

UNIVERSIDADE FEDERAL DA PARAÍBA  
CENTRO DE CIÊNCIAS E TECNOLOGIA  
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

PROCEDIMENTOS HEURÍSTICOS EM PROBLEMAS  
DE SCHEDULING COM RESTRIÇÕES  
DE PRECEDÊNCIA E RECURSOS

VICTOR HUGO DE SOUZA LISBOA

CAMPINA GRANDE - PARAÍBA

JUNHO/1980



L769p Lisboa, Victor Hugo de Souza  
Procedimentos heurísticos em scheduling com restrições de precedência e recursos / Victor Hugo de Souza Lisboa. - Campina Grande, 1980.  
88 f. : il.

Dissertação (Mestrado em Ciências) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia.

1. Métodos Heurísticos 2. Scheduling - 3. Dissertação I. Brucker, Peter Joachim Siegfried, Dr. II. Universidade Federal da Paraíba - Campina Grande (PB) III. Título


CDU 004.023(043)

PROCEDIMENTOS HEURÍSTICOS EM PROBLEMAS DE SCHEDULING COM  
RESTRIÇÕES DE PRECEDÊNCIA E RECURSOS

VICTOR HUGO DE SOUZA LISBOA

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS CURSOS DE PÓS-GRADUAÇÃO EM SISTEMAS E COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DA PARAÍBA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

Aprovada por:



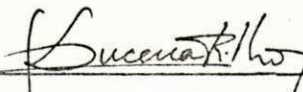
---

Peter Joachim Siegfried Brucker - Ph.D  
- Presidente -



---

Hans Hermann Weber - Ph.D  
- Examinador -



---

Gentil José de Lucena Filho - Ph.D  
- Examinador -

CAMPINA GRANDE  
ESTADO DA PARAÍBA - BRASIL  
JUNHO/1980



## AGRADECIMENTOS

Agradeço sinceramente

a meus pais

pelo apoio e incentivo, apesar da nossa sepa  
ração.

aos professores

Peter Joachim Siegfried Brucker e Gentil José  
de Lucena Filho, pelas suas valiosas colabora  
ções.

a Universidade Federal de Mato Grosso

pela oportunidade dada para a realização deste  
trabalho.

## R E S U M O

Este trabalho procura mostrar procedimentos heurísticos em problemas de scheduling com restrições de precedência e recursos. Para isto, apresenta procedimentos baseados em scheduling de listas que, embora não possam garantir soluções ótimas, produzem resultados aproximados.

Um modelo geral do problema de scheduling é apresentado e ilustrado com exemplos.

Um procedimento baseado no Algoritmo de Hu é implementado para problemas com estruturas de precedência especiais (estrutura de árvore). Uma generalização deste procedimento é apresentada e visa resolver problemas de scheduling com estruturas mais gerais de precedência e recursos. O procedimento resultante pode então ser aplicado, heurísticamente, a todos os demais problemas abordados neste trabalho.

## ABSTRACT

The purpose of this work is to show heuristic procedures in scheduling problems containing both types of constraints which characterize scheduling decisions: precedence constraints and resource constraints. For this, it shows procedures based in list scheduling which, although cannot guarantee optimal solutions, lead to good approximations.

One general model of the scheduling problem is presented and illustrated with examples.

One procedure based in Hu's Algorithm is implemented for problems with special precedence structure (tree structure). A generalization of this procedure is presented and aims to resolver scheduling problems with more general structures of precedence and resources. The resulting procedure may, then, be applied, (in an heuristic way) to all the other problems approached in this work.

## I N D I C E

CAPÍTULO		PÁGINA
I	INTRODUÇÃO	1
II	UM MODELO GERAL PARA PROBLEMAS DE SCHEDULING	5
	2.1 - Especificação do Problema	5
	2.2 - Relações de Precedência	6
	2.3 - Variáveis que descrevem a solução para o problema.	10
	2.4 - Medidas quantitativas para a avaliação de schedules.	11
	2.5 - Funções objetivas	12
	2.6 - Aplicações	15



CAPÍTULO		PÁGINA
III	COMPLEXIDADE EM PROBLEMAS DE SCHEDULING	26
	3.1 - Alguns aspectos da complexidade de um problema.	26
	3.2 - Complexidade de alguns problemas de scheduling.	28
	3.3 - O algoritmo de Hu: exato para resolver um problema da classe P e heurístico para um da classe NP-completo.	35
IV	PROCEDIMENTOS HEURÍSTICOS EM PLANEJAMENTO DE PROJETOS COM RESTRIÇÕES DE RECURSOS.	46
	4.1 - Aspectos de planejamento de projetos.	46
	4.2 - Procedimentos heurísticos em planejamento de projetos com restrições de recursos.	52
	4.3 - Resultados heurísticos de scheduling em projetos com restrições de recursos.	56
V	CONCLUSÕES	71
	APÊNDICE	75
	BIBLIOGRAFIA	84

## CAPÍTULO I

### INTRODUÇÃO

O grande desenvolvimento ora verificado no setor em presarial (indústrias e organizações, em geral) tem gerado problemas de natureza cada vez mais complexa. Nesse contexto, técnicas de pesquisa operacional têm encontrado um vasto campo de aplicação. Entre os problemas citados encontra-se o problema de alocação de recursos no tempo com vistas à execução de um certo conjunto de atividades. Tal problema é, tradicionalmente, denominado de problema de scheduling <sup>1</sup>.

---

(1) - O termo scheduling é usado durante todo o trabalho por não existir, em português, uma palavra que possa substituí-la num sentido completo.

Recursos e atividades são elementos básicos num problema de scheduling em que, tipicamente, procura-se responder perguntas tais como: "dado um conjunto de recursos e atividades, quando e quais recursos serão alocados para executar cada atividade?" Como, em geral, existem restrições, sejam elas de natureza quantitativa, tecnológica ou de precedência, quanto aos recursos disponíveis e atividades, encontrar uma resposta "adequada" para a pergunta acima toma a conotação de um problema de otimização sujeito a restrições. Isto porque, sempre que a quantidade de recursos requerida em cada período de tempo excede a quantidade de recursos disponível (o que, em geral, ocorre), necessário se faz que uma schedule seja feita no sentido de procurar otimizar a alocação dos recursos sobre o tempo entre as diversas atividades em questão.

Na resolução de problemas de scheduling com restrições de recursos e precedência o tamanho (isto é, a quantidade de jobs) do problema pode render métodos ótimos computacionalmente impraticáveis. Em tais casos o problema pode ser resolvido utilizando simples regras de scheduling capazes de produzir resultados racionais. Estas regras constituem os procedimentos comumente denominados de procedimentos heurísticos ou de aproximação.

Tecnicamente, os procedimentos heurísticos utilizados neste trabalho envolvem a atribuição de um grau de prioridade para cada atividade; feito isso, lista-se as atividades em ordem de prioridade segundo a qual os recursos são, então, alocados até serem exauridos. No caso de duas (ou mais) atividades tiverem mesma

prioridade, um outro critério deve ser assumido de tal forma a possibilitar uma escolha entre elas. Neste tipo de scheduling, assume-se a pré-existência da lista ordenada de atividades. Tal lista, comumente denominada de lista de prioridades, é utilizada em todos os algoritmos usados neste trabalho.

No Capítulo II, um modelo geral do problema de scheduling é descrito. Como casos particulares desse modelo, exemplos são dados visando ilustrar o uso de scheduling em problemas reais, comumente encontrados no nosso dia-a-dia.

Problemas de scheduling, por serem, em geral, combinatoriais são quase sempre intratáveis por procedimentos exatos. Tal fato tem motivado inúmeros estudiosos do assunto. A fim de se ter uma idéia das dificuldades inerentes a este tipo de problemas, é apresentada no Capítulo III uma classificação de problemas de scheduling conforme sua complexidade. Essa classificação, visa apenas identificar se um determinado problema de scheduling é ou não solucionável polinomialmente. Se sim, o problema é dito pertencer à classe  $P^1$ ; se não, o problema é dito pertencer à classe NP-completo<sup>2</sup>. Após tentar delinear um limite entre as classes  $P$  e NP-completo para problemas de scheduling, encontra-se, ainda no Capítulo III, uma implementação para o algoritmo de Hu aplicado a um problema de scheduling com uma restrição de precedência especial (estrutura de árvore). O programa correspondente (escrito em

---

(1) -  $P$  = Notação utilizada na teoria da complexidade para representar o conjunto de problemas solucionáveis polinomialmente por algoritmos exatos.

(2) - NP-completo = Termo utilizado na teoria da complexidade de problemas para representar o conjunto de problemas que não podem ser solucionáveis polinomialmente por algoritmos exatos.

Algol-w) a este algoritmo encontra-se no Apêndice.

No Capítulo IV, procura-se situar o uso de scheduling no planejamento de projetos. Como se trata de problemas em que há limitação de recursos, não se pode recorrer aos procedimentos tradicionais de PERT e CPM vez que, como se sabe, em tais procedimentos recursos são considerados em quantidade ilimitada. Ao invés, a abordagem sobre planejamento de projetos é feita usando-se network<sup>1</sup> (que define inclusive a relação de precedência entre as atividades) com limitação de recursos. Um procedimento heurístico é descrito e testado num projeto simulado. O programa correspondente é apresentado no Apêndice.

Finalmente, no Capítulo V são feitas considerações acerca do trabalho.

---

(1) - A palavra network pode ser traduzida como rede de planejamento ou rede de trabalho. Contudo, por ser amplamente utilizada nos textos em português, será aqui mantida em inglês.

## CAPÍTULO II

### UM MODELO GERAL PARA PROBLEMAS DE SCHEDULING

#### 2.1 - ESPECIFICAÇÃO DO PROBLEMA

Um problema de scheduling pode ser especificado com os dados abaixo:

Uma certa quantidade de jobs <sup>1</sup> para serem processados, existindo para cada job  $i = 1, \dots, n$  um tempo de

---

(1) - Por terem sido os primeiros estudos no campo de scheduling motivados por problemas em manufaturamento, a terminologia destes problemas é ainda comumente usada. Assim, o termo "job" é usado no sentido de atividade, tarefa, ou mesmo, um conjunto de operações.

processamento  $p_i$ .

Um ponto  $q_i$ , no tempo, no qual o job  $i$  fica disponível para ser processado, isto é, o tempo mais cedo possível. Quando não declarado assume-se  $q_i$  coincidente com o tempo de início do projeto que, geralmente, é considerado igual a zero.

Um ponto  $d_i$ , no tempo, previsto para o término do processamento do job  $i$ . No caso do tempo de processamento exceder este ponto, isto é, se ocorrer um atraso, normalmente aplica-se uma penalidade.

Tem-se ainda  $r$  tipos de recursos;  $r_{ik}$  unidades de recursos do tipo  $k$  requisitados para o processamento do job  $i$  e  $R_k$  unidades de recurso do tipo  $k$  disponíveis em cada período de tempo. Então  $R = (R_1, \dots, R_r)$  pode ser denominado vetor capacidade e  $r_i = (r_{i1}, \dots, r_{ik})$  vetor demanda do job  $i$ .

## 2.2 - RELAÇÕES DE PRECEDÊNCIA

Entre os jobs existe uma relação de precedência que pode ser descrita por um grafo direcionado  $(V, E)$ . Os jobs correspondem ao conjunto  $V$  de vértices e o conjunto de arcos  $E$  indica a relação de precedência entre os jobs, isto é, ao se escrever  $(i, j) \in E$  significará que o job  $j$  não pode ser inicializado antes do job  $i$  ter sido totalmente processado. Diz-se ainda que o job

$j$  é sucessor do job  $i$  e que o job  $i$  é predecessor do job  $j$ .

As estruturas especiais abaixo são consideradas as mais importantes relações de precedência existentes. Para cada caso, segue-se um exemplo ilustrativo:

a) Estrutura de jobs independentes:

Quando não existe relação de precedência entre os jobs, neste caso  $E = \emptyset$ .

