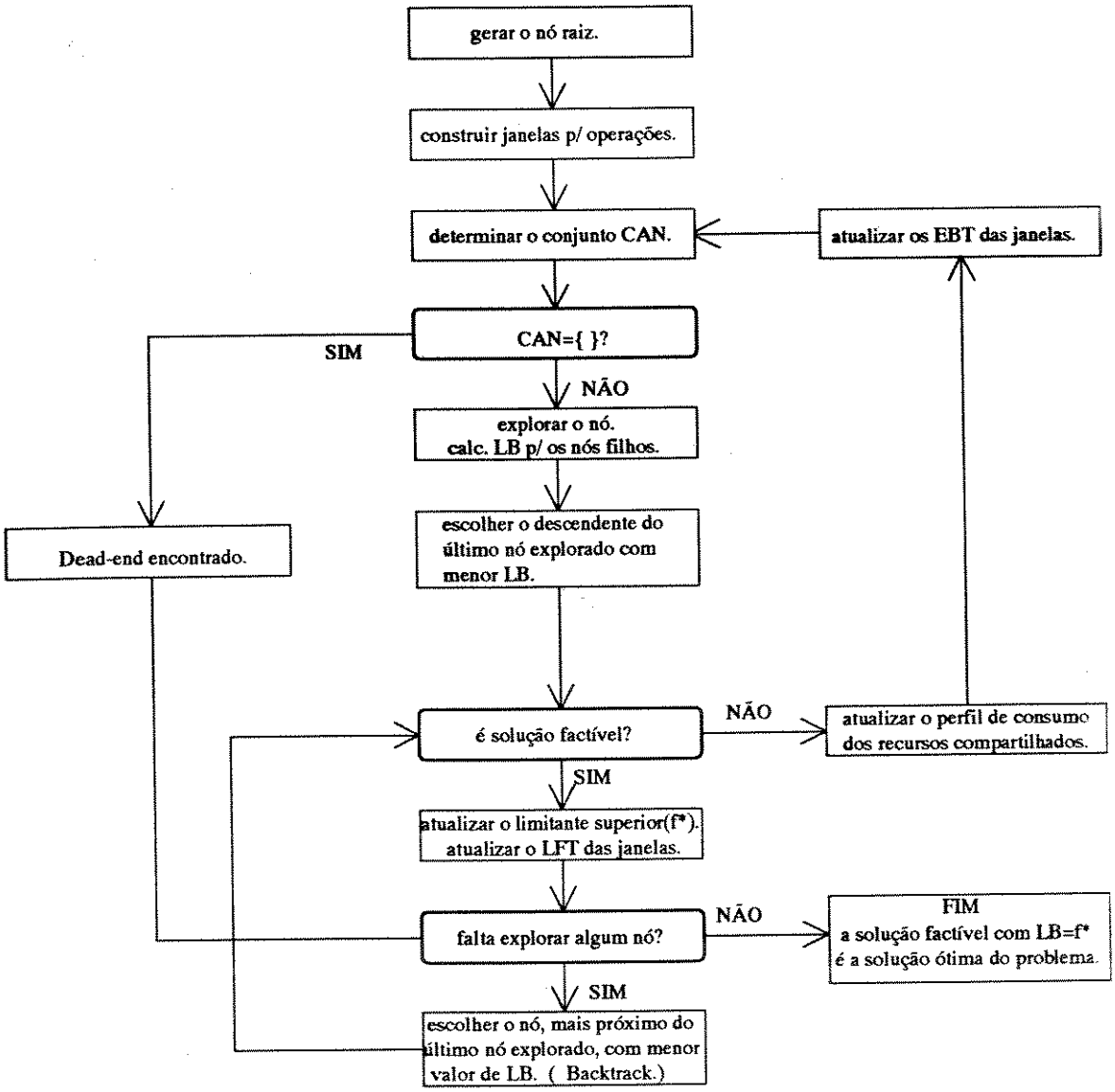


Figura 4.9- O Algoritmo BAB-BPAR

4.3.5- Observações Finais.

A escolha do conjunto de tarefas candidatas pode ser feito levando em consideração apenas a factibilidade da solução parcial que esta sendo gerada, independente do critério de otimização utilizado. No entanto, com um critério de otimização diferente do *makespan*, o algoritmo perde a possibilidade de limitar internamente o prazo de entrega das tarefas internas através do UB atualizado durante o processo de busca (perdendo eficiência), e a análise de restrições é feita verificando apenas se as tarefas

ainda não seqüenciadas violam o DD de alguma tarefa externa. O algoritmo, entretanto, torna-se mais geral, podendo ser aplicado ao seqüenciamento de *flow-shops* com restrições sobre recursos compartilhados com qualquer outro critério de otimização diferente do *makespan*.

Algo semelhante acontece quando se usa a busca-pelo-melhor em lugar da busca-em-profundidade, pois na primeira não se tem a garantia de encontrar uma solução factível rapidamente e, enquanto isto não acontece, não é possível limitar o DD das tarefas internas. Conseqüentemente a factibilidade destas tarefas não serão afetadas, mas elas ainda poderão infactibilizar algumas tarefas externas.

A utilização de heurísticas no caso C5 é opcional. Se o *scheduler* não desejar utilizar heurísticas ou considerar que esta estratégia não é conveniente para o seu caso particular, o algoritmo poderá ser utilizado normalmente com as conclusões C1, C2, C3 e C4 e a simulação. Também, se o *scheduler* desejar ou achar conveniente, a heurística poderá ser utilizada durante todo o algoritmo independente da existência de conflitos. Isto no entanto poderá levar o algoritmo a encontrar soluções sub-ótimas, como acontece com as propostas de Keng e Sycara [Keng-88, Sycara-92]. Uma grande vantagem do BPAR é exatamente o fato de poder abdicar de heurísticas durante o processo de busca.

4.5- Conclusões.

Propôs-se um algoritmo, denominado BPAR (busca em profundidade com análise de restrições), que procura a solução ótima do problema de *flow-shop* com alocações externas e escassez de recursos compartilhados utilizando um algoritmo de busca do tipo BAB. A escassez de recursos é utilizada para guiar a busca pela solução ótima através da minimização do *makespan* e para, juntamente com as alocações externas, reduzir o espaço de busca do problema.

O algoritmo utiliza técnicas clássicas da PO como a própria estrutura do BAB e a análise de restrições temporais, bem como técnicas de IA como a propagação de restrições e a utilização de heurísticas para tomar decisões oportunísticas de seqüenciamento. Com isto procura-se satisfazer ao mesmo tempo os objetivos do problema e suas restrições.

O algoritmo é, portanto, uma alternativa para as técnicas clássicas da PO, que têm por objetivo somente otimizar o desempenho da planta (segundo um ou vários critérios de desempenho) e para as técnicas modernas de IA, que têm por objetivo principal satisfazer as restrições do problema sem se preocupar com a otimização de um critério de desempenho específico.

Utiliza-se a criticalidade de tarefas como heurística para reduzir o espaço de busca do problema. No entanto o BPAR está aberto para receber novas heurísticas que o *scheduler* considere relevantes, ou para ser utilizado sem qualquer regra heurística.

Capítulo 5

Resultados de Simulações

5.1- Introdução.

Neste capítulo são propostos 12 problemas para serem resolvidos pelo BPAR. O objetivo não é fazer uma análise estatística do desempenho do algoritmo, mas mostrar o seu desempenho em diferentes situações. A seção 5.1.1 mostra os problemas propostos com as respectivas soluções ótimas encontradas pelo algoritmo *Branch-and-Bound* com busca-pelo-melhor (BAB-BPM), utilizando funções C1 e C2 para estimar o *makespan*.

Na seção 5.2 apresentam-se os resultados obtidos com alocação externa de tarefas. Compara-se o desempenho do algoritmo BPAR em duas versões, utilizando os algoritmos C1 e C2, com um algoritmo que não verifica, na expansão de cada nó, a possível infactibilização de tarefas. Mostra-se a freqüência com que a heurística proposta para eliminar as tarefas menos críticas é acionada. Um dos problemas é analisado em detalhe para mostrar como se realiza a busca com análise de restrições.

A seção 5.3 mostra a aplicação do BPAR na situação em que não existem tarefas alocadas externamente, sendo todas elas definidas internamente assim que um limitante superior das soluções ótimas é obtido pelo BAB. A comparação é feita com os algoritmos BAB(C1) e BAB(C2).

Finalmente, na seção 5.4, apresenta-se a utilização do algoritmo BPAR na situação em que existem janelas proibidas pré-definidas, o caso típico sendo paradas de processadores para manutenção, que podem ser definidas *a priori* na forma de alocações fixas.

5.1.1- Problemas Propostos

Para mostrar a utilização do algoritmo BPAR será resolvido um conjunto de problemas nas seções abaixo, os quais são numerados de 1 a 12 e cujos dados encontram-se no Anexo A. Os tempos de processamento (em unidades de tempo - ut) e os consumos de recurso (em unidades de recurso/unidade de tempo - ur/ut) de cada operação foram gerados aleatoriamente dentro do domínio especificado para cada um deles. Em todos os problemas se utiliza um único recurso compartilhado pois a utilização de mais recursos implicaria no aumento da dimensão do problema, em termos do número de nós sondados e do tempo de cálculo, não modificando as conclusões acerca do desempenho do algoritmo. O tratamento de vários recursos compartilhados é abordado em [Passos-93].

As dimensões dos problemas e os domínios de TP e Q são:

-P1 a P10: N = 8, M=6, R=1, q = 10 ur/ut, $TP_{ij} \in [1,20]$ e $Q_{ij} \in [1,10]$;
-P11: N = 8, M = 3, R = 1, q = 10 ur/ut, $TP_{ij} \in [1,10]$ e $Q_{ij} \in [1,10]$;
-P12: N = 10, M = 4, R = 1, q = 10 ur/ut, $TP_{ij} \in [1,20]$ e $Q_{ij} \in [1,10]$.

Estes problemas não possuem alocações externas e foram resolvidos inicialmente pelo BAB-BPM. A Tabela 5.1 mostra uma solução ótima (podendo existir várias outras, dado que existe um grande número de soluções factíveis (N!)) para cada um destes problemas. A tabela 5.2 mostra o número de nós sondados (NS), o número de soluções completas encontradas (SC) e o tempo total de CPU (TT) gasto, em segundos, pelos algoritmos C1 e C2. As soluções ótimas obtidas aqui serão utilizadas nas próximas seções para determinar as alocações externas de algumas tarefas. de forma que, quando o problema resultante for resolvido pelo BPAR, as soluções possam ser comparadas e o BPAR validado.