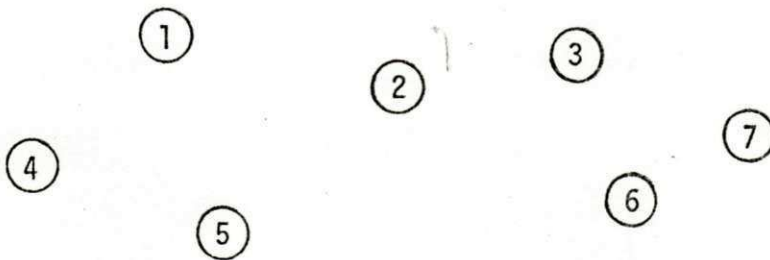


Fig. 2.1 - Estrutura de jobs independentes.

b) Estrutura de árvore "intree"

Neste caso a relação de precedência entre os jobs termina uma árvore direcionada onde cada job tem, no máximo, um sucessor direto. Um job  $i$  é também chamado predecessor direto de um job  $j$  (escreve-se  $i \ll j$ ) se não existe um job  $k$ , tal que,  $i < k < j$ .



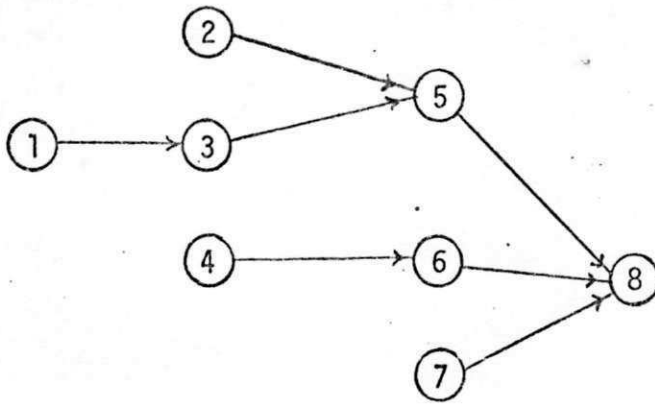


Fig. 2.2 - Estrutura "intree".

c) Estrutura de árvore "outtree"

Como na estrutura anterior, a relação de precedência entre os jobs, também determina uma árvore direcionada, só que, nesta estrutura cada job tem no máximo um predecessor direto.

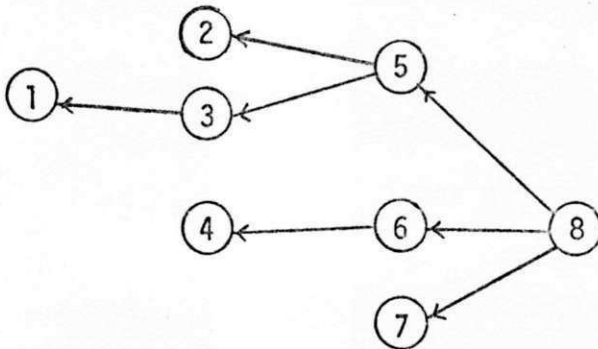


Fig. 2.3 - Estrutura "outtree".

## d) Estrutura "flow-shop"

Nesta estrutura cada job  $J_i$  ( $i = 1, \dots, n$ ) requer uma sequência específica de operações para que o mesmo seja concluído. Este tipo de estrutura, algumas vezes chamada estrutura de precedência linear, é caracterizada por um fluxo de trabalho que é unidirecional.

Existem  $m$  recursos diferentes e cada job compõe-se de  $m$  operações, cada uma das quais requerendo um recurso diferente. Neste caso tem-se recurso significando máquina. Todo job deve ser processado por todas as máquinas em uma mesma ordem, onde  $(i, j)$  significa a operação do job  $J_i$  na máquina  $M_j$  ( $j = 1, \dots, m$ ).

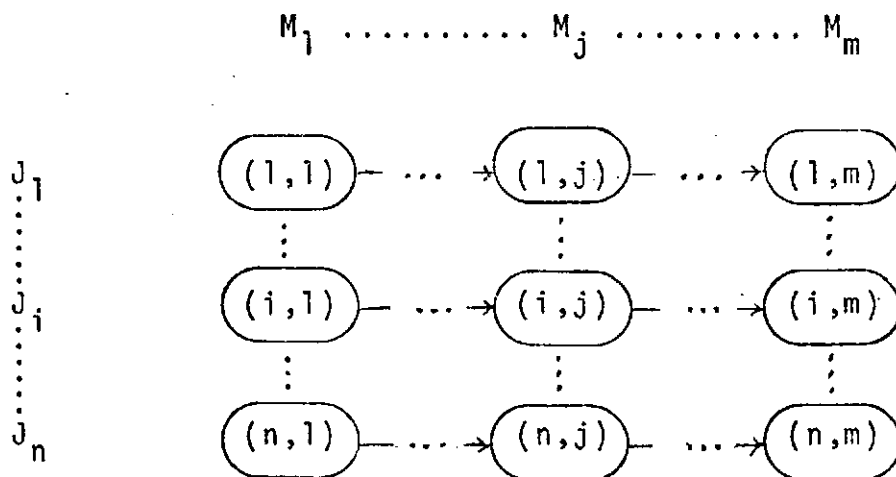


Fig. 2.4 - Estrutura "flow-shop"

## e) Estrutura "job-shop"

Como na estrutura "flow-shop", também nesta estrutura cada job requer várias operações para ser concluído. Entretanto, um job pode necessitar de menos operações que o número de máquinas

nas disponível, além disso, a ordem de utilização das máquinas pode ser diferente para cada job. Por conveniência, indica-se que uma operação  $j$  do job  $J_i$  requer uma máquina  $M_k$  pela tripla  $(i,j,k)$ .

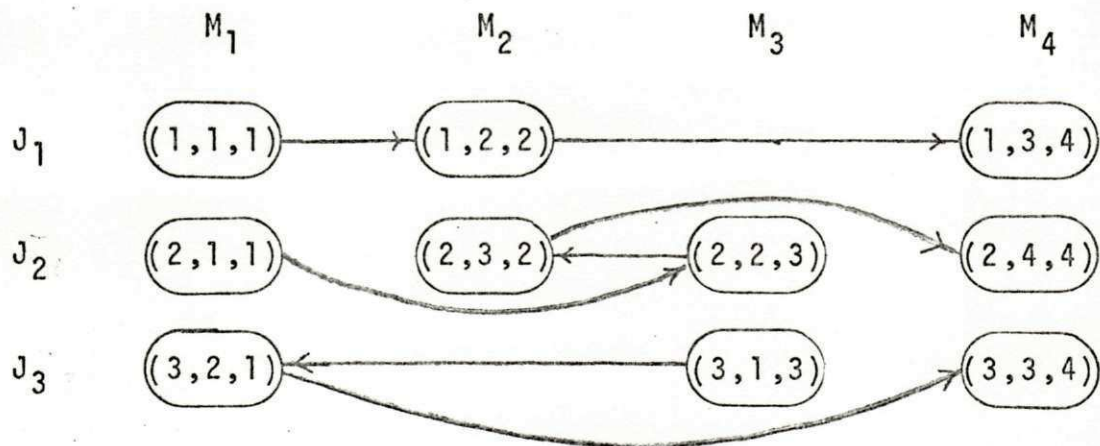


Fig. 2.5 - Estrutura "job-shop"

### 2.3 VARIÁVEIS QUE DESCREVEM A SOLUÇÃO PARA O PROBLEMA

Indicando por  $x_i$  o tempo de início do processamento do job  $i$  e assumindo que, uma vez inicializado, o mesmo será ininterruptamente processado, uma schedule é possível quando:

- 1)  $x_i + P_i \leq x_j \quad \forall (i,j) \in E$
- 2)  $\sum_{i \in N_t} r_i \leq R$  (onde  $N_t$  representa o conjunto de jobs em processamento no período de tempo  $t$ ).

Onde a expressão 1) significa que todo job  $i$  é processado em tempo menor ou igual ao tempo de início do processamento de seu sucessor  $j$ . A expressão 2) significa que o job  $i$ , ao ser

processado, não consome mais do que o total disponível de recursos do tipo  $k$ , em cada período de tempo  $t$ .

#### 2.4 - MEDIDAS QUANTITATIVAS PARA A AVALIAÇÃO DE SCHEDULES

Uma schedule pode ser avaliada de acordo com vários critérios. Medidas quantitativas para a avaliação de schedules são usualmente função dos tempos de conclusão  $C_i = x_i + p_i$  de cada job  $i$ .

Entre as mais importantes quantidades, pode-se citar, além de  $C_i$  :

$$a) D_i = C_i - d_i \quad (\text{Desvio})$$

Representa a quantidade de tempo pela qual o tempo de conclusão do job  $i$  excede, antecede ou coincide com o tempo previsto para o seu término  $d_i$ .

É importante notar que esta quantidade assume valores negativos se o job  $i$  é concluído mais cedo, o que representa melhor serviço que o requisitado. Por outro lado, quando  $D_i$  assume valores positivos estará representando um serviço pior que o requisitado. Neste último caso, geralmente, custos de penalidades estarão associados com  $D_i > 0$ , o que não ocorre com benefícios para  $D_i < 0$ . Por isso é muito útil trabalhar com uma quantidade que mede somente valores positivos de  $D_i$ .

$$b) A_i = \max \{0, D_i\} \quad (\text{Atraso})$$

Será igual a  $D_i$  se o job  $i$  não é concluído no tempo  $d_i$  previsto para o seu término; caso contrário, será igual a 0 (zero).

Existe ainda uma função característica que indicará se um determinado job atrasou ou não. Esta função é definida como:

$$c) U_i = \begin{cases} 0 & \text{se } C_i \leq d_i \\ 1 & \text{caso contrário} \end{cases}$$

## 2.5 - FUNÇÕES OBJETIVAS

O problema geral de scheduling é encontrar uma possível schedule que minimize uma função objetiva  $Z = f(x_1, \dots, x_n) = f(C_1 - p_1, \dots, C_n - p_n)$ . A seguir encontram-se algumas das funções objetivas mais importantes:

$$a) f(x_1, \dots, x_n) = \max_{i=1}^n \{C_i\} = C_{\max}$$

Tem como objetivo encontrar uma schedule onde a conclusão do último job ocorra no menor tempo possível, ou seja, minimizar o tempo de término da schedule que é medido pelo maior  $C_i$ .

$$b) f(x_1, \dots, x_n) = \sum_{i=1}^n w_i C_i$$

Nesta função,  $w_i$  representa um peso que está relacionado

nado com a importância do job  $i$ . O objetivo é encontrar uma schedule onde o somatório dos tempos de conclusão ponderados seja mínimo.

$$c) f(x_1, \dots, x_n) = \max_{i=1}^n \{D_i\} = D_{\max}$$

O objetivo aqui é encontrar uma schedule onde o maior atraso seja o mínimo possível.

$$d) f(x_1, \dots, x_n) = \sum_{i=1}^n w_i A_i$$

Aqui,  $w_i$  representa custos de penalidades que são proporcionais aos possíveis atrasos. O objetivo neste caso, é encontrar uma schedule onde a soma dos possíveis custos de penalidades seja mínima.

$$e) f(x_1, \dots, x_n) = \sum_{i=1}^n w_i U_i$$

Se  $w_i = 1$  para todo job  $i$ , então o objetivo é encontrar uma schedule onde o número de atrasos seja o mínimo possível.

Se os  $w_i$ 's são diferentes para diferentes jobs, então o objetivo será encontrar uma schedule onde o somatório dos produtos dos possíveis atrasos pelos seus respectivos custos de penalidades,  $w_i$ , seja mínimo.

Com a finalidade de classificar individualmente os

problemas de scheduling, pode ser usada a seguinte notação <sup>1</sup>:

$\alpha|\beta|\gamma|\nu|\epsilon$ , onde

$\alpha$  - representa o número de jobs.

$\beta$  - no caso de problemas com estrutura "flow-shop" ou "job-shop" representa número de recursos diferentes e, nas demais estruturas, representa quantas unidades de um único tipo de recurso estarão disponíveis em cada período de tempo <sup>2</sup>.

$\gamma$  - representada a estrutura de precedência entre os jobs do problema, podendo assumir os seguintes valores, com seus respectivos significados:

$\emptyset$  - para jobs independentes

F - para estrutura "flow-shop"

G - para estrutura "job-shop"

IT - para estrutura "intree"

OT - para estrutura "outtree"

$q_i \neq 0$  - para os tempos de início de processamento diferentes de o (zero).

---

(1) - Como referência para esta notação tem-se a publicação de Lenstra (4).

(2) - Extensões para situações mais gerais onde vários grupos de recursos (possivelmente não idênticos) estão disponíveis em paralelo não serão considerados até o capítulo IV. Naquele capítulo, quando é abordado o problema de scheduling em projetos com recursos limitados,  $\beta$  representará também o número de recursos diferentes. Nesse caso, a disponibilidade desses recursos no tempo deverá aparecer de maneira explícita.

$v$  - representa restrições com respeito a  $p_j$ , isto é, aos tempos de processamento do job, por exemplo:

Se  $p_j = 1$ : todos os jobs tem o mesmo tempo de processamento igual a 1.

Se  $p_j \in \{1,2\}$ : os jobs terão tempo de processamento possivelmente igual a 1 ou igual a 2.

$\epsilon$  - representa a função objetiva.

A inexistência de restrições relativas a qualquer das variáveis acima indica-se por um ponto, ".", na notação, no lugar referente àquela variável. Dessa forma, um problema geral de scheduling pode ser representado por:

.|.|.|.|\epsilon

## 2.6 - APLICAÇÕES

Com o propósito de ilustrar, de maneira geral, o comportamento de variáveis em um problema de scheduling como também o enquadramento de diversas situações reais nesse tipo de problema, são apresentados, a seguir, diversos exemplos. As soluções obtidas para os exemplos considerados não são necessariamente ótimas apesar de estar declarado em cada exemplo que a função deva ser minimizada.



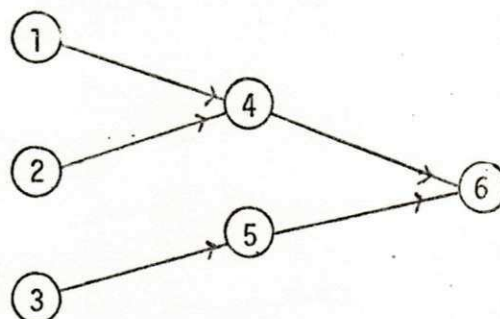
1) Seja o problema de processar um número arbitrário de jobs formando uma estrutura de precedência "intree", com cada job requerendo um tempo arbitrário de processamento. Considere ainda, que existe apenas uma unidade de um certo tipo de recurso disponível em cada período de tempo, e que o objetivo do problema é encontrar uma schedule onde o somatório dos tempos de conclusão ponderados seja mínimo.

Este problema pode ser declarado da seguinte maneira:

$$\min \sum_i w_i C_i$$

Ilustração: são dados

job $i$	$p_i$	$w_i$
1	1	40
2	2	50
3	3	0
4	2	100
5	2	10
6	1	20

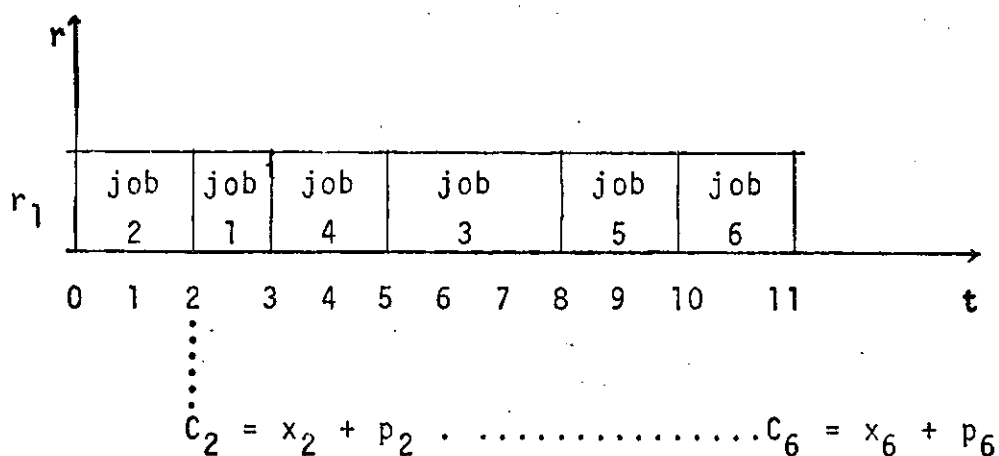


Estrutura de precedência entre os jobs.

Onde job  $i$ ,  $p_i$  e  $w_i$  referem-se, respectivamente, ao número do job, seu tempo de processamento e o peso associado a

importância do job  $i$ .

Pode-se construir uma possível schedule utilizando um diagrama de Gantt, o qual consiste de um sistema de eixos cartesianos, onde recursos disponíveis são mostrados ao longo do eixo vertical e uma escala de tempo ao longo do eixo horizontal:



Na schedule apresentada objetivou-se conseguir  $\sum_{i=1}^6 w_i \cdot C_i$  com o menor valor possível. Para isso, os jobs com maiores pesos devem ser processados primeiro obedecendo-se, entretanto, a relação de precedência. O resultado obtido é:

$$\sum_{i=1}^6 w_i \cdot C_i = 40 \cdot 3 + 50 \cdot 2 + 0 \cdot 8 + 100 \cdot 5 + 10 \cdot 10 + 20 \cdot 11 = 1040$$

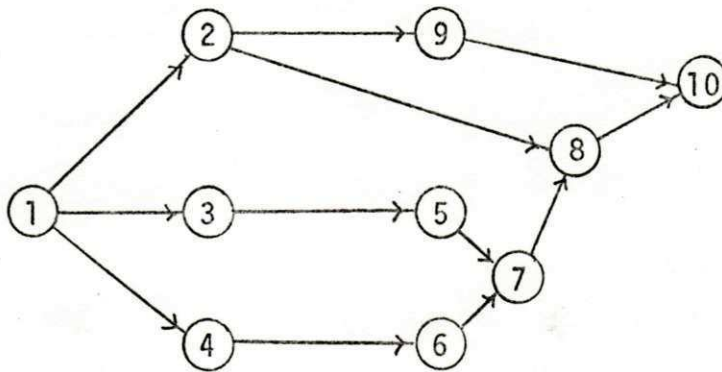
2) Considere um número arbitrário de jobs existindo entre si uma estrutura de precedência também arbitrária. Todos os jobs requerem tempos de processamento iguais a 1, ou seja,  $p_i = 1$ . Existem duas unidades de um tipo de recurso disponível em cada

período de tempo, e o objetivo é encontrar uma schedule onde a conclusão do último job ocorra no menor tempo possível.

Este problema pode ser declarado da seguinte maneira:

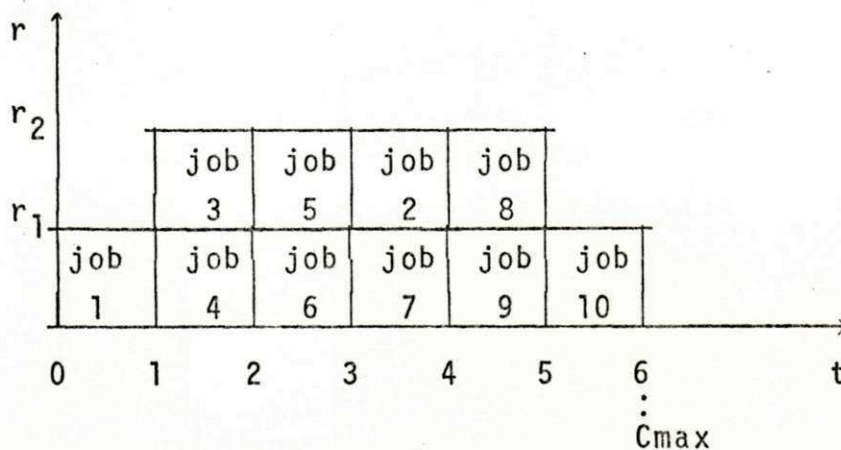
$$|2| \cdot |p_i=1| C_{max}$$

Ilustração: são dados 10 jobs com  $p_i=1$  ( $i = 1, \dots, 10$ ) com a seguinte estrutura de precedência:



Estrutura qualquer formada pelos jobs.

Uma possível schedule é:



Com a schedule acima, o mínimo de  $f(x_1, \dots, x_{10}) =$   
 $= C_{max}$  será igual a 6 unidades de tempo. Verifica-se, observando  
 o grafo, que este valor realmente é o mínimo da função, visto que,  
 existe pelo menos um caminho (1,4,6,7,8,10) que não pode ser pro-  
 cessado em tempo menor.

Uma aplicação prática desse tipo de problema ocorre,  
 por exemplo, em planejamento de projetos, onde tem-se jobs signi-  
 ficando atividades ou tarefas e recursos significando homens, m $\bar{a}$ -  
 quinas, material de construção, etc. Este tipo de aplicação é tra-  
 tado, especificamente no Capítulo IV.

3) Um conjunto de jobs formando uma estrutura de pre-  
 cedência "flow-shop" deve ser processado em duas máquinas disponí-  
 veis em cada período de tempo. Esses jobs requerem tempos de pro-  
 cessamento arbitrário, e o objetivo do problema é conseguir uma  
 schedule onde o maior atraso seja o mínimo possível. Este proble-  
 ma pode ser declarado da seguinte maneira:

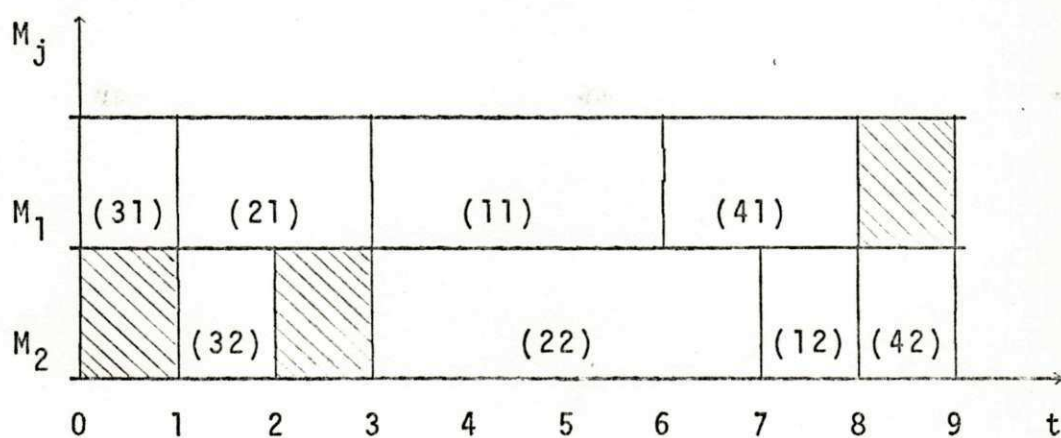
$$. |2|F|. |D_{max}$$

Ilustração: são dados

job i	$M_1$	$M_2$	$d_i$
1	3	1	7
2	2	4	5
3	1	1	2
4	2	1	11

Onde  $M_1$  e  $M_2$  representam máquinas disponíveis e  $d_i$  o tempo previsto para o término de processamento do job  $i$ .

No diagrama abaixo é mostrada uma possível schedule; as partes sombreadas representam máquinas ociosas.



$$C_3 = 2$$

$$d_3 = 2$$

$$D_3 = 0$$

$$C_2 = 7 \quad C_1 = 8 \quad C_4 = 9$$

$$d_2 = 5 \quad d_1 = 7 \quad d_4 = 11$$

$$D_2 = 2 \quad D_1 = 1 \quad D_4 = -2$$

$$f(x_1, x_2, x_3, x_4) = \max_{i=1}^4 \{D_i\} = \max\{1, 2, 0, -2\} = 2 = D_{\max}$$

Procurou-se construir a schedule observando os  $d_i$ 's, isto é, deu-se prioridade na scheduling para os jobs de menores  $d_i$ 's. Conseguiu-se como uma possível solução para o problema  $D_{max} = 2$  unidades de tempo.

Uma aplicação prática de problemas de scheduling com estas características ocorre em linhas de montagem. Por exemplo, em uma fábrica de automóveis (shop) a montagem de cada carro (job) requer que o mesmo seja processado por um certo conjunto de máquinas numa sequência específica. Esta sequência na utilização das máquinas deve ser observada para todo carro produzido.

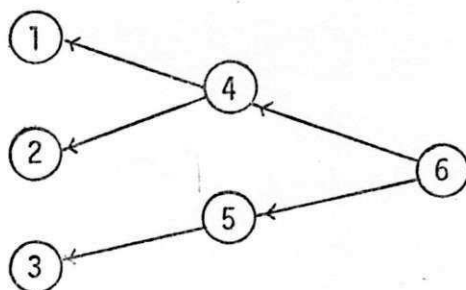
4) Formando uma estrutura de precedência "outtree", um número arbitrário de jobs deve ser processado utilizando-se um único tipo de recurso disponível. Existem duas unidades deste recurso por período de tempo. O objetivo do problema é encontrar uma schedule onde a soma dos produtos dos possíveis atrasos ( $A_i$ ) pelos custos de penalidades ( $w_i$ ) seja mínima.