Tabela 5.1: Solução Ótima para os Problemas Propostos

P	Seqüência ótima	*Makespan*
1	10732645	334
2	35206741	329
3	06215734	363
4	64071253	326
5	47613205	277
6	47103265	322
7	24531067	391
8	72650341	266
9	37054612	257
10	16537402	253
11	41723056	76
12	6480591327	278

Tabela 5.2- Resultados: BAB-BPM(C1) x BAB-BPM(C2)

	BAB-BPM(C1)			BAB-BPM(C2)		
P	NS	SC	TT	NS	SC	TT
1	17540	2263	195.20	13419	518	169.10
2	19450	3036	197.91	14904	779	161.41
3	16691	1993	149.22	11861	485	117.58
4	15455	838	178.86	13274	324	173.70
5	14709	2406	117.72	10438	75	184.33
6	20471	2367	277.89	17153	1055	245.5
7	42590	14845	788.93	27394	2474	469.90
8	9274	462	71.28	7918	249	69.92
9	9466	257	89.57	7387	51	87.95
10	10740	253	119.23	8498	129	102.98
11	4215	258	16.59	2516	1	39.62
12	246397	11704	1328.80	164746	1140	1166.65

Comentários a respeito das respostas obtidas com o algoritmo BAB-BPM:

- de acordo com a seção 3.3.4, para estimar o LB de um nó o algoritmo C2 faz os mesmos cálculos que C1 mais os cálculos necessários para prever a perda mínima de recursos na Região II (*lookahead*). Portanto, para sondar um mesmo nó, C2 consome mais tempo do que C1. Este consumo a mais de tempo só é compensado se o número de nós sondados por C2 é bem menor do que o número de nós sondados por C1. Da Tabela 5.2 nota-se que C2 só apresenta uma redução significativa de TT em

relação a C1 nos casos em que NS(C2) é muito menor do que NS(C1), p. ex. em P1, P2, P3, P7, P10 e P12.

- os resultados apresentados para P12 foram obtidos utilizando a busca-em-profundidade (BP) em lugar da busca-pelo-melhor. A BPM não foi capaz de encontrar a solução ótima deste problema, porque neste tipo de busca o número de nós gerados pelo BAB que ficam esperando para serem sondados (nós abertos) cresce exponencialmente com o tempo. Como o espaço de busca do problema é muito grande (10!), o número de nós abertos cresce muito ultrapassando a capacidade computacional disponível. Na BP o número de nós abertos cresce linearmente com o tempo, tornando o problema mais tratável, embora geralmente o número de nós sondados seja maior do que na BPM [Korf-85, Pearl-84].

- o número de soluções completas pesquisadas é apresentado nas simulações realizadas para mostrar que o *lookahead* realizado pelo algoritmo C2 consegue dirigir bem melhor a busca pela solução ótima, se comparado com C1. Pode-se notar da Tabela 5.2 que de C1 para C2 a redução do número de soluções completas encontradas é bem maior que a redução do número de nós sondados (em valores relativos);

- todos os programas foram escritos em linguagem C e executados em uma estação-de-trabalho do tipo SUN-SPARC 2.

5.2- *Flow-shop* com Alocações Externas.

No capítulo 4 foi proposto um algoritmo (denominado BPAR) capaz de seqüenciar e alocar tarefas em um *flow-shop*, considerando que existem restrições na oferta de recursos compartilhados, bem como limitações sobre os intervalos de tempo em que algumas tarefas devem rigorosamente ser executadas. Para ilustrar a eficiência do algoritmo, ou seja, a redução da árvore de busca que pode ser obtida através da incorporação de testes de factibilidade sobre as janelas de demanda das operações e sobre os recursos compartilhados, o conjunto de problemas da Tabela 5.1 é resolvido com algumas tarefas alocadas externamente. As janelas destas tarefas foram escolhidas de forma a utilizar as soluções ótimas determinadas na seção 5.1.1.

As soluções obtidas pelo BPAR são comparadas com as soluções obtidas por uma abordagem denominada Solução Simplificada (SS). Esta solução simplificada consiste em utilizar o BAB sem verificar *a priori* a possibilidade de ocorrência de infactibilidades, ou seja:
i) o conjunto CAN é determinado com base apenas no RT das tarefas ainda não seqüenciadas e, portanto, farão parte deste conjunto todas as tarefas do conjunto U e as tarefas de J que possuem o seu RT dentro da Região II;

ii) uma infactibilidade só é detectada quando o tempo de conclusão de uma operação em um determinado processador for maior que o último instante de início (LBT) de uma outra operação que utiliza o mesmo processador.

A Tabela 5.3 mostra os resultados obtidos, primeiro, pela solução simplificada (SS) proposta acima e, segundo, pelo BPAR. A estimativa dos LB's dos nós é feita pelos algoritmos C1 e C2.

Tabela 5.3- Resultados: SS(C1 e C2) x BPAR(C1 e C2)

P	Alocações Externas tarefa: rt-dl	SS(C1)	BPAR(C1)	SS(C2)	BPAR(C2)
1	3: 50-250 5: 0-200 6: 0-200 7: 50-250	NS=3903 SC=375 TT=14.80	NS=536 SC=17 TT=4.82 H=112	NS=3364 SC=70 TT=15.65	NS=506 SC=11 TT=5.3 H=103
2	2: 0-200 3: 0-200 4: 50-300 6: 50-300	NS=3577 SC=170 TT=13.16	NS=283 SC=4 TT=3.54 H=45	NS=2942 SC=80 TT=14.72	NS=276 SC=3 TT=3.91 H=45
3	0: 50-300 1: 150-350 2: 50-300 5: 200-400	NS=2160 SC=218 TT=9.29	NS=367 SC=14 TT=4.77 H=18	NS=1783 SC=116 TT=10.67	NS=363 SC=13 TT=5.43 H=17
4	0: 0-250 1: 50-300 5: 100-350 7: 50-250	NS=4327 SC=117 TT=18.11	NS=514 SC=5 TT=7.78 H=76	NS=3891 SC=54 TT=20.78	NS=513 SC=4 TT=8.72 H=76
5	1:50-250 2: 100-300 4: 0-200 7: 0-200	NS=4165 SC=173 TT=14.60	NS=388 SC=12 TT=4.63 H=39	NS=3604 SC=27 TT=16.75	NS=384 SC=8 TT=5.22 H=39
6	1:0-200 2:50-300 4:0-200 5:50-350	NS=5779 SC=627 TT=23.19	NS=756 SC=7 TT=10.17 H=138	NS=4711 SC=258 TT=25.20	NS=753 SC=7 TT=11.31 H=134
7	4:200-400 5:100-300 6:0-250 7:0-300	NS=7790 SC=1761 TT=28.10	NS=1149 SC=12 TT=12.12 H=141	NS=5582 SC=198 TT=26.40	NS=1138 SC=9 TT=13.48 H=141
8	0:50-250 1:50-300 3:100-300 4:50-300	NS=1977 SC=92 TT=9.63	NS=433 SC=5 TT=6.91 H=65	NS=1719 SC=71 TT=11.14	NS=427 SC=5 TT=7.65 H=65
9	0:0-200 1:50-250 2:100-300 3:0-200	NS=4144 SC=133 TT=16.39	NS=442 SC=8 TT=6.50 H=99	NS=3271 SC=34 TT=18.40	NS=441 SC=8 TT=6.81 H=99
10	0:50-300 3:50-250 4:100-300 5:0-250	NS=2388 SC=51 TT=11.84	NS=502 SC=6 TT=8.25 H=86	NS=1987 SC=27 TT=13.13	NS=550 SC=9 TT=9.49 H=76
11	0: 30-60 2: 0-50 4: 0-40 5: 50-80 7: 10-50	NS=287 SC=23 TT=0.48	NS=42 SC=4 TT=0.19 H=16	NS=232 SC=2 TT=0.56	NS=40 SC=2 TT=0.19 H=15
12	0: 50-250 1: 50-250 2: 100-300 3: 100-300 4: 0-200 5: 0-250 6: 0-250 7: 100-300 8: 0-250 9: 50-250	NS=22878 SC=223 TT=93.48	NS=2005 SC=8 TT=27.27 H=1216	NS=17747 SC=26 TT=99.20	NS=1972 SC=6 TT=30.45 H=1164

A Tabela 5.4 mostra os resultados obtidos pelo BPAR(C1) para o problema 1 com 10 conjuntos diferentes para as janelas das tarefas externas. Nestas tabelas, H é o número de vezes que a heurística baseada na criticalidade foi utilizada. Para todos os conjuntos de tarefas externas obteve-se a solução ótima do problema.

Tabela 5.4- Solução de P1 com várias janelas

P	Alocações Externas tarefa: rt-dl	BPAR(C1)	P	Alocações Externas tarefa: rt-dl	BPAR(C1)
1.1	3:50-250 5:0-200 6:0-200 7:50-250	NS=536 SC=17 TT=4.84 H=112	1.6	1:150-350 3:50-300 5:50-300 7:50-300	NS=860 SC=8 TT=13.92 H=197
1.2	1:100-350 2:0-200 4:50-250 7:100-300	NS=428 SC=9 TT=5.16 H=72	1.7	2:0-250 3:50-250 5:50-300 6:0-200	NS=709 SC=7 TT=7.86 H=119
1.3	0:150-300 4:50-250 5:0-250 6:0-200	NS=483 SC=8 TT=5.45 H=59	1.8	1:100-350 4:0-250 5:50-200 7:100-350	NS=376 SC=5 TT=5.84 H=62
1.4	1:200-350 2:0-150 3:100-250 6:0-200	NS=190 SC=8 TT=2.40 H=22	1.9	0:100-350 1:100-350 5:50-200 6:0-150	NS=282 SC=5 TT=3.90 H=43
1.5	0:200-350 2:0-150 3:100-300 7:100-300	NS=273 SC=5 TT=4.30 H=53	1.10	2:0-100 4:0-200 6:0-200 7:50-300	NS=139 SC=3 TT=1.30 H=27

Comentários:

- utilizando o algoritmo BPAR obtiveram-se as soluções ótimas para todos os problemas apesar do recurso à heurística proposta na seção 4.3.3.1 que elimina do conjunto **CAN** as tarefas menos críticas. Obviamente não existe garantia de que isto ocorrerá sempre. As duas tabelas mostram que a utilização da heurística é grande em relação ao número de nós sondados, o que se traduz em um ganho de tempo considerável. Por exemplo, para o problema 1 da Tabela 5.3, que corresponde ao problema 1.1 da Tabela 5.4, SS(C1) gerou 3903 nós gastando 14.8s enquanto que o BPAR(C1) gerou apenas 536 nós gastando 4.8s, sendo que a heurística foi responsável pela eliminação de 112 nós. Se a heurística não tivesse sido utilizada, o número de nós sondados pelo BPAR(C1) seria 536 nós + (112 nós + seus possíveis descendentes).

- das Tabelas 5.3 e 5.4 conclui-se que o BPAR consegue dirigir eficientemente a busca em direção à solução ótima, não importando qual o conjunto de tarefas externas é utilizado nem quantas vezes que a heurística é acionada, até porque isto só acontece em situações claras de conflito entre tarefas.