Este problema pode ser declarado como

$$\min \sum_i w_i A_i$$

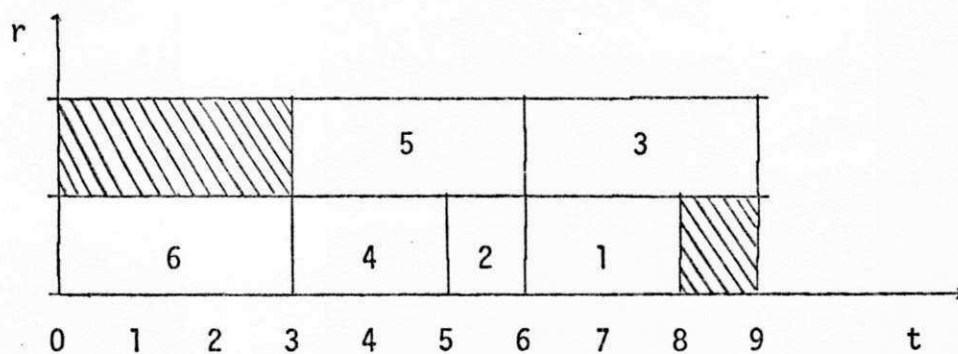
Ilustração: são dados

job $i$	1	2	3	4	5	6
$p_i$	2	1	3	2	3	3
$d_i$	9	8	7	5	7	3
$w_i$	1	3	6	5	8	9



Estrutura de precedência dos jobs.

Uma possível schedule  $\bar{e}$ :



E para esta schedule tem-se os resultados na tabela seguinte:

job i	1	2	3	4	5	6	
$C_i$	8	6	9	5	6	3	
$D_i$	-1	-2	2	0	-1	0	
$A_i$	0	0	2	0	0	0	$\sum w_i A_i$
$w_i A_i$	0	0	12	0	0	0	12

A estrutura de precedência apresentada nesse problema é típica de problemas de classificação. Uma aplicação prática seria uma agência de correios onde são classificados milhares de CEPs (Código de Endereçamento Postal). O destino específico de cada correspondência é obtido através de classificações progressivas, refinadas em vários passos. Esta progressão e refinamento podem ser denotadas por uma estrutura de árvore.

5) Cinco jobs formando entre si uma estrutura de precedência job-shop devem ser processados em duas máquinas disponíveis. Cada job requer tempo arbitrário para processamento e o objetivo é conseguir uma schedule onde o número de atrasos seja o menor possível. O problema pode ser declarado como:

$$5|2|G|.\sum_i U_i$$

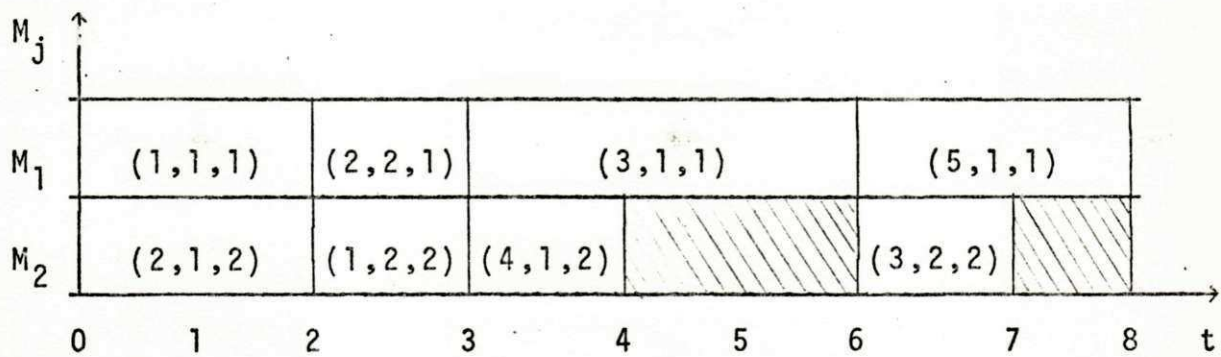
Ilustração: são dados



job $J_i$	$M_1$	$M_2$	$d_i$
1	2	1	3
2	1	2	3
3	3	1	6
4	0	1	5
5	2	0	9

$$w_i = 1 \quad \forall i = 1, 2, 3, 4, 5$$

Uma possível schedule  $\bar{e}$ :



onde temos:

job i	$C_i$	$d_i$	$U_i$
1	3	$\leq 3$	0
2	3	$\leq 3$	0
3	7	$> 6$	1
4	4	$\leq 5$	0
5	8	$\leq 9$	0

Assim sendo, obtemos  $\sum U_i = 1$

Uma aplicação prática de problemas de scheduling com estas características seria o caso de uma grande oficina de consertos de automóveis (shop), que aceita tipos diferentes de carros (jobs), requerendo diferentes tipos de reparos (operações). Observa-se, nesse caso, que o conserto de cada carro poderá requerer um ou vários reparos diferentes em ordens diferentes.

## CAPÍTULO III

### COMPLEXIDADE DE PROBLEMAS DE SCHEDULING

#### 3.1 - ALGUNS ASPECTOS DA COMPLEXIDADE DE UM PROBLEMA

Existem importantes trabalhos examinando a complexidade de um problema. Esses trabalhos identificam problemas para os quais o esforço computacional para resolvê-los é limitado polinomialmente, isto é, existem algoritmos para resolvê-los cujo número de passos é limitado por um polinômio. A complexidade de um algoritmo que resolve um dado problema irá se referir somente ao seu tempo de execução expresso como uma função do principal parâmetro do problema. Especificamente este parâmetro irá sempre ser o número  $n$  de jobs ou  $m$  de máquinas. Assim, usando a notação de ordem de magnitude  $O(.)$  pode-se dizer que um algoritmo tem complexidade  $O(n^2)$  quando existe uma constante  $c$  tal que a função  $cn^2$

limita o tempo de execução como uma função de  $n$ . Todo algoritmo cuja complexidade é limitada por um polinômio é denominado algoritmo polinomial, ou um algoritmo que processe em tempo polinomial. O problema correspondente é dito ser solucionável polinomialmente.

Vários problemas combinatoriais, tais como o de menores caminhos, de fluxo em redes de planejamento e alguns problemas especiais de scheduling são solucionáveis utilizando-se algoritmos polinomiais. Tais problemas podem ser resolvidos eficientemente por um computador. Quando um problema é desse tipo diz-se que o mesmo é polinomialmente solucionável e irá pertencer a uma classe comumente anotada por  $P$ . Então,  $P$  é um conjunto de problemas para os quais, existem algoritmos polinomialmente limitados para resolvê-los.

Existe um conjunto de problemas, cada um dos quais chamado NP-completo, que inclui muitos problemas clássicos como o problema do caixeiro viajante, o de encontrar o número cromático de um grafo, etc. Em termos de complexidade, todos os problemas NP-completos são equivalentes no seguinte sentido: se for possível encontrar um algoritmo polinomial para resolver um desses problemas, então é possível encontrar um algoritmo polinomial para qualquer um outro problema da classe, de acordo com um procedimento que normalmente varia de um problema para outro. Nos problemas citados acima os polinômios seriam em termos do número de cidades para o primeiro e o número de vértices no grafo para o segundo. Assim, ou existem algoritmos polinomiais para todos os pro

blemas NP-completos, ou nenhum deles tem tal algoritmo; qual das duas afirmações é a correta, não se sabe. O que se pode afirmar é que até o momento os estudiosos do assunto têm falhado para saírem desse impasse.

Do ponto de vista prático, é de extrema importância a prévia identificação de um dado problema como pertencente, ou não, à classe NP-completo: primeiro porque já exclui, de imediato, a viabilidade de uma busca por algoritmos de otimização que possam ser utilizados na resolução do problema e, segundo, como consequência, porque sugere uma alternativa de solução como, por exemplo, uma abordagem através de algoritmos heurísticos.

Obviamente a discussão acima é muito informal. Deve-se proceder, mais formalmente, definindo uma base matemática para representação de algoritmos e exemplos de problemas, definindo a complexidade de um problema em termos do modelo e definindo a transformação (ou redução) de um problema para outro. Com tal formalismo pode-se encontrar uma excelente abordagem em (1).

### 3.2 - COMPLEXIDADE DE ALGUNS PROBLEMAS DE SCHEDULING

A complexidade de alguns problemas de scheduling é ainda uma questão em aberto, isto é, tem-se falhado em encontrar algoritmos polinomialmente limitados para resolvê-los. Além disso ainda não se conseguiu mostrar que tais problemas sejam efetivamente NP-completos. Lenstra et al. (3,4) abordam o problema da determinação de uma linha limite entre problemas de scheduling com

respeito à sua complexidade. Nestes trabalhos, através de problemas conhecidos da pesquisa operacional e que são provados pertencerem a classe NP-completo, mostra-se, por meio de técnicas da teoria da complexidade, que muitos problemas de scheduling também pertencem a esta classe.

A seguir, serão listados alguns resultados conhecidos sobre a complexidade de alguns problemas de scheduling empregando, para simplificação, o termo máquina ao invés de recurso. Primeiramente serão abordados problemas onde uma máquina está disponível em cada período de tempo para o processamento dos jobs, depois problemas onde temos múltiplas máquinas disponíveis:

1 - Com uma única máquina disponível

a)  $\alpha|1|\gamma|\nu|C_{max}$  e  $\alpha|1|\gamma|\nu|\sum w_i C_i$

O problema  $.|1|.|.|C_{max}$  é solucionável por um algoritmo de complexidade  $O(n^2)$  atribuído a Lawler (5), onde ele utiliza uma função de custos arbitrária não decrescente, mostrando que este problema é trivial porque qualquer schedule é ótima.

Lenstra e Rinnooy Kan (3) mostram que os problemas  $.|1|.|p_i \in \{0,1\}|\sum C_i$  e  $.|1|.|p_i = 1|\sum w_i C_i$  são NP-completos.

Quando não existe restrições com respeito aos tempos de processamento dos jobs, os quais formam uma estrutura de jobs independentes, Smith (6) mostrou que o problema  $.|.|\phi|.|\sum w_i C_i$  pode ser resolvido ordenando os jobs em ordem não decrescente de acordo com a razão  $p_i/w_i$ . Depois Horn e Sidney (7) desenvolveram um algoritmo de complexidade  $O(n \log n)$  extendendo este resultado

para o problema onde os jobs formam uma estrutura de árvore; ou seja, para os problemas  $|1|IT|.\Sigma w_i C_i$  e  $|1|OT|.\Sigma w_i C_i$ .

$$b) \alpha|1|\gamma|v|\Sigma w_i T_i \text{ e } \alpha|1|\gamma|v|\Sigma w_i U_i$$

Normalmente, problemas de scheduling com estes critérios são bastante difíceis. Quando não existem restrições com respeito aos tempos de processamento com os jobs formando, entre si, uma estrutura de jobs independentes, o problema  $|1|\phi|.\Sigma w_i T_i$  é mostrado ser NP-completo (4). Nestas mesmas condições, o problema  $|1|\phi|.\Sigma w_i U_i$  também foi mostrado ser NP-completo por Karp (8).

Com os custos de penalidades unitários, Moore (9) construiu um algoritmo que resolve o problema  $|1|\phi|.\Sigma U_i$  em  $O(n \log n)$  e Lawler (10) construiu um que resolve o problema  $|1|\phi|.\Sigma T_i$  em  $O(n^4 \Sigma p_i)$  entretanto, este último é considerado um problema "aberto", isto é, não se pode garantir que esteja na classe P nem na classe NP-completo, pois a expressão  $n^4 \Sigma p_i$ , que é o grau de complexidade do algoritmo, para ser um polinômio ou não, depende do fator  $\Sigma p_i$ .