- BPAR(C1) x BPAR(C2): O número de nós sondados é praticamente o mesmo. Em tempo, C2 e C1 são praticamente semelhantes. Observa-se que, em SS, o número de nós sondados por C2 é bem inferior ao número de nós sondados por C1. Isto acontece porque na estimativa de C1 vários nós que ficavam abaixo do limitante superior da busca da busca e eram explorados, na estimativa de C2 ficavam acima e eram descartados. No entanto, no BPAR isto não acontece porque antes que um ramo da árvore atinja

um LB > UB para ser descartado, o *lookahead* (análise e propagação de restrições) consegue determinar que a seqüência em questão é infactível, eliminando o nó independente da melhor estimativa de LB feita por C2.

- BPAR(C1,C2) x SS(C1,C2): A diminuição do número de nós sondados e do tempo é altamente significativa, mostrando o ganho que se tem quando se faz a escolha de tarefas de forma a evitar a perda do prazo de entrega de outras (BPAR) em relação a quando se deixa para verificar a factibilidade das soluções *a posteriori* (SS).

5.2.1- Exemplo detalhado

Esta seção apresenta com detalhes a solução do problema P11 da Tabela 5.3 utilizando o BPAR(C1). Os tempos de processamento das operações e a quantidade de recurso compartilhado que cada uma delas consome estão especificados na Tabela 5.5.

TABELA 5.5- Tempos. de proc. e cons. de recurso

	tempo proc.			cons. de recurso		
	O. 1	O. 2	O. 3	O. 1	O. 2	O. 3
T. 0	3	7	2	6	3	4
T. 1	5	7	8	5	6	3
T. 2	8	5	6	4	6	2
T. 3	5	7	3	6	4	2
T. 4	10	4	6	9	2	5
T. 5	6	9	3	7	7	4
T. 6	9	2	5	3	6	3
T. 7	4	4	6	5	6	2

Na Figura 1 é mostrada a carta de Gantt da solução ótima, na Figura 2 o perfil de consumo do recurso compartilhado e na Figura 3 a árvore de solução do problema. Na carta de Gantt são mostradas, em linhas tracejadas, as janelas de demanda das tarefas externas. Observa-se que as janelas atribuídas para as tarefas externas deixavam uma folga grande para cada uma delas, o que se traduz em flexibilidade para o BAB fazer a alocação e seqüenciamento destas tarefas.

Na árvore de solução do problema, em cada nível, os nós que foram explodidos primeiro estão colocados mais à esquerda, e aqueles que foram explodidos por último mais à direita. Nota-se que, à medida que se caminha para o lado direito, o número de nós gerados reduz-se substancialmente pois o limitante superior da busca diminui enquanto que o limitante inferior dos nós aumenta, o que significa dizer que as janelas das operações estão cada vez menores aumentando a ocorrência de conflitos e conseqüentemente de nós descartados pela análise de restrições. Nota-se ainda que apenas os ramos 472305, 4172306 e 24 foram eliminados pelo critério do limitante superior (LB≥f*), todos os demais tendo sido eliminados pela análise de restrições.

A Tabela 5.6 mostra todos os nós que foram sondados pelo BPAR(C1). A primeira coluna mostra o nó analisado, a segunda o conjunto inicial de tarefas candidatas a fazer parte da seqüência parcial do nó, a terceira as tarefas que foram atrasadas pela análise de restrições, a quarta coluna as regras responsáveis pelo atraso destas tarefas, a quinta e a sexta coluna o limitante inferior e o limitante superior da busca, respectivamente. Por exemplo, o nó 47 possuía inicialmente o conjunto de tarefas candidatas {1,3,6,0,2}, das quais as tarefas 0 e 6 foram eliminadas pelas regras C3 e C5, respectivamente.

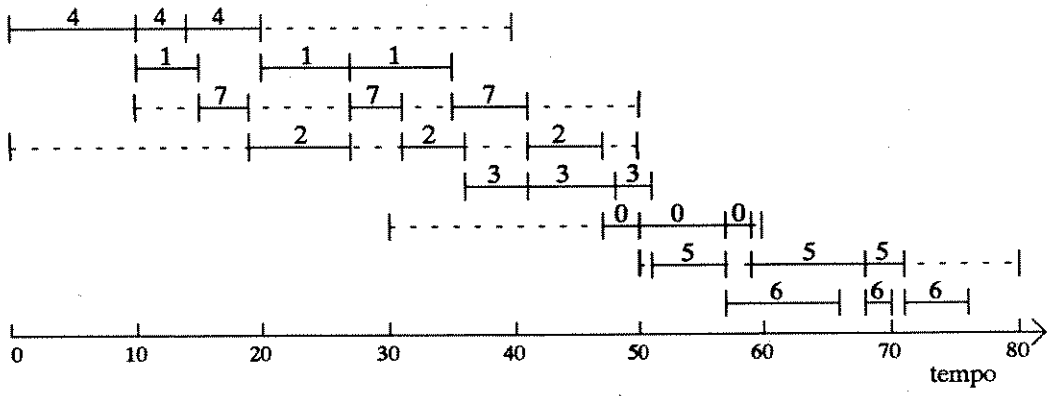


Figura 1- Carta de Gantt da seqüência ótima 41723056.

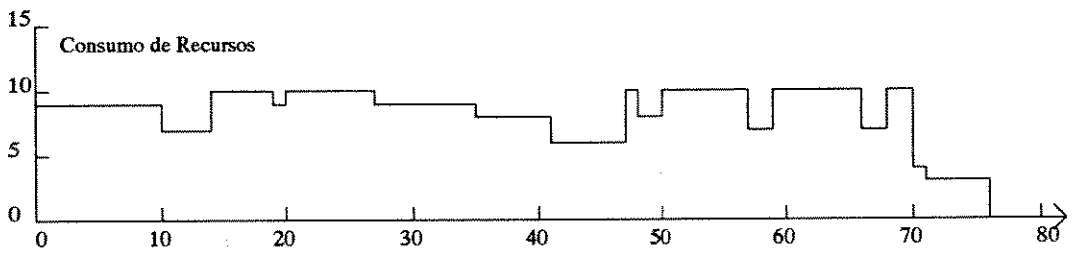


Figura 2- Perfil de consumo do recurso compartilhado.

Nota-se que, no ramo que conduz à solução ótima do problema, a heurística de atrasar a tarefa menos crítica (R3:C5) foi utilizada três vezes: no nó Raiz, com a eliminação da tarefa 6; no nó 4, com a eliminação da tarefa 6; e no nó 4172, novamente com a eliminação da tarefa 6. Ao todo, a heurística foi utilizada 16 vezes, provocando uma redução substancial dos nós gerados, sem comprometer a solução ótima do problema, mostrando mais uma vez que o uso de criticalidades em situação clara de conflito é bastante razoável.

Outra observação interessante é que a utilização da heurística geralmente ocorre nos níveis mais altos da árvore de busca (0, 1, 2, ...) porque nestes níveis existem poucas tarefas alocadas e portanto as janelas das operações ainda não seqüenciadas são grandes o suficiente para evitar o conflito entre duas operações, mas não para evitar o conflito de uma operação com um conjunto de operações. Nos níveis

mais profundos as janelas das operações tornam-se menores e os conflitos passam a ocorrer entre pares de tarefas, eliminando a necessidade de utilização da heurística.

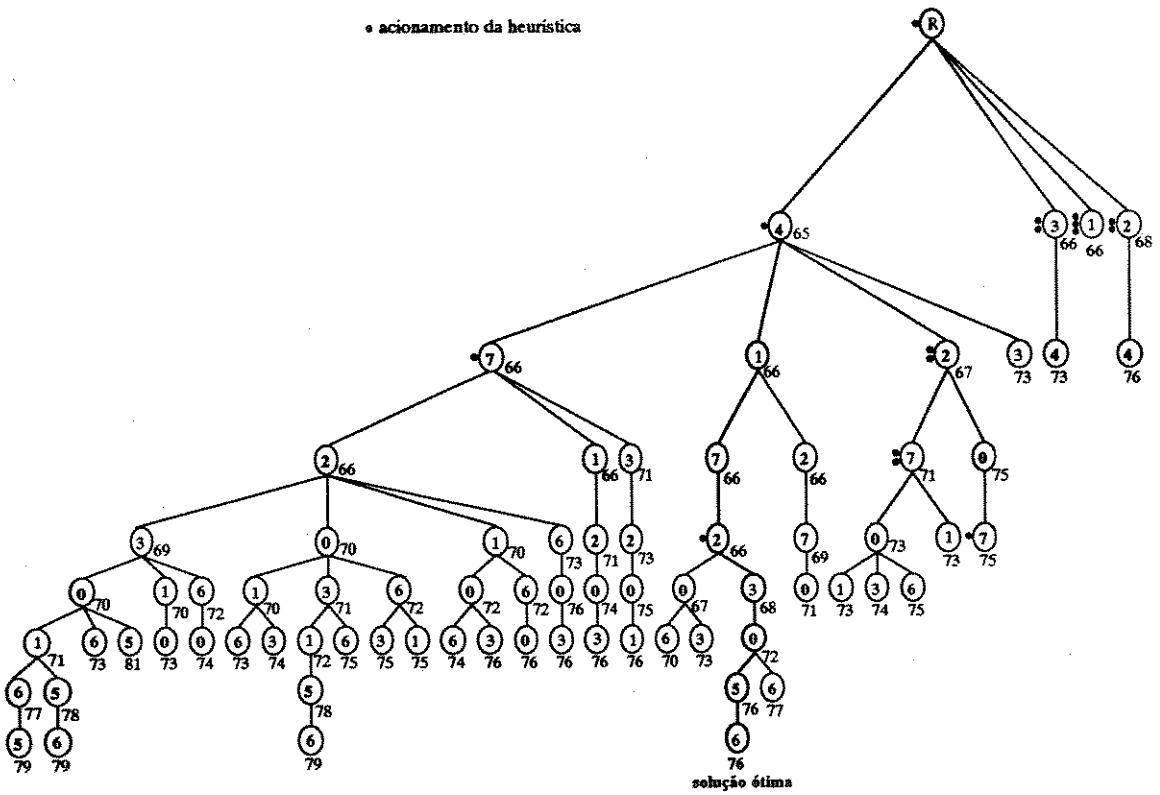


Figura 3- Árvore de solução do problema

Observa-se que, à medida que o limitante superior da busca (f*) diminui, há uma redução substancial do número de nós explodidos. No início da busca, quando f*=TF (somatória dos tempos de processamento de todas as operações), a árvore de solução do BPAR é muito semelhante àquela da solução simples, as diferenças aumentando com a diminuição do f*. Isto foi observado em todas as simulações realizadas neste capítulo.

Tabela 5.6- Nós Sondados pelo BPAR

NÓ (Seqüência)	CAN inicial	Tarefas atrasadas	Regras Aplicadas	LB	UB
Raiz	1,3,6,2,4	6	C5	65	134
4	1,3,6,2,7	6	C5	65	.
47	1,3,6,2,0	0,6	C3,C5	66	.
472	1,3,6,0	-	-	66	.
4723	1,6,0	-	-	69	.
47230	1,6,5	-	-	70	.
472301	6,5	-	-	71	.
4723016	5	-	-	77	.
47230165	-	-	-	79	79
4723015	6	-	-	78	.
47230156	-	-	-	79	79
472306	1,5	5,1	C3,S	73	.
472305	-	-	-	81	.
47231	6,0,5	6,5	C3,C3	70	.
472310	6,5	5,6	C3,S	73	.

Tabela 5.6- Continuação

NÓ (Seqüência)	CAN inicial	Tarefas atrasadas	Regras Aplicadas	LB	UB
47236	1,0,5	1,5	C3,C3	72	.
472360	1,5	5,1	C3,S	74	.
4720	1,3,6	-	-	70	.
47201	3,6,5	5	S	70	.
472016	3,5	3,5	S,S	73	.
472013	6,5	5,6	C3,S	74	.
47203	1,6,5	5	C3	71	.
472031	6,5	6	S	72	.
4720315	6	-	-	78	.
47203156	-	-	-	79	79
472036	1,5	5,1	C3,S	75	.
47206	1,3	-	-	72	.
472063	1,5	5,1	C3,S	75	.
472061	3,5	3,5	C2	75	.
4721	3,6,0	3	C3	70	.
47210	3,6,5	5	C3	72	.
472106	3,5	5,3	C3,S	74	.
472103	6,5	6,5	C2	76	.
47216	3,0,5	3,5	C3,C3	72	.
472160	3,5	5,3	C3,S	76	.
4726	1,3,0	1,3	C3,C3	73	.
47260	1,3,5	5,1	C3,S	76	.
472603	1,5	5,1	C3,S	76	.
471	3,6,0,2	3,6,0	C3,C3,C3	66	.
4712	3,6,0	3,6	C3,S	71	.
47120	3,6,5	5,6	C3,S	74	.
471203	6,5	6,4	C2	76	.
473	1,6,0,2	1,6,0	C3,C3,C3	71	.
4732	1,6,0	1,6	C3,S	73	.
47320	1,6,5	5,6	C3,S	75	.
473201	6,5	6,5	C2	76	.
41	3,6,0,2,7	3,6,0	C3,C3,C3	66	.
417	3,6,0,2	3,6,0	C3,C3,C3	66	.
4172	3,6,0	6	C5	66	.
41720	3,6	-	-	67	.
417206	3,5	3,5	S,S	70	.
417203	6,5	5,6	C3,S	73	.
41723	6,0,5	5,6	C3,C3	68	.
417230	6,5	-	-	72	.
4172305	6	-	-	76	.
41723056	-	-	-	76	76
4172306	-	-	-	77	.
412	3,6,0,7	3,6,0	C3,C3,C3	66	.
4127	3,6,0	3,6	C3,C3	69	.
41270	3,6,5	5,3,6	C3,S,S	71	.
42	1,3,6,0,7	1,3,6	C3,C5,C5	67	.
427	1,3,6,0	3,6	C5,C5	71	.
4270	1,3,6	-	-	73	.
42701	3,6,5	3,6,5	C4,S,S	73	.
42703	1,6,5	1,6,5	C3,C3,S	74	.
42706	1,3,5	3,5,1	C3,C3,S	75	.
4271	3,6,0,5	3,5,0,6	C3,C3,S,S	73	.
420	1,3,6,7	1,3,6	C3,C3,C3	75	.
4207	1,3,6,5	5,6,1,3	C3,C5,S,S	75	.
43	1,6,0,2,7	1,6,0,2,7	C3,C3,S,S	73	.
3	1,6,2,4,7	1,6,2,7	C5,C5,S,S	66	.
34	1,6,0,2,7	1,6,0,2,7	C3,C3,C3,S,S	73	.
1	3,6,2,4,7	3,6,2,4,7	C5,C5,C4,S,S	66	.
2	1,3,6,4,7	1,3,6,7	C5,C5,C5,S	68	.
24	-	-	-	76	.

As Figuras 5.4 a 5.11 mostram os estados de alguns nós quando estes são explorados. Estas figuras mostram as operações alocadas e as janelas das operações ainda não alocadas nos processadores

0, 1 e 2. Do lado direito dos processadores está colocado o número da tarefa (N), o tempo de processamento (TP) e o consumo de recurso (CR) de cada operação no formato (N,TP,CR). Abaixo dos processadores é mostrado o perfil de consumo do recurso compartilhado.

A Figura 5.4 mostra o estado do nó raiz (R), para o qual UB=TF=134 (seção 4.3.2). O conjunto de candidatas iniciais é composto das tarefas 1, 2, 3, 4 e 6 (as tarefas 0, 5 e 7 não são candidatas porque, como elas têm RT>>0, qualquer seqüência que começasse com uma destas tarefas estaria deixando de utilizar uma grande quantidade de recursos disponíveis e conseqüentemente a solução resultante seria ruim). No processador 1, fazendo i={4}, j={1,2,3,6} e aplicando a regra R3 chega-se a conclusão que a tarefa 4 deve ser programada antes de 1v2v3v6. Como 6 é a tarefa menos crítica, ela é atrasada.

A Figura 5.5 mostra o estado do nó 47230. Como o prazo de entrega das tarefas internas ainda é 134, as três tarefas ainda não seqüenciadas (1,5,6) têm tempo suficiente para serem alocadas. A Figura 5.6 mostra o estado do nó 472306. Ao atingir este nó o limitante superior da busca já caiu de 134 para 79, o mesmo acontecendo com o prazo de entrega das tarefas internas. A análise de restrições permite então concluir que as tarefas 1 e 5 não podem mais ser alocadas sem provocar a infactibilização da solução e este nó é descartado.

As Figuras 5.7 e 5.8 mostram os nós 417230 e 41723056, respectivamente, sendo este último a solução ótima do problema. Após a passagem por este ramo o limitante superior da busca cai para 76, diminuindo consideravelmente o espaço de soluções que ainda não havia sido pesquisado.

A Figura 5.9 mostra o estado do nó 43. O prazo de entrega das tarefas internas caiu para 76 provocando uma diminuição substancial das janelas destas tarefas e aumentando a competição pelo uso do recurso compartilhado e do tempo. A análise de restrições permite concluir que este nó é infactível.

A Figura 5.10 mostra o estado do nó 1. O nó 1 já é um nó infactível, pois todas as tarefas candidatas a fazer parte da seqüência foram eliminadas pela análise de restrições. Veja por exemplo a Figura 5.11 em que a tarefa 3 foi alocada. Após a propagação de restrições verifica-se que a janela de tempo da tarefa 4 no processador 2 foi cancelada. Olhando apenas para as janelas de tempo das tarefas 3 e 4 não é possível detectar esta infactibilidade, pois ela é causada pela escassez do recurso compartilhado. Este é um caso em que a simulação se faz necessária. O nó 13 da Figura 5.11 não é gerado pela busca e foi mostrado aqui para ilustrar a necessidade da simulação para detetar conflitos pela utilização de um recurso compartilhado.

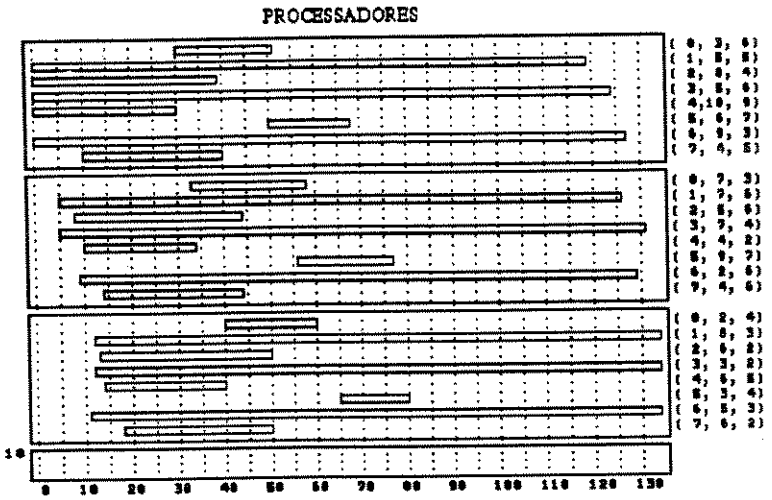


Figura 5.4- Estado do nó raiz.

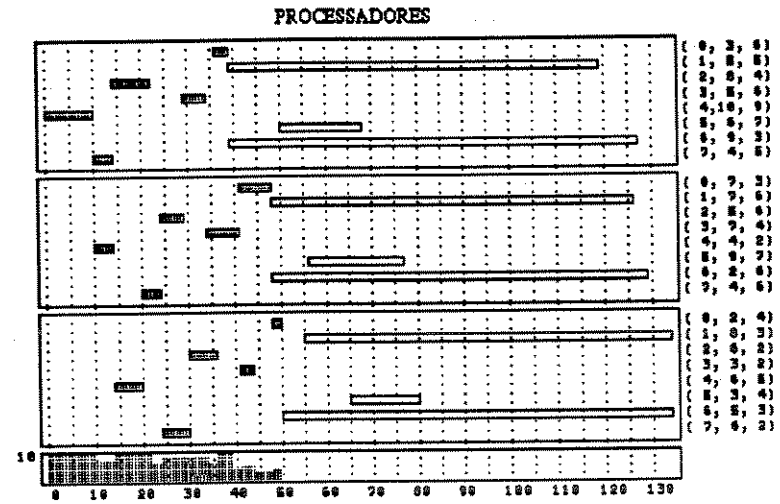


Figura 5.5- Estado do nó 47230.

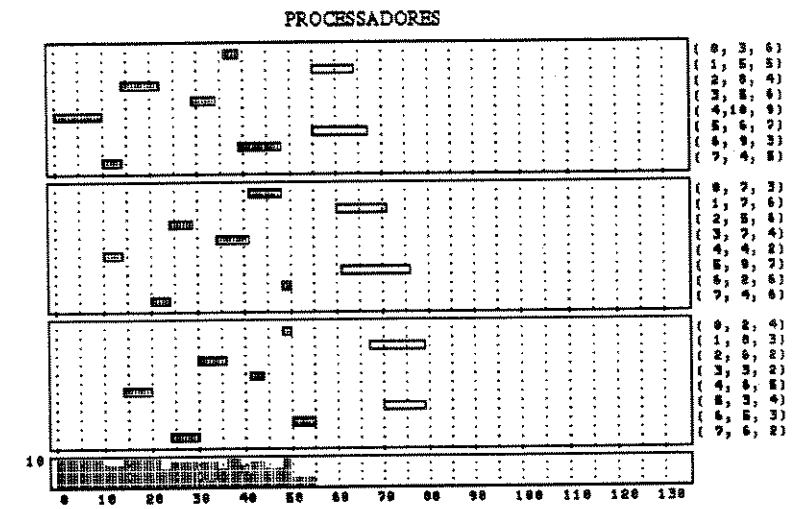


Figura 5.6- Estado do nó 472306.