No caso de tempos de processamento unitário Lawler (10) mostrou que o problema  $|1|\phi|1|\Sigma w_i U_i$  é solucionável pelo algoritmo de Moore. O problema  $|1|\phi|1|\Sigma w_i T_i$  pode ser visto como um problema de designação linear, e portanto, solucionável em  $O(n^3)$  passos. Mesmo com os tempos de processamento e custos de penalidade unitários, a introdução de estruturas gerais de precedência transforma esses dois últimos problemas em NP-completos (6,11).

## 2 - Com múltiplas máquinas disponíveis

Os problemas  $.|2|\phi|.|C_{max}$  e  $.|2|\phi|.|\Sigma w_i C_i$  são mostrados serem NP-completos em (4,11)

Coffman Jr. e Graham (12) desenvolveram um algoritmo de  $O(n^2)$  que resolve o problema  $.|2|.|p_i = 1|C_{max}$  e que também produz uma schedule ótima para o problema  $.|2|.|p_i = 1|\Sigma C_i$ .

O problema  $.|2|.|p_i \in \{1,2\}|C_{max}$  foi provado ser NP-completo por Ullman (13) em um dos principais papers sobre complexidade de problemas de scheduling. Lenstra e Rinnooy Kan (3) determinam que o problema  $.|2|.|p_i \in \{1,2\}|\Sigma C_i$  é NP-completo. Com  $w_i = 1$  os problemas  $.|2|F|.|\Sigma T_i$  e  $.|2|F|.|\Sigma U_i$  são mostrados serem NP-completos ao lado do problema  $.|2|F|.|D_{max}$  em (4).

Os problemas  $.|m|I^T|p_i = 1|C_{max}$  e  $.|m|O^T|p_i = 1|C_{max}$ , ou seja, os problemas de scheduling de jobs com iguais tempos de processamento em m máquinas idênticas sujeitos às restrições "intree" ou "outtree" e com o objetivo de minimizar o tempo de conclusão do último job, podem ser resolvidos por um algoritmo polinomial atribuído a Hu (14). Este algoritmo é descrito na seção seguinte.

Os problemas  $.|m|.|p_i = 1|C_{max}$  e  $.|m|.|p_i = 1|\Sigma C_i$  são mostrados serem NP-completos por Ullman (13) e por Lenstra (3) respectivamente.

Na classificação apresentada na Tabela 3.1 são representados casos mais simples de NP-completos. Assim ao serem tomadas as inferências apropriadas com respeito a problemas mais gerais,



obviamente serão produzidos problemas pelo menos tão difíceis quanto os originais.

Tabela 3.1 - Complexidade de Problemas de Scheduling

solucionáveis polinomialmente	NP-completos
.   $q_i \neq 0$   $C_{max}$	.   1   $p_i = 1$   $\sum w_i C_i$
.   1   $I$   $\sum w_i C_i$	.   1   $q_i \neq 0, \Phi$   $\sum C_i$
.   1   $OT$   $\sum w_i C_i$	.   1   $\sum C_i$
.   1   $D_{max}$	.   1   $q_i \neq 0, \Phi$   $D_{max}$
.   1   $q_i \neq 0$   $p_i = 1$   $D_{max}$	
.   1   $q_i \neq 0, \Phi$   $p_i = 1$   $\sum w_i T_i$	.   1   $\Phi$   $\sum w_i T_i$
.   1   $p_i = 1$   $\sum T_i$	
.   1   $q_i = 0, \Phi$   $p_i = 1$   $\sum w_i U_i$	.   1   $\Phi$   $\sum w_i U_i$
.   1   $\Phi$   $\sum U_i$	.   1   $p_i = 1$   $\sum U_i$
.   2   $F$   $C_{max}$	.   2   $q_i \neq 0, F$   $C_{max}$
2   $F$   $C_{max}$	.   3   $F$   $C_{max}$
2   $G$   $C_{max}$	.   2   $F$   $\sum C_i$
	.   $F$   $\sum C_i$
	.   2   $F$   $D_{max}$
	.   2   $F$   $\sum T_i$
	.   2   $F$   $\sum U_i$

Continuação da Tabela 3.1

Solucionáveis polinomialmente	NP-completos
. 2 .   $p_i = 1$   Cmax	. 2 .   $p_i \in \{1,2\}$   Cmax . 2  $\Phi$  .   Cmax
. 2 .   $p_i = 1$   $\Sigma C_i$	. 2 .   $p_i \in \{1,2\}$   $\Sigma C_i$ . 2  $\Phi$  .   $\Sigma w_i C_i$ . 2  $\Phi$  .   Dmax . 2  $\Phi$  .   $\Sigma T_i$ . 2  $\Phi$  .   $\Sigma U_i$
. .  OT  $p_i = 1$   Cmax . .   $\Phi$  .   $\Sigma C_i$ . .   $q_i \neq 0, \Phi$   $p_i = 1$   $\Sigma w_i C_i$ . .  IT  $p_i = 1$   Dmax	. .   $p_i = 1$   Cmax . .  F .   $\Sigma C_i$ . .   $p_i = 1$   $\Sigma C_i$ . .  OT  $p_i = 1$   Dmax

Obs: A construção desta tabela foi feita baseando-se em (4).

A Tabela 3.2, contém para muitos destes problemas que são solucionáveis em tempo limitado polinomialmente, referências onde pode ser encontrado o algoritmo em questão e contém ainda a ordem do número de passos nas melhores implementações.

Tabela 3.2 - Referências para algoritmos polinomialmente limitados.

Problemas	Referências	Ordem
$\cdot  1 q_i \neq 0  \cdot  C_{max}$		$O(n^2)$
$\cdot  1 OT  \cdot  \sum w_i C_i$	W.E.Smith (6)	$O(n \log n)$
$\cdot  1 IT  \cdot  \sum w_i C_i$	W.E.Smith (6)	$O(n \log n)$
$\cdot  1  \cdot   \cdot  D_{max}$	E.L.Lawler (5)	$O(n^2)$
$\cdot  1 q_i \neq 0 p_i = 1 D_{max}$	B.J.Lageweg, J.K.Lenstra e A.H.G.Rinnooy Kan (15)	$O(n^2)$
$\cdot  1 q_i \neq 0, \phi p_i = 1 \sum w_i T_i$	E.L.Lawler (16)	$O(n^3)$
$\cdot  1 q_i \neq 0, \phi p_i = 1 \sum w_i U_i$	B.J.Lageweg e E.L.Lawler (17)	$O(n^2)$
$\cdot  1 \phi  \cdot  \sum U_i$	J.M.Moore (9)	$O(n \log n)$
$\cdot  2 F  \cdot  C_{max}$	S.M.Johnson (18)	$O(n \log n)$
$\cdot  2  \cdot  p_i = 1 \sum C_i$	E.G.Coffman, R.L.Graham (12)	$O(n^2)$
$\cdot   \cdot  G  \cdot  C_{max}$	G.L.Nemhanser, W.Szware e W.W.Hardgrave (19)	$O(m^2)$
$\cdot   \cdot  IT p_i = 1 C_{max}$	T.C. Hu (14)	$O(n)$
$\cdot   \cdot  OT p_i = 1 C_{max}$	T.C. Hu (14)	$O(n)$
$\cdot   \cdot  q_i \neq 0, \phi p_i = 1 \sum w_i C_i$	E.L.Lawler (16)	$O(n^2)$
$\cdot   \cdot  \phi  \cdot  \sum C_i$	R.W.Conway, M.L.Maxwell e L.W.Miller (20)	$O(n \log n)$

Obs: Nesta tabela, na coluna de ordem,  $m$  está associado ao número de máquinas e  $n$  está associado ao número de jobs. A construção desta tabela foi baseada em (4).

### 3.3 - O ALGORITMO DE HU: EXATO PARA RESOLVER UM PROBLEMA DA CLASSE P E HEURÍSTICO PARA UM DA CLASSE NP-COMPLETO.

O algoritmo que será apresentado é de otimização, e, é utilizado para resolver problemas de scheduling com máquinas paralelas e jobs com iguais tempos de processamento, formando uma estruturaintree. Se nestes problemas os jobs requererem diferentes tempos para o processamento, o que faz destes problemas elementos da classe NP-completo, então o algoritmo, apresentado inicialmente exato, é modificado para resolvê-los heurísticamente.

Um problema de scheduling envolvendo jobs dependentes, com tempos de processamento iguais em processadores (máquinas) paralelos, de modo que o tempo de conclusão do último job seja no menor tempo possível, pode ser considerado bem mais difícil do que com jobs independentes.

Para o problema  $||IT|p_i = 1|C_{max}$ , que é um caso simples, Hu (14) apresenta um algoritmo polinomial para resolvê-lo. Este algoritmo, contém basicamente duas fases; uma de rotulagem e outra de scheduling. Na fase de rotulagem coloca-se o rótulo zero para o job terminal ou raiz e, para cada job  $i$ , identifica o único sucessor direto  $k$  e lhe coloca o rótulo  $\alpha_i = \alpha_k + 1$ . Na segunda fase constroi-se a schedule utilizando-se de uma lista, construída ordenadamente observando as restrições de precedência e de máquinas.

A descrição deste algoritmo está em uma pseudo-lingua

gem e pode ser facilmente transformada em um programa para o computador.

Na representação do algoritmo são utilizadas as seguintes variáveis:

- n            número de jobs
- m            número de máquinas
- L            lista construída, onde os jobs aparecem em ordem não decrescente de seus rótulos.
- $L_j$         lista dos jobs que foram processados em cada máquina  $j = 1, \dots, m$ .
- $t_j$         tempo de conclusão corrente, dos últimos jobs em cada máquina.
- T            tempo total requerido para a scheduling.
- rot(i)      array onde serão colocados os rótulos do job  $i = 1, \dots, n$ .
- pred(i)     array onde estarão os números de predecessores diretos de cada job  $i = 1, \dots, n$ .
- sucs(i)     array que contém o sucessor direto de cada job  $i = 1, \dots, n$ .
- tproc       quantidade de tempo requisitado para o processamento de qualquer job. Nesta primeira aplicação, temos  $tproc = 1$  para qualquer job  $i = 1, \dots, n$ .
- INS         procedure para inserir os jobs em uma lista L em ordem não decrescente de seus rótulos.

## Algoritmo 1

Passo 1. (inicialização)

```

L ←  $\emptyset$  ;
for j ← 1 to m step 1 do
  begin
    Lj ←  $\emptyset$  ;
    tj ← 0 ;
  end;

```

Passo 2. (rotulagem e construção de uma lista dos jobs, em ordem não decrescente, em relação aos seus rótulos).

```

for i ← 1 to n step 1 do
  begin
    if o job i é a raiz then
      rot(i) ← 0
    else
      begin
        identificar o único k para o qual  $i << k$ ;
        rot(i) ← rot(k) + 1;
      end;
    Comment chama procedure INS;
    INS(L,i);
  end;

```

Passo 3. (scheduling dos jobs)

```

While      L ≠ ∅      do
    begin
        for j ← 1 to m step 1 do
            if existe i em L, tal que, pred(i)=0 Then
                begin
                    escolher o primeiro job i da lista;
                    L ← L \ {i};
                    Lj ← Lj U {i};
                    tj ← tj + tproc;
                end;
    end;

```

Passo 4. (Encontrar o tempo de conclusão da schedule)

$$T \leftarrow \max_{1 \leq j \leq m} \{t_j\};$$

Ullman (13) mostra que a generalização do problema  $||IT||p_i = 1||C_{max}$ , permitindo restrições de precedência arbitrária acíclica, é NP-completo, portanto, provavelmente não pode ser resolvido, com um algoritmo eficiente. Todavia, Brucker, Garey e Johnson (21) fazem uma generalização alternativa do problema de Hu, onde cada job tem um específico  $d_j$  e, com objetivo de encontrar uma schedule que minimize o maior atraso, apresentam um algoritmo eficiente para resolver esta generalização.

Quando os tempos de processamento são diferentes para os jobs, pode-se utilizar o mesmo Algoritmo 1, modificando apenas o passo 2 para resolver o problema encontrando uma solução heurística. A modificação seria:

Passo 2. (rotulagem e construção de uma lista dos jobs em ordem não decrescente, em relação aos seus rótulos).

```

for i ← 1 do n step 1 do
  begin
    if job i é a raiz then
      rot(i) ← tproc(i)
    else
      begin
        identificar o único k, tal que, i << k;
        rot(i) ← rot(k) + tproc(i);
      end;
      Comment chama procedure INS;
      INS(L,i);
    end;

```

Neste passo a variável  $tproc(i)$  é um array contendo os diferentes tempos de processamento dos jobs.