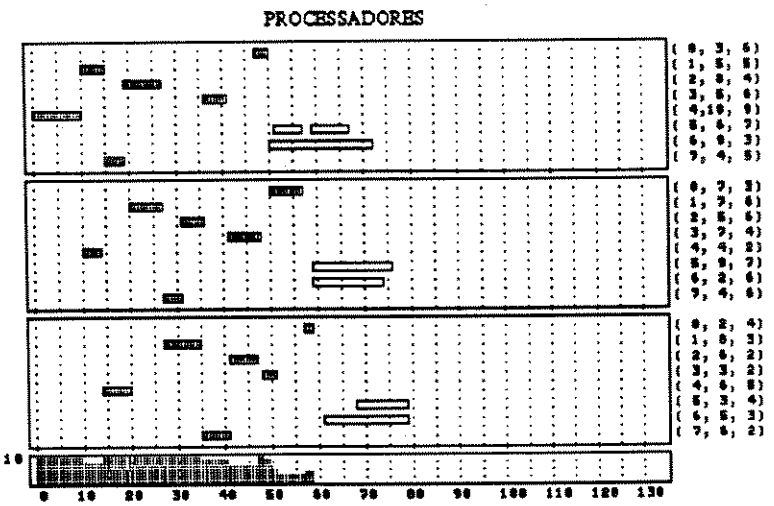


Figura 5.7- Estado do nó 417230.

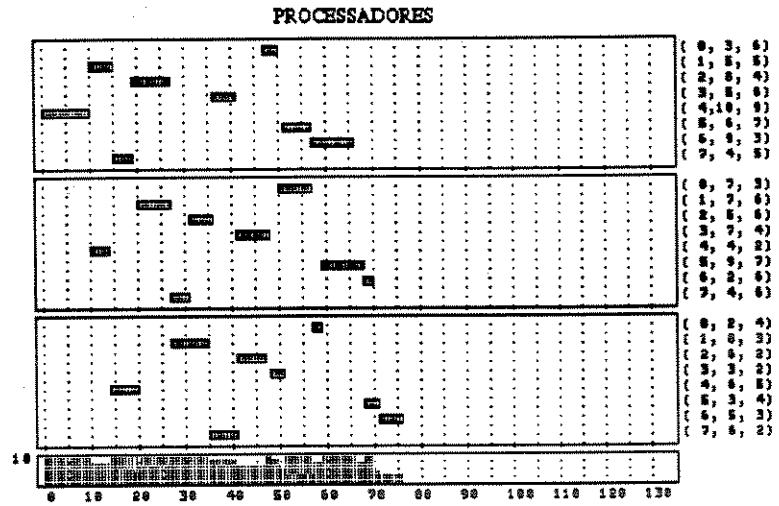


Figura 5.8- Estado do nó 41723056: solução ótima.

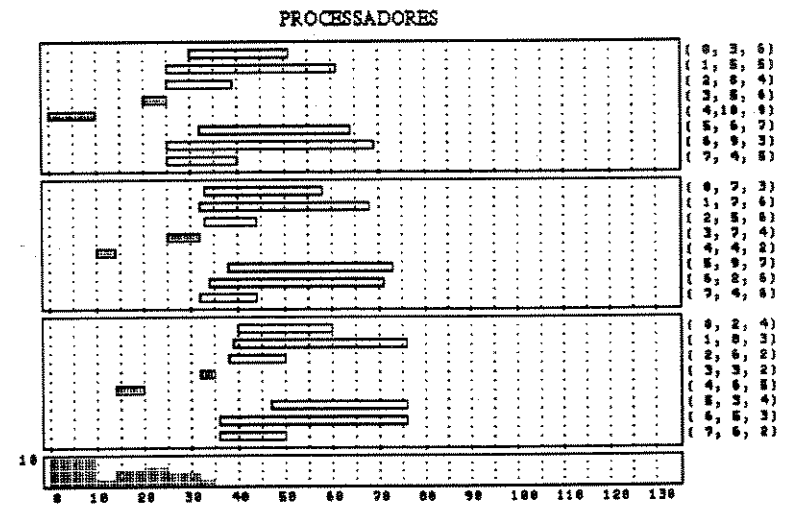


Figura 5.9- Estado do nó 43.

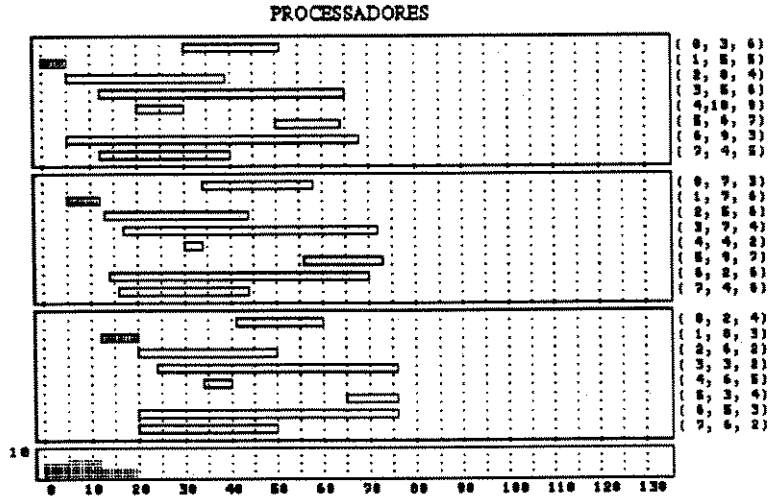


Figura 5.10- Estado do nó 1.

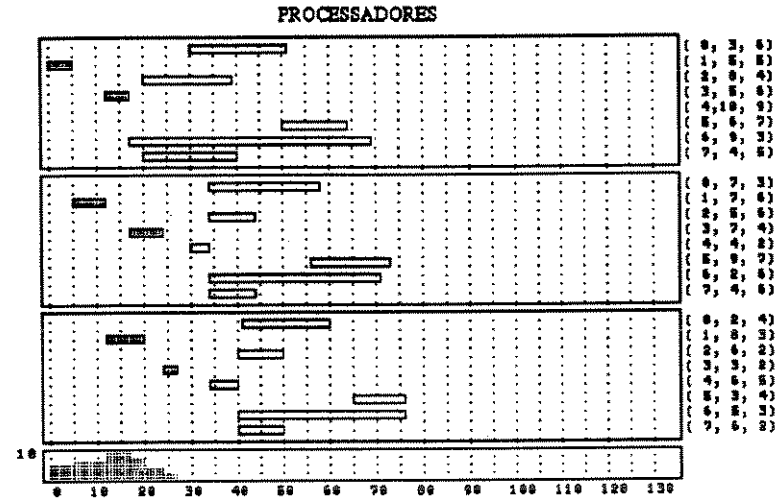


Figura 5.11- Estado do nó 13 (este nó não é pesquisado).

Comentários:

O desempenho do algoritmo em relação ao número de tarefas alocadas externamente e à dimensão destas janelas não é fácil de ser determinado porque estão envolvidos múltiplos fatores inter-dependentes: o tempo total de processamento das tarefas, os tempos de processamento de cada uma de suas operações, a quantidade de recurso necessário ao seu processamento e a disponibilidade deste recurso.

As simulações realizadas permitiram, no entanto, observar o seguinte comportamento do BPAR:

1- em relação ao número de alocações externas: o tempo total gasto (TT) diminui com o aumento do número de alocações externas;

2- em relação ao tamanho das janelas: quanto menos flexível for a alocação de uma tarefa, maior será a diminuição do tempo gasto para determinar a solução ótima.

Uma observação importante com relação ao tamanho das alocações externas diz respeito ao uso da criticalidade da tarefa como heurística de seqüenciamento. Em quase todas simulações realizadas neste trabalho o BPAR encontrou a solução ótima para o problema proposto. No entanto a heurística pode levar à perda da solução ótima do problema, exemplo disto é o problema P11. Alocando as tarefas 0, 2, 4, 5 e 7 nos intervalos já mencionados o BPAR encontra a solução ótima (*makespan*=76). No entanto, se forem alocadas apenas as tarefas 0, 2, 4 e 7 o algoritmo não consegue mais chegar à solução ótima (a solução encontrada tem *makespan* igual a 79), porque a tarefa 5 acaba sendo atrasada pela heurística em função da criticalidade das outras tarefas. Isto sugere a necessidade de uma heurística mais elaborada, capaz de identificar com mais precisão o impacto da alocação de uma tarefa sobre as demais, como meio de evitar problemas desta natureza.

5.3- *Flow-shop* sem Alocações Externas.

O algoritmo BPAR pode ser utilizado apenas com restrições sobre o recurso compartilhado, não havendo alocação externa de tarefas. O interesse do algoritmo nesta situação vem da diminuição de nós explorados devido à utilização de janelas para todas as tarefas, janelas que são definidas a partir do limitante superior do BAB. Como é feita busca em profundidade, após N iterações, todas as tarefas, que a princípio tinham um horizonte de seqüenciamento igual a TF, passam a ser limitadas pelo limitante superior f* (ou seja, o prazo de entrega de todas as tarefas internas passa a ser f*). Isto limita o número de seqüências factíveis possíveis (que era N!), permitindo que a análise de restrições elimine nós que, pelo critério do LB (LB<f*), poderiam ainda ser expandidos .

As Tabelas 5.7 e 5.8 apresentam os resultados obtidos para os 12 problemas propostos, utilizando os algoritmos BAB(C1,C2) com busca em profundidade e os algoritmos BPAR(C1,C2).

Tabela 5.7- Resultados: BAB-BP x BPAR (Algoritmo C1)

	BAB-BP(C1)			BPAR(C1)			
P	NS	SC	TT	NS	SC	TT	H
1	17478	2255	82.65	9061	7	63.90	276
2	19201	2972	90.53	9262	5	63.33	131
3	19908	3480	98.78	10397	26	74.10	175
4	15618	972	74.78	7505	22	55.72	281
5	15919	2958	70.68	8650	9	50.77	158
6	21212	2724	95.60	10543	8	71.70	470
7	42767	15000	320.38	21674	29	131.11	60
8	9182	458	43.6	4483	5	31.58	211
9	12053	1588	53.37	6507	18	39.23	256
10	11186	627	49.15	5432	10	35.70	335
11	4297	321	10.92	2820	12	7.75	574
12	246397	11704	1328.80	157656	53	1002.26	7505

Tabela 5.8- Resultados: BAB-BP x BPAR (Algoritmo C2)

	BAB-EP(C2)			BPAR(C2)			
P	NS	SC	TT	NS	SC	TT	H
1	13237	521	80.97	8806	5	69.21	198
2	14542	775	85.25	8897	4	67.85	115
3	14217	1084	92.73	9922	23	77.99	117
4	13069	397	81.33	7431	15	63.68	269
5	11050	194	64.46	8430	7	55.30	146
6	17402	1205	97.98	10371	9	78.56	433
7	27442	2529	149.29	21420	26	132.38	52
8	7616	246	48.49	4397	5	36.24	208
9	8666	120	54.73	6336	14	43.61	247
10	8888	192	52.56	5280	9	40.42	312
11	2545	7	9.80	2275	7	7.59	490
12	164746	1140	1166.65	132999	24	1037.15	4589

O algoritmo BPAR obteve a solução ótima para todos os problemas, apesar de usar a heurística apresentada na seção 4.3.3.1. Nota-se que houve uma redução significativa de NS, SC e TT da BAB-BP(C1) para o BAB-BPAR(C1), o mesmo acontecendo para o C2. Observa-se também que da BP(C1) para a BP(C2) houve uma redução significativa de NS e SC e uma redução razoável de TT. O mesmo não é verdade para BPAR(C1) e BPAR(C2), sendo que o TT deste último chega a ser maior do que o TT da BPAR(C1). Como a análise de restrições se sobrepõe ao cálculo melhor feito pelo algoritmo C2, o diferença de nós sondados pelos dois algoritmos é pequena e C2 acaba demorando mais tempo do que C1.

5.4- Utilização do Algoritmo com Janelas Proibidas.

O algoritmo BPAR desenvolvido permite a sua utilização na situação inversa: existem janelas definidas *a priori* onde um processador não estará disponível para nenhuma operação. Esta situação pode corresponder, p.ex., à manutenção de equipamentos, ou a não disponibilidade de um equipamento porque ele é compartilhado por duas linhas de produção.

Para lidar com restrições como a necessidade de parada de um processador durante a realização das tarefas seria necessário criar restrições de tempo (janelas) para todas as operações que utilizam aquele processador, o que provocaria o aumento da complexidade do problema, pois haveria um número maior de restrições a serem satisfeitas. O BPAR permite que restrições deste tipo sejam modeladas como uma tarefa (fictícia) com uma operação alocada (alocação fixa) no instante previsto da parada do processador, com tempos de processamento nulos nos demais processadores e com consumo de recursos compartilhados igual a zero.

O problema P11 da Tabela 5.2 será resolvido pelo BPAR supondo que haverá uma parada do processador 2 no período que vai de 30 a 40 ut. Para isso é criada uma tarefa fictícia 8, com TP=10 no processador 2 e TP=0 nos processadores 1 e 3, uma vez que não existe restrições de utilização destes processadores. Ao tempo de pronto da tarefa fictícia é atribuído o valor 30 e ao prazo de entrega o valor

40 (alocação fixa). A solução ótima encontrada foi a seqüência 416835702 com makespan igual a 78. Para chegar a esta solução o BPAR explorou 2104 nós, encontrou 11 soluções factíveis e gastou 6.84 segundos. As Figuras 5.13 e 5.14 mostram a carta de Gantt e o perfil de consumo do recurso compartilhado para esta solução.

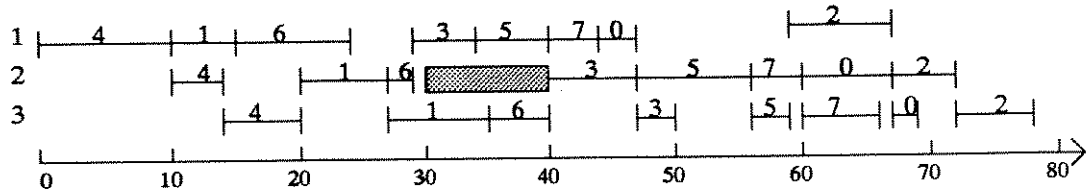


Figura 5.13- Carta de Gantt da seqüência ótima 41635702.

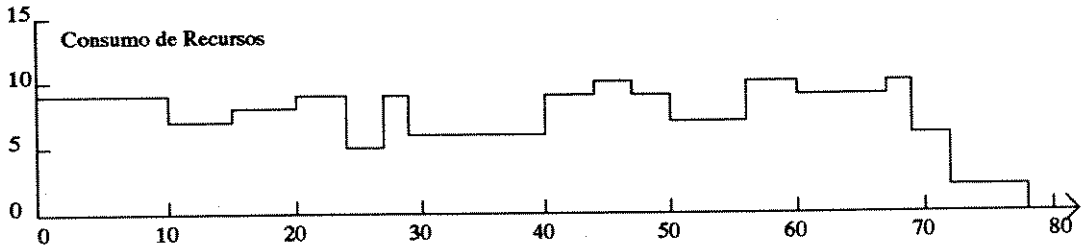


Figura 5.14- Perfil de consumo do recurso compartilhado para a seqüência ótima.

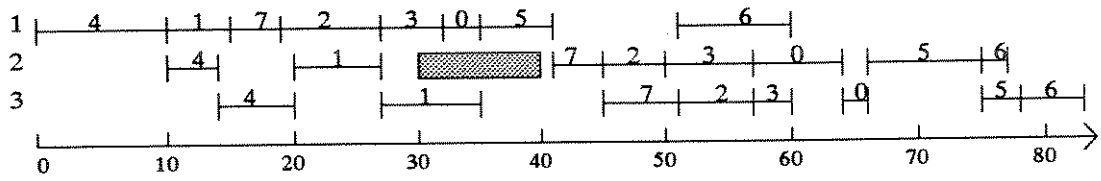


Figura 5.15- Carta de Gantt da seqüência 41723056 factibilizada.

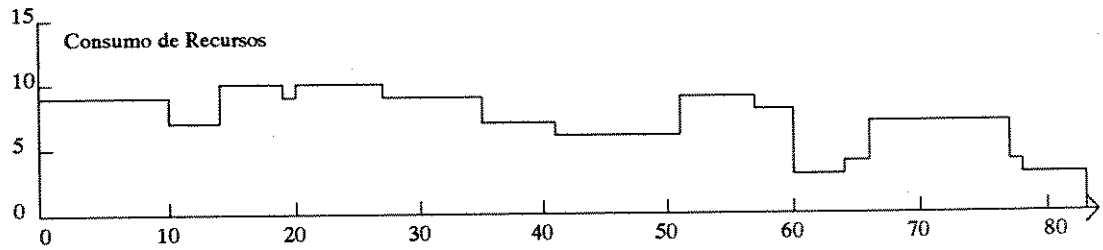


Figura 5.16- Perfil de consumo do recurso compartilhado.

Como era de se esperar a solução encontrada é completamente diferente da solução do problema original. As figuras 5.15 e 5.16 mostram a factibilização *a posteriori* da solução original do

problema para acomodar a parada do processador 2. O makespan da seqüência passou de 76 para 83 ut. Assim como a relaxação das restrições sobre recursos e a posterior factibilização da solução encontrada (seção 3.3.4) não é uma estratégia aceitável, também a relaxação de restrições temporais, como a apresentada aqui, seguida de factibilização não é uma estratégia aceitável.

5.5- Conclusões.

Mostrou-se a utilização do BPAR em três situações diferentes: *flow-shop* com alocações externas, *flow-shop* sem alocações externas e *flow-shop* com janelas proibidas. Nos três casos considerou-se restrições sobre os recursos compartilhados.

Embora não tenha sido feita uma análise estatística do desempenho do algoritmo, chegou-se à conclusão que seu desempenho é bom, se comparado com um BAB que não selecione as tarefas candidatas a fazer parte de uma seqüência parcial representada por um nó.

Mostrou-se que o conceito de criticalidade de tarefas é muito útil para reduzir o espaço de busca do problema e conduz a bons resultados quando utilizado apenas nas situações de conflito entre tarefas, diferente do que acontece nas propostas de Keng [Keng-88], Sadeh [Sadeh-89] e Sycara [Sycara-91], em que as heurísticas são utilizadas sempre e por isso não conseguem otimizar o desempenho da planta.

Conclusões

Neste trabalho foi proposta uma estratégia de solução para o problema de minimização do *makespan* em sistemas de produção do tipo *flow-shop* com seqüências de permutação, alocações externas e restrições sobre os recursos compartilhados, que é um problema comum e muito importante na indústria de processos químicos.

A estratégia proposta combina técnicas de inteligência artificial, voltadas para a satisfação das restrições do problema, com uma técnica da pesquisa operacional denominada *Branch-and-Bound*, a qual permite encontrar a solução ótima do problema. Utiliza-se também uma regra heurística que estende o conceito de criticalidade de tarefas na presença de recursos compartilhados e que permite reduzir o espaço de busca do problema sem comprometer a otimalidade da solução.

Resumindo, o algoritmo proposto consiste de um pré-processamento que é aplicado a cada nó da árvore de busca, antes de sua explosão, para selecionar entre todas as tarefas candidatas a fazer parte da seqüência parcial do nó aquelas que potencialmente levarão a uma seqüência final factível. A minimização do makespan é feita pelos algoritmos C1 e C2 propostos em [Rodrigues-92] e a busca é feita em profundidade. A estratégia de selecionar as tarefas candidatas a fazer parte da seqüência permite que o BAB se aproxime da BS (busca com *backtracking*) da inteligência artificial, que no caso limite pode ser vista como uma regra de despacho.

Os resultados do capítulo 5 mostram que o algoritmo consegue dirigir bem a busca em direção à solução ótima, não importando a natureza das janelas externas das tarefas. A heurística utilizada para reduzir a dimensão do problema geralmente não impede a obtenção da solução ótima. Mostra-se também o bom desempenho do algoritmo quando aplicado a problemas em que não existem alocações externas. Neste caso as alocações externas são definidas internamente pelo algoritmo sempre que ele consegue encontrar um limitante superior menor para as soluções do problema. Finalmente mostra-se que o algoritmo pode ser empregado para resolver problemas onde existem janelas de tempo que são proibidas para um conjunto de operações. Neste caso, em lugar de se criar alocações externas para cada uma das operações em questão, a janela proibida é modelada como uma operação que possui uma alocação externa fixa e consumo de recursos compartilhados igual a zero.

Como propostas para trabalhos futuros pode-se citar:

- a extensão da técnica proposta para um modelo da planta mais geral, que considere entre outra coisas: uma estrutura de processamento mais genérica (*job-shop* ou *open-shop*), armazenagem intermediária limitada, e processamento paralelo de tarefas;

- a combinação desta técnica com a estrutura multinível proposta em [Passos-93], que permite classificar os diversos recursos compartilhados de forma a resolver o problema em diferentes níveis de abstração, com a conseqüente diminuição da dimensão do espaço de busca do problema;4

- refinar a heurística utilizada no algoritmo, para que a cada iteração o conjunto de tarefas candidatas a fazer parte da seqüência que está sendo gerada contenha o menor número de elementos, se possível um único, aproximando o BAB de uma regra de despacho. Isto permitiria o seu uso em problemas de grandes dimensões;

- adaptar a técnica proposta para permitir o *scheduling* dinâmico de tarefas.

Apêndice A: Dados dos Problemas Propostos no Capítulo 5.

As Tabelas A1 a A12 contém os tempos de processamento e os consumos de recurso das operações dos 12 problemas propostos no capítulo5.