Segue uma aplicação do algoritmo de Hu, primeiramente para um problema onde os jobs formam uma estrutura "intree" e re que rem tempos iguais de processamento. Portanto para este exem



plo, como já declarado anteriormente, o algoritmo é exato e foi implementado para um problema com os dados da Tabela (3.3) e tendo disponíveis 3 máquinas iguais em paralelo. Todo job requer 1 unidade de tempo para o processamento. Nesta tabela, na primeira linha se encontram os números dos jobs, na segunda o sucessor direto, e, na terceira o número de predecessores diretos de cada job.

---

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
3	6	9	9	11	11	11	13	14	16	16	19	19	21	21	22	23	23	23	23	24	24	24	0
0	0	1	0	0	1	0	0	2	0	3	0	1	1	0	2	0	0	2	0	2	1	4	3

---

Tabela 3.3

O algoritmo é ilustrado nas Figuras 3.1a e 3.1b

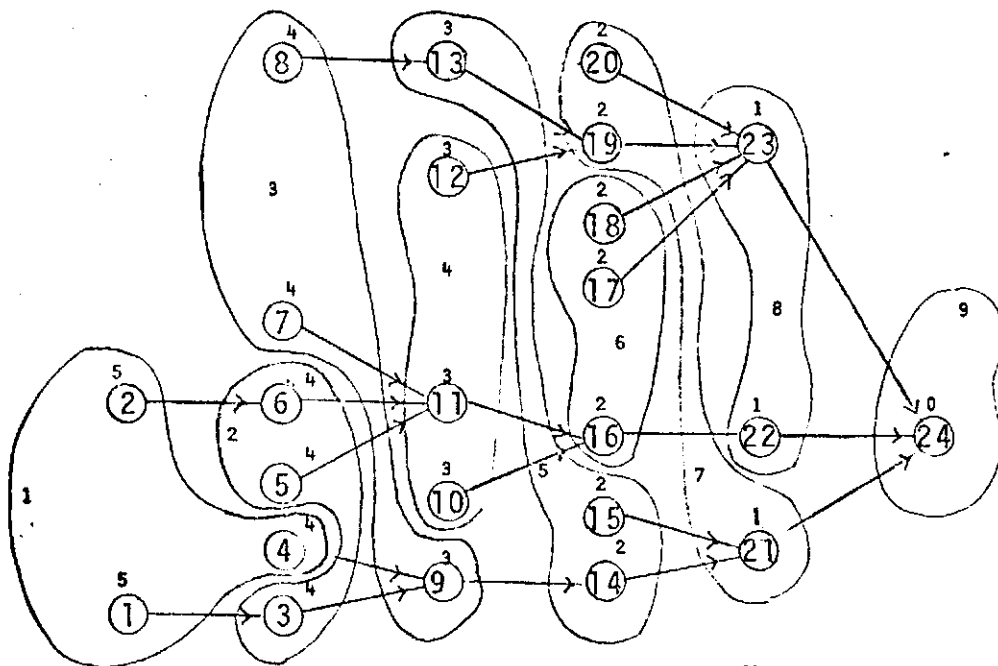


Fig. 3.1a - Representação da estrutura de precedência dos 24 jobs do problema.

Os números colocados acima de cada job representam seus respectivos rótulos. Os contornos representam os jobs que devem ser processados conjuntamente e estão numerados de acordo com o intervalo de tempo correspondente.

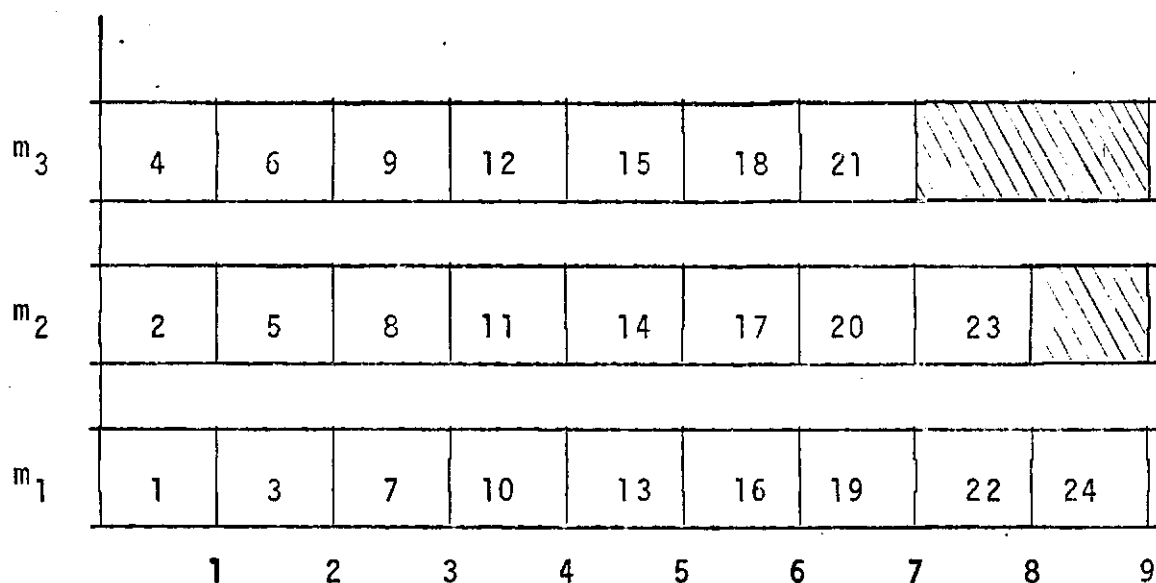


Fig. 3.1b - Schedule resultante da aplicação do algoritmo de Hu representada por Gantt.

São apresentados a seguir os resultados conseguidos com a implementação do algoritmo de Hu para um problema com os mesmos dados da Tabela (3.3) e com 3 máquinas à disposição. Neste caso, os jobs requerem diferentes tempos para o processamento, ou seja:

job i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
p <sub>i</sub>	3	4	1	2	3	1	1	2	1	4	1	3	2	2	1	3	1	2	1	5	1	1	2	1

Com estes dados o tempo gasto para a scheduling de todos os jobs foi de 19 unidades de tempo.

Supondo que estes jobs possam ser processados em partes, isto é, depois de inicializado o processamento de um certo job, este pode ser interrompido e voltar a ser processado mais tarde, até a sua conclusão, obtem-se um limite inferior de tempos para a conclusão de todos os jobs. Feita a implementação nestas condições consegue-se um limite inferior de 17 unidades de tempo. Com isso foi gerado um intervalo, razoavelmente, pequeno para a localização da solução ótima, ou seja:

$$17 \leq \text{solução ótima} \leq 19$$

Portanto a solução heurística é uma "boa" resposta para o problema.

Foi testado ainda o mesmo problema, agora com os diferentes tempos, requeridos para o processamento dos jobs, assumindo valores em uma variação um pouco maior, ou seja:

job i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
p <sub>i</sub>	1	2	2	2	3	1	3	1	6	5	1	1	3	2	9	1	2	6	1	1	1	6	2	1

Na implementação do algoritmo heurístico conseguiu-se

uma solução de 29 unidades de tempo total para o processamento .  
 Aplicando o algoritmo exato, para o mesmo problema, com os jobs  
 tendo direito à interrupção, obteve-se um limite inferior de 25  
 unidades de tempo (ver fig. 3.2). Ficou determinado com isso um  
 intervalo para a localização do ótimo, ou seja:

$$25 \leq \text{solução ótima} \leq 29$$

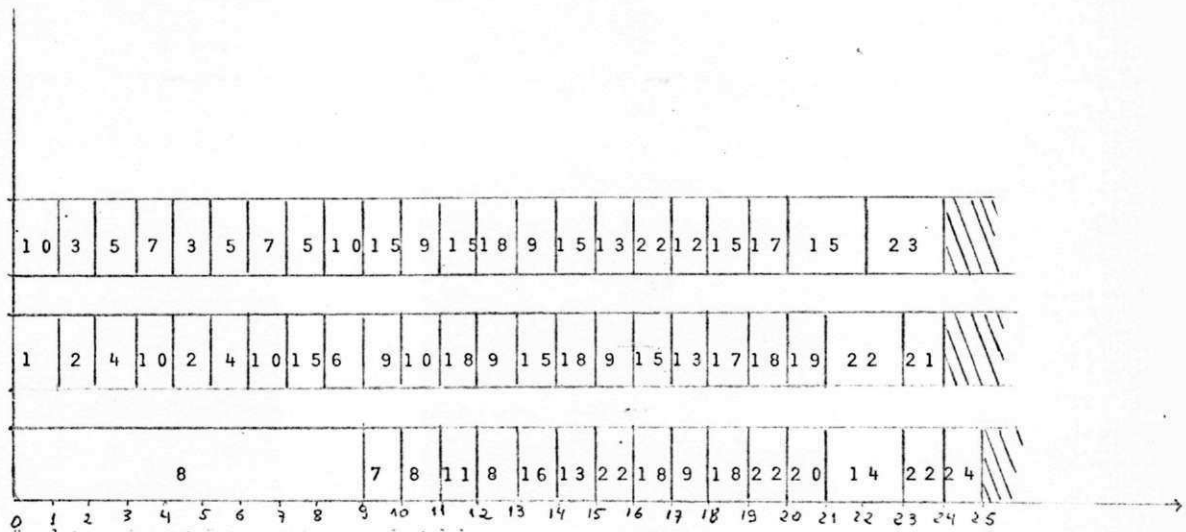


Fig. 3.2a: - Schedule construída com os  
 jobs tendo direito a interrup  
 ção.

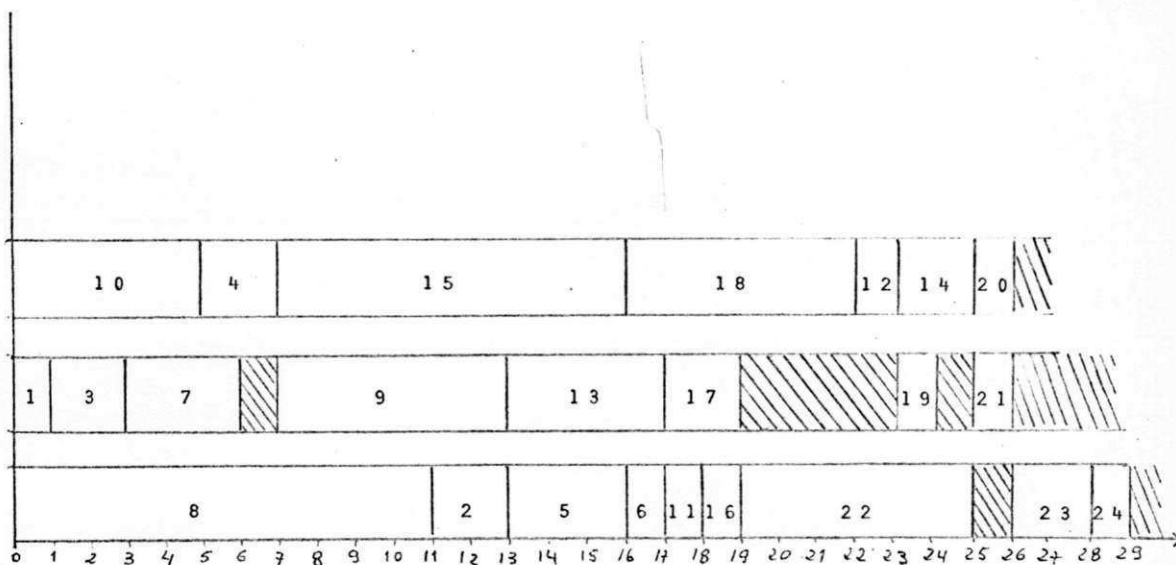


Fig. 3.2b: - Schedule construída com os jobs respeitando a estrutura de precedência definida na Tabela (3.1).

Fig. 3.2 - Schedules obtidas aplicando Hu.

Apesar do intervalo para a localização da solução ótima ter aumentado pode-se observar, com o auxílio das schedules geradas, que a solução heurística é bastante "satisfatória".