TABELA A.1 - Dados do problema P1

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	10	6	15	4	10	17	8	1	4	1	8	7
T. 1	4	10	5	17	19	9	8	1	6	9	4	7
T. 2	14	6	4	13	11	15	8	7	3	7	4	6
T. 3	4	8	6	12	17	6	1	7	4	7	6	1
T. 4	8	17	19	7	17	9	6	5	8	1	4	4
T. 5	15	8	13	11	5	13	2	1	5	6	9	7
T. 6	10	3	19	12	1	19	6	1	4	5	1	2
T. 7	17	3	1	9	9	19	1	4	7	3	4	9

TABELA A.2 - Dados do problema P2

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	3	8	1	12	10	10	6	3	1	3	5	1
T. 1	13	7	17	14	14	3	5	7	7	9	5	6
T. 2	18	14	17	16	15	14	1	7	6	4	6	6
T. 3	9	11	15	18	5	17	9	9	6	6	2	3
T. 4	10	1	2	12	5	8	2	4	2	5	2	2
T. 5	2	15	4	4	6	10	2	8	4	2	2	4
T. 6	15	18	14	2	14	17	7	9	5	1	6	5
T. 7	10	13	2	5	9	3	7	5	3	5	6	1

TABELA A.3 - Dados do problema P3

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	10	18	14	18	9	12	7	2	7	1	6	9
T. 1	3	9	6	16	5	4	8	1	9	3	3	7
T. 2	18	14	12	13	9	18	5	8	7	1	9	8
T. 3	18	16	12	9	11	15	7	7	6	9	7	4
T. 4	6	18	4	15	16	11	8	3	2	4	6	9
T. 5	18	15	1	12	9	4	3	9	8	9	3	6
T. 6	5	18	6	11	4	19	1	1	4	2	3	7
T. 7	18	10	16	17	15	7	8	2	2	2	6	4

TABELA A.4 - Dados do problema P4

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	13	15	1	9	5	12	3	8	5	4	8	6
T. 1	2	3	11	10	17	6	2	4	5	5	4	5
T. 2	11	19	9	19	16	17	5	1	8	8	7	8
T. 3	4	13	18	16	1	7	2	3	2	7	7	3
T. 4	8	5	10	16	9	10	3	9	3	4	6	1
T. 5	15	16	9	1	17	8	5	5	1	3	3	8
T. 6	10	6	2	9	15	14	8	7	5	1	6	9
T. 7	6	18	13	18	3	18	8	3	8	5	3	7

TABELA A.5 - Dados do problema P5

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	15	14	4	14	4	3	6	2	9	5	4	1
T. 1	11	10	13	17	5	16	3	5	7	2	9	9
T. 2	7	12	7	12	19	7	1	2	8	6	2	3
T. 3	16	1	3	11	5	12	8	5	1	6	3	3
T. 4	18	18	13	16	6	19	9	7	8	9	6	7
T. 5	3	8	13	4	2	1	7	8	2	5	3	4
T. 6	3	9	10	17	1	2	2	2	1	5	3	1
T. 7	10	3	2	9	16	4	3	3	2	7	1	5

TABELA A.6 - Dados do problema P6

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	1	18	13	10	2	9	3	7	5	9	7	6
T. 1	10	5	18	12	13	3	5	8	9	2	1	7
T. 2	14	8	13	17	9	4	5	3	4	9	5	2
T. 3	7	8	2	3	19	14	6	1	4	3	4	7
T. 4	3	4	16	8	19	16	1	6	6	7	4	3
T. 5	15	3	14	10	5	13	8	5	1	5	7	8
T. 6	13	1	19	16	8	16	5	8	2	7	5	7
T. 7	15	4	4	2	6	9	6	6	4	2	5	5

TABELA A.7 - Dados do problema P7

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	7	9	7	16	1	10	4	6	1	7	8	4
T. 1	19	7	7	14	4	5	8	2	2	5	9	8
T. 2	14	9	9	14	3	8	9	3	5	6	8	3
T. 3	16	13	18	13	6	12	4	8	4	6	8	8
T. 4	19	3	18	11	11	1	7	5	5	6	8	1
T. 5	12	3	11	18	8	8	6	6	5	3	9	9
T. 6	3	1	11	6	18	19	8	9	1	7	2	9
T. 7	4	1	18	14	3	13	6	2	3	8	5	8

TABELA A.8 - Dados do problema P8

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	14	19	1	3	19	12	4	4	6	4	1	3
T. 1	9	9	14	16	14	7	3	6	4	8	7	9
T. 2	14	9	5	12	16	12	3	4	7	4	1	5
T. 3	6	19	15	3	12	10	2	6	5	1	4	9
T. 4	15	1	1	14	3	4	4	4	5	5	2	7
T. 5	9	2	9	7	10	3	9	7	1	2	1	1
T. 6	12	19	3	15	8	2	1	1	8	8	7	2
T. 7	12	18	13	8	10	17	7	8	3	5	5	2

TABELA A.9 - Dados do problema P9

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	2	10	14	10	18	13	5	2	3	1	2	1
T. 1	18	11	2	18	5	9	6	9	6	2	5	1
T. 2	14	10	1	10	10	16	7	5	8	2	4	7
T. 3	14	5	12	13	18	9	9	3	5	4	9	1
T. 4	12	18	3	17	14	8	1	3	5	5	6	5
T. 5	7	1	6	15	13	16	1	7	5	1	2	3
T. 6	1	19	14	5	17	4	4	2	7	8	4	4
T. 7	1	15	7	2	16	2	7	4	2	2	5	6

TABELA A.10 - Dados do problema P10

	tempos de processamento.						consumos de recurso					
	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6	O. 1	O. 2	O. 3	O. 4	O. 5	O. 6
T. 0	8	1	8	16	17	15	6	1	8	7	3	9
T. 1	8	13	9	1	15	11	9	3	7	5	4	2
T. 2	14	10	15	8	5	1	1	6	1	9	7	9
T. 3	4	11	9	4	5	7	7	1	5	7	5	2
T. 4	8	16	6	1	7	12	6	2	4	4	1	2
T. 5	4	19	4	4	16	11	2	8	1	8	3	4
T. 6	10	19	6	13	7	7	6	3	5	8	1	6
T. 7	9	13	2	14	3	6	8	1	2	8	5	9

TABELA A.11 - Dados do problema P11

	tempos de proc.			consumos de recurso		
	O. 1	O. 2	O. 3	O. 1	O. 2	O. 3
T. 0	3	7	2	6	3	4
T. 1	5	7	8	5	6	3
T. 2	8	5	6	4	6	2
T. 3	5	7	3	6	4	2
T. 4	10	4	6	9	2	5
T. 5	6	9	3	7	7	4
T. 6	9	2	5	3	6	3
T. 7	4	4	6	5	6	2

TABELA A.12 - Dados do problema P12

	tempos de processamento				consumos de recurso			
	O. 1	O. 2	O. 3	O. 4	O. 1	O. 2	O. 3	O. 4
T. 0	2	15	14	4	5	6	4	7
T. 1	14	13	7	18	4	5	2	1
T. 2	5	4	1	10	6	8	4	1
T. 3	19	20	17	4	4	7	7	2
T. 4	11	12	17	12	5	4	8	5
T. 5	9	20	11	13	5	8	8	7
T. 6	20	14	14	16	7	5	9	6
T. 7	9	2	2	6	7	6	5	7
T. 8	6	3	7	11	2	7	7	6
T. 9	3	14	15	11	3	2	5	9

REFERÊNCIAS

1. J.Adams, E.Balas and D.Zawack, "The shifting Bottleneck procedure for job-shop scheduling", *Management Science*, vol.34, n.3, pp.391-401, 1988.
2. B.Adenso-Díaz, "Restricted neighborhood in the tabu search for the flowshop problem", *European Journal of Operational Research*, vol.62, pp.27-37, 1992.
3. H.Atabakhsh, "A survey of constraint based scheduling systems using an artificial intelligence approach", *Artificial Intelligence in Engeneering*, vol.6, n.2, pp.58-73, 1991.
4. K.R.Baker(a), "A comparative study of flow-shop algorithms", *Operations Research*, vol.23, n.1, pp.62-73, 1973.
5. K.R.Baker(b), "An elimination method for the flow-shop problem", *Operations Research*, vol.23, n.1, pp.62-73, 1973.
6. K.R.Baker, *Introduction to Sequencing and Scheduling*, John Wiley and Sons, New York, 1974.
7. K.M.Baumgartner, "Computer scheduling algorithms: past, present, and future", *Information Sciences*, **57-58**, pp.319-345, 1991.
8. R.Bellman, A.O.Esogbue and I.Nabeshina, *Mathematical Aspects of Scheduling and Applications*, Pergamon Press, 1982.
9. G.Bel, E.Bensana, D.Dubois, J.Erschler and P.Esquirol, "A knowledge-based approach to industrial job-shop scheduling", Knowledge-Based Systems in Manufacturing, Ed.Taylor & Francis, New York, 1989.
10. E.Bensana, G.Bel and D.Dubois, "OPAL: a multiknowledge-based system for industrial job-shop scheduling", *International Journal of Production Research*, vol.26, n.5, pp.795-819 1988.
11. P.Burke and P.Prosser, "A distribuited assynchronous system for preditive and reative scheduling", *Artificial Intelligence in Engineering*, vol.6, n.3, pp.106-124, 1991.
12. J.Carlier and E.Pinson, "An algorithm for solving the job-shop problem", *Management Science*, vol.35, n.2, pp.164-175, 1989.
13. C.Chu, "A branch-and-bound algorithm to minimize total flow time with unequal release dates", *Naval Research Logistics*, vol.39, pp.859-875, 1992.
14. E.G.Coffman,Jr., *Computer and Job-Shop Scheduling Theory*, John Wiley and Sons, New York, 1976.
15. R.Conway,W.Maxwell,J.O.McClain and L.J.Thomas, "The role of WIP inventory in serial production lines", *Operations Research*, vol.36, n.2, pp.229-241, 1988.
16. K.Currie and A.Tate, "O-Plan: the open planning architecture", *Artificial Intelligence*, vol.52, pp.49-86, 1991.
17. D.G.Dannenbring, "An evaluation of flow-shop sequencing heuristics", *Management Science*, vol.23, n.11, pp.11174, 1977.
18. H.Das, P.T.Cummings and M.D.Le Van, "Scheduling of serial multiproduct batch process via simulated annealing", *Computers and Chemical Engeneering*, vol.14, n.12, pp.1351-1362, 1990.
19. R.Dattero, J.J.Kanet and E.M.White,"Enhancing manufacturing planning and control systems with artigicial intelligence techniques", Knowledge-Based Systems in Manufacturing, Ed.Taylor & Francis, New York, pp.137-150, 1988.
20. R.Dechter, "Enhancement schemes for constraint processing: backjumping, learning and cutset decomposition", *Artificial Intelligence*, vol.41, pp.273-312, 1989/1990.
21. E.Demenlemeester and W.Herroelen, "A branch-and-bound procedure for the multiple Resource constrained Project Scheduling Problem", *Management Science*, vol.38, n.12, pp.1803-1818, 1992.
22. R.A.Dudek, S.S.Paawalkar and M.L.Smith,"The lessons of flowshop scheduling research", *Operations Research*, vol.40, n.1, pp.7-13, 1992.
23. J.Erschler and P.Esquirol, "Decision-aid in job-shop scheduling: a knowledge-based approach", *IEEE International Symposium on Robotics and Automation*, pp.1651-1656, 1986.
24. J.Erschler and F.Roubellat, "An approach for real time scheduling of activities with time and resource constraints", Advances in Project Scheduling, edited by R.Slowinski and J.Weglarz, Elsevier Science Publishers B.V., Amsterdam, pp.237-272, 1989.
25. J.Erschler, F.Roubellat and J.P.Vernhes, "Finding some essential characteristics of possible solutions for scheduling problem", *Operations Research*, vol.24, n.4, pp.775-783, 1976.