Depois de haver situado problemas de scheduling dentro das classes P e NP-completo foi descrito, ainda neste capítulo, o algoritmo de Hu. Este algoritmo foi implementado de maneira

exata e heurística conforme a complexidade do problema. Com as imple<sup>me</sup>ntações feitas foi possível determinar intervalos para a lo<sup>ca</sup>lização de soluções ótimas. Para isto, e com o objetivo de mos<sup>tr</sup>ar de maneira mais clara um procedimento heurístico, foram abor<sup>da</sup>dos exemplos de problemas com uma estrutura de precedência par<sup>ti</sup>cular (estrutura de árvore) onde os jobs requerem para processa<sup>me</sup>nto recursos de um mesmo tipo (máquinas idênticas). Problemas mais complexos, como problema de scheduling em projetos com restri<sup>ç</sup>ões de precedência e recursos, serão abordados no próximo capítu<sup>lo</sup>.

## CAPÍTULO IV

### PROCEDIMENTOS HEURÍSTICOS EM PLANEJAMENTO DE PROJETOS COM RESTRIÇÕES DE RECURSOS

#### 4.1 - ASPECTOS DE PLANEJAMENTO DE PROJETOS

A base de todas as técnicas de scheduling em network é o diagrama network de projeto. A Figura 4.1 mostra um diagrama atividade-em-nodo representando um projeto. Como já foi observado no Capítulo II, nesta representação os nodos representam jobs e os arcos precedência.

Uma alternativa para a representação de network é o diagrama-em-arco como mostra a Figura 4.2, no qual, são trocados os nodos por arcos. No diagrama atividade-em-arco os arcos denotam os jobs e cada job começa e termina em um único par de nodos chamados eventos, os quais, servem como identificadores dos jobs. Os primeiros diagramas network foram do tipo atividade-em-arco e

esta representação é, ainda hoje, amplamente usada. Jobs "fantasmas", de duração zero, usualmente representados por linhas pontilhadas são, às vezes, requisitados para corrigir alguma situação na lógica do network.

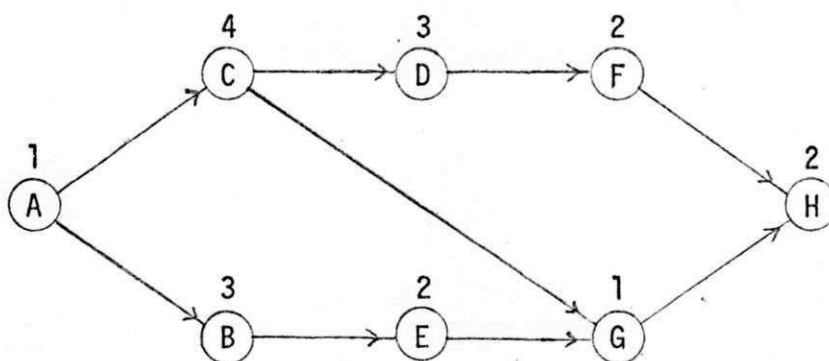


Fig. 4.1 - Diagrama atividade-em-nodo.

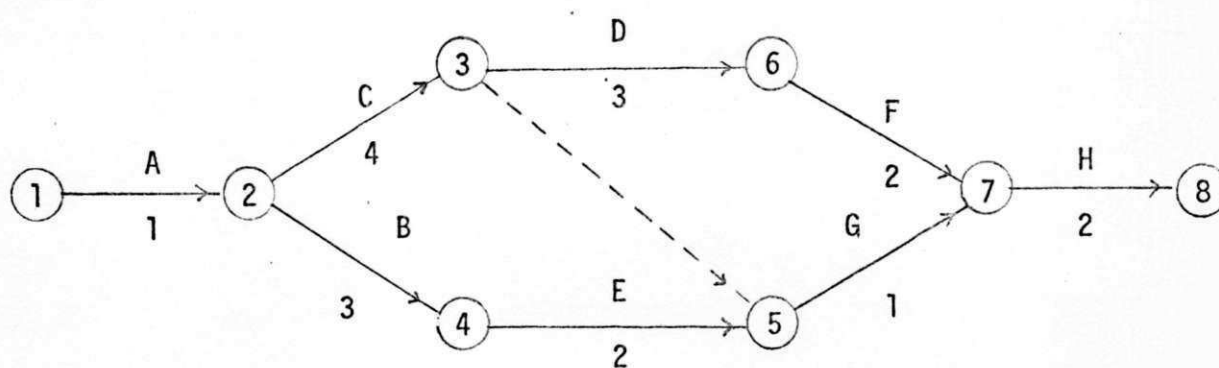


Fig. 4.2 - Diagrama atividade-em-arco representando o mesmo projeto da Figura 4.1.



O diagrama atividade-em-nodo, relativamente desconhecido durante os anos 60, tem aumentado sua popularidade nos recentes anos. Mesmo não sendo tão amplamente usado como o diagrama atividade-em-arco ele está disponível atualmente, como uma opção para grandes programas computacionais, para análise de network. Uma vantagem desta representação, é o fato de não se utilizar jobs "fantasmas", a não ser que o projeto tenha mais de um job inicial ou final.

Network de projeto também é a base para todos os cálculos dos métodos do caminho crítico (CPM); do tempo de início mais cedo (earliest start time, E S T), do tempo de início mais tarde (latest start time, L S T), da folga total, etc... Os procedimentos de CPM geralmente utilizam apenas tempo como informação e por isso para calcular o caminho crítico do network não considera requerimento e disponibilidades de recursos..

Os tempos de início e término de um job, produzidos dos cálculos do método CPM, implicam em perfis de recursos, utilizados sobre o tempo, isto é, quantidade de recursos requerida em função do tempo, como ilustra a Figura 4.3. Quando os níveis de disponibilidades de recursos são checados contra esses níveis de demandas requeridos, aparece então o problema de alocação de recursos. Pode ser que as demandas excedam os níveis de disponibilidade em alguns períodos de tempo. Pode ser, também, que a variação do perfil de recurso seja considerada excessiva, e exista uma razão para reduzir picos em excesso e nivelar este perfil. Pode ser ainda que, numa primeira fase, a duração do projeto não seja satisfatória e, então, são requeridas alocações de recursos adi

cionais podendo, com isso, os jobs serem concluídos em menos tempo e conseqüentemente diminuir a duração do projeto. Em suma, o problema de alocação de recursos aparece em uma variedade de tipos de problemas. Um exemplo é o problema de scheduling em projetos com limite pré-fixado de recursos.

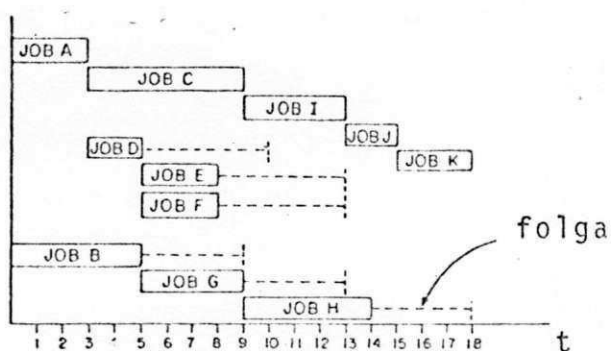


Fig. 4.3a - Schedule construída de um network onde todos os jobs estão no EST.

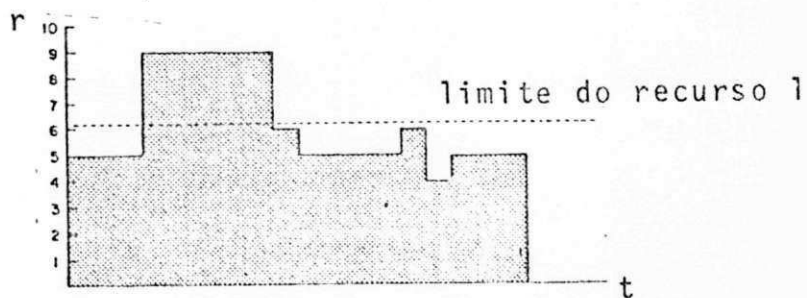


Fig. 4.3b - Perfil de um recurso tipo 1.

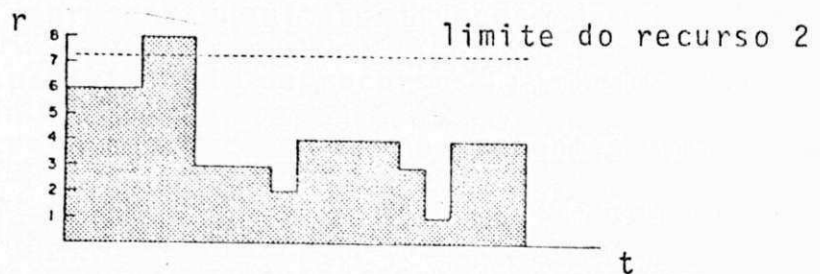


Fig. 4.3c - Perfil de um recurso tipo 2.

Fig. 4.3 - Geração de perfis de recurso de um diagrama network.

Um problema de scheduling em projetos com limite pré-fixado de recursos é assim denominado quando existem quantidades fixas de recursos disponíveis  $R_k$ , ou melhor, é um problema de scheduling com toda a generalidade, dentro do modelo descrito no Capítulo II. Quando as quantidades  $R_k$  não são suficientes para satisfazerem as demandas dos jobs concorrentes, aparecem então decisões de sequenciamento, muitas vezes, resultando em um aumento na duração do projeto, além da duração do caminho crítico. Neste tipo de problema, o objetivo mais comum é minimizar  $C_{max}$ , isto é, reduzir a execução do projeto ao menor tempo possível; outros ob

provado através da farta bibliografia no assunto que modelos net  
work são meios utilmente aplicáveis numa grande variedade de problema  
s de scheduling e planejamento. Mas, é reconhecido que estes pro  
cedimentos básicos não representam muitas situações da vida real,  
porque eles assumem disponibilidade de recursos ilimitada. Como re  
sultado, tem sido crescente a atenção dada nos recentes anos aos  
problemas de scheduling com restrições de recursos, e tem sido i  
gualmente crescente o desenvolvimento no campo de procedimento em  
network nesta área.

O problema também ganha importância pelo fato do mes  
mo ser representativo da classe NP-completo. E como tal, deve ser  
abordado por procedimentos heurísticos. Pois, como foi declarado no  
Capítulo II, não existem (pelo menos até o presente) algoritmos que  
venham conseguir soluções exatas para problemas NP-completos. Observe  
que o problema de sequenciamento em job-shop, pelas suas caracter  
ísticas, se assemelha bastante com este tipo de problema. Isto tem  
estimulado inclusive o interesse de pesquisadores na aplicação dos  
procedimentos desenvolvidos, a priori, para problemas com estrutura  
job-shop para problemas com estruturas gerais de precedência e re  
ursos. Backer (31) considera o problema de scheduling em projetos  
como uma extensão do job-shop onde ele substitui máquinas por gru  
pos de máquinas.

#### 4.2 PROCEDIMENTOS HEURÍSTICOS NO PLANEJAMENTO DE PROJETOS COM RESTRIÇÕES DE RECURSOS.

Existem muitas maneiras possíveis pelas quais, procedimentos de scheduling com restrições de recursos podem ser classificados. Por exemplo, quanto ao tipo de resultados, esses procedimentos podem ser agrupados nas duas classes abordadas no Capítulo III; procedimentos ótimos ou exatos, e procedimentos heurísticos ou de aproximações. Dentro de cada uma dessas classes existem ainda algumas categorias. Por exemplo, a existência de procedimentos heurísticos se verifica dentro dos métodos serial e paralelo. Estes termos, serial e paralelo, foram usados inicialmente por Kelle, (22). Serial, para se referir a uma estratégia onde uma possível schedule pode ser construída, considerando os jobs na ordem de seus aparecimentos em uma lista e processando-os um de cada vez, o mais cedo possível, desde que sejam obedecidas as restrições de precedência e recursos disponíveis. O termo paralelo, para se referir a uma estratégia, pela qual, vários jobs são processados de uma vez.

Para ambos os procedimentos, heurísticos e exatos, pode ser feita ainda uma classificação em termos da quantidade de diferentes tipos de recursos envolvidos no modelo de projetos, onde os projetos envolvem:

- a) Um único tipo de recurso, comum a todos os jobs do projeto, com requerimentos e disponibilidades expressas em múltiplos da unidade; por exemplo,

3 homens ou 2 guindastes ou 4 caminhões, etc.