26. S.French, *Sequencing and Scheduling. An Introduction to the Mathematics of the Job-Shop*, Ellis Horwood Limited, Chichester, 1982.
27. M.S.Fox, "Constraint-directed search: a case study of the job-shop scheduling", Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, December 1983.
28. M.R.Garey and D.Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H.Freeman and Company, San Francisco, 1979.
29. T.J.Grant, "Lessons for O.R. from A.I.: a scheduling case study", *Journal of the Operational Research Society*, vol.37, n.1, pp.41-58, 1986.
30. S.C.Graves, "A review of production scheduling", *Operations Research*, vol.29, n.4, pp.647-668, 1981.
31. G.Harhalakis, "Evaluation of resource allocations in project scheduling", *Advances in Project Scheduling*, ed. by R.Slowinski and J.Weglarz, Elsevier Science Publishers B.V., Amsterdam, pp.67-86, 1989.
32. E.Horowitz and S.Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1978.
33. S.Karabati, P.Kouvelis and A.S.Kiran, "Games, critical path and assigement problems in permutation flow-shops and cyclic scheduling flow-time enviroments", *J.Opl.Res.Soc.*, vol.43, n.3, pp.241-258, 1992.
34. N.P.Keng, D.Y.Y.Yun and M.Rossi, "Interaction-sensitive planning system for job-shop scheduling", Expert Systems and Intelligent Manufacturing, ed. by Michael D.Oliff, pp.57-69, 1988.
35. S.Kirkpatrick, C.D.Gelatti and M.P.Vecchi, "Optimization by simulated annealing", *Science*, vol.220, n.4598, pp.671-680, 13 may 1983.
36. F.C.Knopf, "Sequencing a generalized two-stage flowshop with finite intermediate storage", *Computers and Chemical Engineering*, vol.9, n.3, pp.207-221, 1985.
37. E.Kondili, C.C.Pantelides and R.W.H.Sargent, "A general Algorithm for short term scheduling of batch operation - I: MILP formulation.", *Computer and Chemical Engineering*, vol.17, n.2, pp.211-227, 1993.
38. R.E.Korf, "Depth-first iterative-deepening: an optimal admissible tree search", *Artificial Intelligence*, vol.27, pp.97-109, 1985.
39. K.Kuriyan and G.V.Reklaitis, "Scheduling network flowshops so as to minimize makespan", *Computers and Chemical Engineering*, vol.13, n.1/2, pp.187-200, 1989.
40. A.Kusiak, "Designing Expert Systems for scheduling of automated manufacturing", *Industrial Engineering*, vol.19, n.9, pp.42-46, 1987.
41. C.Le Pape, "Using constraint propagation in blackboard systems: a flexible software architecture for reative and distribuited systems", *Computer*, may 1992, pp.60-62.
42. R.Leinsten, "Flow-shop sequencing problems with limited buffer storage", *Int.J.Prod.Res.*, vol.28, n.11, pp.2085-2100, 1990.
43. V.J.Leon and S.D.Wu, "On scheduling with ready-times, due-dates and vacations", *Naval Research Logistics*, vol.39, pp.53-65, 1992
44. K.N.Mckay, F.R.Safayeni and J.A.Buzacott, "Job-shop scheduling theory: what is relevant?", *Interfaces*, 18:4, pp.84-90, 1988.
45. R.F.H.Musier and L.B.Evans, "An approximate method for production scheduling of industrial batch process with parallel units", *Computers and Chemical Engineering*, vol.13, n.1/2, pp.229-238, 1989.
46. P.A.Newman, "Scheduling in CIM systems", Artificial Intelligence Implications for CIM, Edited by A.Kusiak, IFS(Publications) Ltd. & Springer-Verlag, UK, pp.361-402, 1988.
47. N.Nilsson, *Principles of Artificial intelligence*, Palo Alto, CA, 1980.
48. A.Nouweland, M.Krabbenborg and J.Potters, "Flow-shop with dominant machine", *European Journal of Operational Research*, vol.62, pp.38-46, 1992.
49. F.A.Ogbu and D.K.Smith, "The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem", *Computers & Operations Research*, vol.17, n.3, pp.243-253, 1990.
50. P.S.Ow, S.F.Smith and R.Howie, "A cooperative scheduling system", Expert Systems and Intelligent Manufacturing, Ed.Elsevier Science Publishing Co., pp.43-56, 1988.

51. S.S.Panwalkar and W.Iskander, "A survey of scheduling rules", *Operations Research*, vol.25, n.1, pp.45-61, 1977.
52. C.A.S.Passos, "Uma abordagem multinível para o problema do seqüenciamento de flow-shops com oferta limitada de recursos em indústrias de processos químicos", Tese de Doutorado, Unicamp, setembro de 1993.
53. J.H.Patterson, R.Slowinski, F.B.Talbot and J.Weglars, "An algorithm for general class of precedence and resource constrained scheduling problems", Advances in Project Scheduling, ed. by R.Slowinski e J.Weglarz, Elsevier Science Publishers B.V., Amsterdam, pp.3-28, 1989.
54. J.Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley Publishing Company, 1984.
55. J.F.Pekny and D.L.Miller, "Exact solution of the no-wait flowshop scheduling problem with a comparation to heuristic methods", *Computers and Chemical Engineering*, vol.15, n.11, pp.741-748, 1991.
56. C.N.Potts and L.N.Van Wassenhove, "Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity", *J.Opl.Res.Soc.*, vol.43, n.5, pp.395-406, 1992.
57. D.Rajagopalan and I.A.Karimi, "Completion times in serial mixed-storage multiproduct process with transfer and set-up times", *Computers and Chemical Engineering*, vol.13, n.1/2, pp.175-186, 1989.
58. C.Rajendran and D.Chaudhuri, "A multi-stage parallel-processor flowshop problem with minimum flowtime", *European Journal of Operational Research*, **57**, pp.111-122, 1992.
59. R.Ramazani and N.Younis, "Repetitive pure flow-shop problem: a permutation approch.", *Computer and Industrial Engineering*, vol.24, n.1, pp.125-129, 1993.
60. J.Rickel, "Issues in the design of scheduling systems", Expert Systems and Intelligent Manufacturing, ed. by Michael D. Oliff, pp.70-89, 1988.
61. F.A.Rodammer and K.P.White, "A recent survey of production scheduling", *IEEE Trans. Systems, Man, and Cybernetics*, vol.18, No.6, pp.841-851, 1988.
62. M.T.M.Rodrigues, "Seqüenciamento e alocação de operações em flow-shops na indústria química com restrições sobre os recursos compartilhados: uma abordagem de busca orientada por restrições", Tese de Doutorado, Unicamp, agosto de 1992.
63. R.V.Rogers and K.P.White Jr., "Algebric, mathematical programming, and network models of the deterministic job-shop scheduling problem", *IEEE Trans. on Systems, man, and cybernetics*, vol.21, n.3, pp.693-697, may/june 1991.
64. E.D.Sacerdoti, "Planning in a hierarchy of abstrations spaces", *Artificial Intelligence*, vol.5, pp.115-135, 1974.
65. N.Sadeh and M.S.Fox, "CORTES: An exploration into micro-opportunistic job-shop scheduling", *AAAI-Sigman - Workshop on Manufacturing Scheduling / IJCAI - 89/23, august.*
66. N.Shah, C.C.Pantelides and R.W.H.Sargent, "A general Algorithm for short term scheduling of batch operation - II: Computational Issues.", *Computer and Chemical Engineering*, vol.17, n.2, pp.229-244, 1993.
67. S.F.Smith and P.S.Ow, "The use of multiple problem decompositions in time constrained planning tasks", *Proceedings of the 9th. International Joint Conference on Artificial Intelligence*,pp.1013-1015, 1985.
68. S.F.Smith, P.S.Ow, J.Y.Potvin, N.Muscettola and D.C.Matthys, "An integrated framework for generating and revising factory schedules", *Journal of the Operational Research Society*, vol.41, n.6, pp.539-552, 1990.
69. S.F.Smith and D.K.Pathak, "Balancing antagonistic time and resource utilization constraints in over-subscribed scheduling problems", *IEEE*, pp.113-119, 1992.
70. Y.N.Sotskov, "The complexity of shop-scheduling problems with two or tree jobs", *European Journal of Operational Research*, **53**, pp.326-336, 1991.
71. M.S.Steffen and T.J. Greene, "An application of hierarchical planning and constraint-directed search to scheduling parallel processors", *IEEE International Symposium on Robotics and Automation*, pp.910-917, 1986.
72. K.Sycara(a), S.Roth, N.Sadeh and M.S.Fox, "Distributed Constrained Heuristic Search", *IEEE Trans. systems, man, and cybernetics*, vol.21, n.6, pp.1446-1460, 1991.

73. K.Sycara(b), S.Roth, N.Sadeh and M.S.Fox, "Resource allocation in distributed factory scheduling", *IEEE Expert*, pp.29-40, february 1991.
74. M.Tandon, P.T.Cummings and M.D.Le Van, "Flowshop sequencing with nom-permutation schedules", *Computers and Chemical Engeneering*, vol.15, n.8, pp.601-607, 1991.
75. A.Tate, "Applications of knowledge-based planning systems", Applications of Knowledge-Based Expert Systems in Industry, Ed. by Jiri Kriz, pp.130-144, Ellis Horwood Limited, 1987.
76. A.Toker, S.Kondakci and N.Erkip, "Scheduling under non-renewable resource constraint", *Journal of the Operational Research Society*, vol.42, n.9, pp.811-814, 1991.
77. L.H.Ungar and M.R.Weinstein, "Process scheduling using artificial intelligence", *Computer and Chemical Engineering*, received for publication in 5-09-89.
78. P.J.M.Van Laarhoven, E.H.L.Aarts and J.K.Lenstra, "Job shop scheduling by simulated annealing", *Operations Research*, vol.40, n.1, pp.113-125, 1992.
79. D.E.Wilkins, "Hierarchical Planning: Definition and Implementation", *Advances in Artificial Intelligence-II*, B.D.Boulay, D.Hogg and L.Steels(Editors), Elsevier Science Publishers B.V., North-Holland, 1987.