- b) Mais de um tipo de recurso por projeto, mas sõ mente um tipo por job, com disponibilidades e re querimentos limitados a 1 unidade de recurso, co mo no problema de scheduling em job-shop.
- c) Mais de um tipo de recurso por job e por proje to, com requerimentos e disponibilidades expres sos em múltiplos da unidade.

Estes três casos são, respectivamente denominados de modelos de único-recurso, modelos job-shop, e modelos de multi-recursos.

Por ser o problema de scheduling em projetos com restrições de recursos um problema NP-completo, os procedimentos para a resolução do mesmo deverão ser heurísticos. Como já ficou claro antes, no final do Capítulo III, apesar de não produzirem resultados ótimos, estes procedimentos conduzirão, sistematicamen te, a boas schedules.

Em termos gerais, as estratêgias dos métodos se riais e paralelos, representam a abordagem heurística básica para a resolução de problemas em larga escala.

Procedimentos heurísticos para o caso de único-recur so representam, para modelos de scheduling em projetos, o tipo mais simples. E, por isso, a maioria dos trabalhos desenvolvidos neste caso são designados, com poucas exceções, para textos acadê micos.

Para modelos de projetos onde existem estruturas "job-shop", o mais significativo estudo de procedimentos heurísticos

foi conduzido por Mueller-Mehrbach (23), onde se discutem solu  
ções obtidas heurísticamente, comparando-as com soluções ótimas  
de problemas já resolvidos.

Provavelmente, a primeira publicação em que se dis  
cutem procedimentos heurísticos para multi-projetos foi Kelley  
(22), onde se descreve uma rotina serial, programada para um IBM  
650, e que é capaz de trabalhar com um total de 4 diferentes tipos  
de recursos por job e 9 por projeto. Como resultado, obteve-se, em  
vários projetos, reduções de 35 % a 50 % em picos de recursos re  
queridos, com um crescimento médio na duração do projeto em torno  
de apenas 5 %.

Alguns pacotes de procedimentos heurísticos, são co  
mercialmente disponíveis para grandes projetos. Um dos mais conhe  
cidos, foi desenvolvido por Wiest (24). Seus programas SPAR (Sche  
duling Program for Allocating Resources) I e II têm sido aplicados  
para problemas com mais de 200 jobs e 20 diferentes tipos de recur  
sos, com requerimentos e limites de recursos fixos ou variáveis.  
Muitos desses projetos por serem desenvolvidos nas próprias organi  
zações comerciais, não são disponíveis, em geral, quanto aos de  
talhes de operação. Outros, porém, com detalhes operacionais colo  
cados à disposição do usuário, são listados (extraído de Davis  
(25)) na Tabela 4.1.

Tabela 4.1 - Características de alguns programas computacionais referentes a procedimentos heurísticos para scheduling em projetos com restrições de recursos.

Nome do programa	Características
CPM-RPSM (Resource Planning and Scheduling Method) CEIR, Inc.	2000 - 8000 jobs por projeto, 4 tipos de recurso por projeto, permitindo interrupção para os jobs e restrições de tempo de início e término para os jobs. Usa um determinado procedimento heurístico.
MSCS (Management Scheduling and Control System) McDonnell Automation	Capacidade para 25 múltiplos projetos, 18.000 jobs, 12 tipos de recursos por job. Scheduling heurística baseada em complexas funções de prioridades, controláveis pelo usuário.
PMS/360 (Project Management System) IBM Corp.	Um grande e complexo sistema de informação consistindo de módulos um dos quais sendo sobre alocação de recurso. Manipula com diagramas atividade-em-arco ou diagramas de precedência; é permitido até 225 múltiplos projetos com 32.000 jobs e 250 tipos de recurso. Apresenta opções e escolha de sequenciamento heurístico.



Continuação Tabela 4.1

Nome do programa	Características
PPS IV (Project Planning System) Control Data Corp.	2000 jobs por projeto permitindo até 20 tipos de recursos por job. Pode ser usado com múltiplos projetos ou um único projeto; permite superposição de jobs. Faz nivelamento de recurso com duração fixada. Prioridades entre os recursos podem ser especificadas. Usa um procedimento heurístico fixo.
Project/2, Project Software Inc.	Permite 50 múltiplos projetos, 32.000 jobs, centenas de tipos de recurso, fácil atualização e apresenta análise de custos. Opção de sequências heurísticas, especificadas pelo usuário.

#### 4.3 - RESULTADOS HEURÍSTICOS DE SCHEDULING EM PROJETOS COM RESTRIÇÕES DE RECURSOS.

Nesta seção, é descrito um procedimento heurístico que visa encontrar uma "boa" solução para um problema de scheduling em projetos com restrições de recursos. Quase todos os heurísticos se baseiam em listas para a construção de algoritmos (8,11, 31). O procedimento descrito nesta seção também utiliza lista de

prioridade e é baseado num algoritmo desenvolvido por Brucker (21). Basicamente, este procedimento tem duas partes. Na primeira parte encontra-se uma lista de todos os jobs, a qual é usada na segunda parte para se construir uma possível schedule. A lista é construída colocando os jobs do projeto em ordem não decrescente conforme um dos critérios mais conhecidos, tal como: tempo de início mais tarde, tempo de início mais cedo, folga total, tempo de processamento, a soma da folga total e tempo de processamento dos jobs, etc. Estes critérios são muito bem definidos por McLaren e Buesnel (26). Na segunda parte, scheduling, caso exista mais que um job com o mesmo grau de prioridade, um segundo critério será necessário como um meio de escolha entre eles para a scheduling.

Para construção da schedule examina-se a lista ordenada para encontrar o primeiro job que pode ser processado no tempo vigente  $T$ . Este job é colocado na schedule e é feita a atualização dos recursos disponíveis para o período de tempo em que o job é processado. É selecionado o próximo job que pode ser processado no tempo  $T$  e que também é colocado na schedule, e assim por diante. Se para o tempo  $T$  não existir, em condições de processamento, mais algum job, então  $T$  será substituído pelo menor  $C_j$  de algum job  $j$  que seja maior que  $T$ . Este procedimento continua até a lista ficar vazia, ou melhor, até que todos os jobs estejam na schedule. Mais detalhes deste procedimento podem ser observados no algoritmo a seguir. Antes da descrição do algoritmo, lista-se abaixo as variáveis utilizadas no mesmo e seus respectivos significados:

$n$	número de jobs
$P(j)$	array contendo o número corrente de predecessores do job $j$ .
$A(i,j)$	array contendo os elementos da matriz adjacência do network. Será atualizado apropriadamente.
$C(j)$	array onde estão armazenados os tempos de término do job $j = 1, \dots, n$ .
$X(j)$	array onde estão armazenados os tempos de início dos jobs.
$T_{max}$	é um limite para o tempo de processamento de todo o projeto.
$R(t)$	o total de recurso utilizado pelos jobs na schedule até o tempo $T$ .
$L$	lista dos jobs ordenados segundo o critério utilizado.
$R_j$	a demanda do job $j$ .
$p_j$	tempo de processamento do job $j$ .
$R$	vetor capacidade

Indica-se que um job  $j$  é eliminado da lista por  $L \setminus \{j\}$ .

## Algoritmo 2

Passo 1.

```

for j ← 1 to n step 1 do
  begin
    P(j) ←  $\sum_{i=1}^n A(i,j)$ ;
    C(j) ← ∞;
  end;
for T ← 0 to Tmax step 1 do
  R(T) ← 0;
  T ← 0;

```

Passo 2.

```

While L ≠ ∅ do
  begin
    if não existe j em L com P(j) = 0 e R(T) + Rj ≤ R then
      begin
        Encontrar j* com C(j*) = min{C(j) | T ≤ C(j)};
        T ← C(j*);
      end
    end

```

```

for j ← 1 to n step 1 do
  if C(j) = T then
    begin
      for i ← 1 to n step i do
        if A(j,i) = 1 then
          begin
            P(i) ← P(i) - 1;
            A(j,i) ← 0;
          end;
        L ← L \ {j};
      end;
    end
  else
    begin
      Seja j o primeiro job em L com P(j) = 0 e  $R(T) + R_j \leq R$ ;
      X(j) ← T;
      for k ← T to T + pj - 1 step 1 do
        R(k) ← R(k) + Rj;
      C(j) ← T + pj;
    end;
  end;

```

No Apêndice encontra-se uma listagem de programa com base no Algoritmo 2. Este procedimento pode ser testado para problemas de scheduling com restrições de recursos e precedência e é aplicado ao exemplo a seguir.

Exemplo:

Considere o projeto definido nas quatro primeiras colunas da Tabela 4.2.

As restrições dos recursos são feitas sobre os quatro tipos diferentes disponíveis em quatro unidades para o tipo 1, seis para o tipo 2, cinco para o 3 e sete unidades para o tipo 4. O objetivo do problema é a execução do projeto no menor tempo possível, ou seja, minimizar  $C_{max}$ .

O network para este exemplo é mostrado na Figura 4.4. Os números colocados em cima de cada job representam seus respectivos tempos de processamento. Na notação  $a/b$  colocada abaixo de cada job,  $a$  representa o tempo de início mais cedo (EST) e  $b$  representa o tempo de início mais tarde (LST), de cada job. Então, a folga total será  $b - a$ .

Para este exemplo, utilizando o critério LST, conseguiu-se uma solução heurística de 56 unidades de tempo. Esta solução é mostrada na Figura 4.5, onde aparece a schedule solução e logo abaixo, os perfis dos recursos utilizados em cada período de tempo.

Tabela 4.2

job i	$p_i$	Sucessores diretos	$r_i$ : vetores de recursos requeridos	EST	LST	Folga total	Folga to tal + $p_i$
1	4	6,7,8	(1,1,2,0)	0	4	4	8
2	9	8	(2,0,1,3)	0	0	0	9
3	2	9	(3,4,2,5)	0	5	5	7
4	5	9,10	(0,1,0,1)	0	2	2	7
5	7	11	(0,3,2,5)	0	2	2	9
6	1	12	(0,0,0,1)	4	16	12	13
7	3	13	(0,0,1,0)	4	8	4	7
8	2	13	(0,1,0,0)	9	9	0	2
9	6	14,15	(1,0,0,0)	5	7	2	8
10	5	15	(3,4,2,5)	5	8	3	8
11	4	15	(2,2,0,1)	7	9	2	6
12	1	16	(2,1,3,0)	5	17	12	13
13	7	16	(0,4,3,3)	11	11	0	7
14	4	16,17,18,19	(0,2,0,3)	11	13	2	6
15	5	20	(1,1,1,1)	11	13	2	7
16	5	21,22	(1,0,0,1)	18	18	0	5
17	1	23	(0,1,1,1)	15	23	8	9
18	2	23	(1,0,0,5)	15	22	7	9
19	7	23	(1,2,0,3)	15	17	2	9
20	3	24	(0,1,0,2)	16	18	2	5
21	4	25	(1,0,0,2)	23	23	0	4
22	3	26	(1,2,3,4)	23	24	1	4
23	2	27,29	(0,1,1,3)	22	24	2	4
24	5	27	(1,2,3,4)	19	21	2	7

Continuação da Tabela 4.2

Job $i$	$p_i$	Sucessores diretos	$r_i$ : Vetores de recursos requeridos	EST	LST	Folga total	Folga to tal + $p_i$
25	2	28	(1,4,3,1)	27	27	0	2
26	1	28,29	(1,0,0,1)	26	27	1	2
27	1	30,31	(0,1,2,2)	24	26	2	3
28	3	-	(4,6,5,7)	29	29	0	3
29	4	-	(0,0,0,1)	27	28	1	5
30	5	-	(1,2,0,0)	25	27	2	7
31	2	-	(1,1,2,1)	25	30	5	7

Se não houvesse limitações de recursos,  $C_{max}$  seria igual a 32 unidades de tempo, como pode ser observado na Figura 4.6. Nesta figura aparece a schedule. Logo abaixo, as quantidades de recursos consumidos em cada período de tempo. Observando atentamente, pode-se verificar que os picos máximos de recursos requeridos durante toda a scheduling foram de 8 unidades, 9 unidades, 9 unidades e 14 unidades, respectivamente, para os tipos 1,2,3 e 4. Então, tomando estes picos como limites de recursos disponíveis, poder-se-ia chegar à conclusão do projeto no tempo crítico, ou seja, na menor quantidade de tempo possível e portanto, conseguindo uma schedule ótima. Verifica-se ainda na Figura 4.6, que a



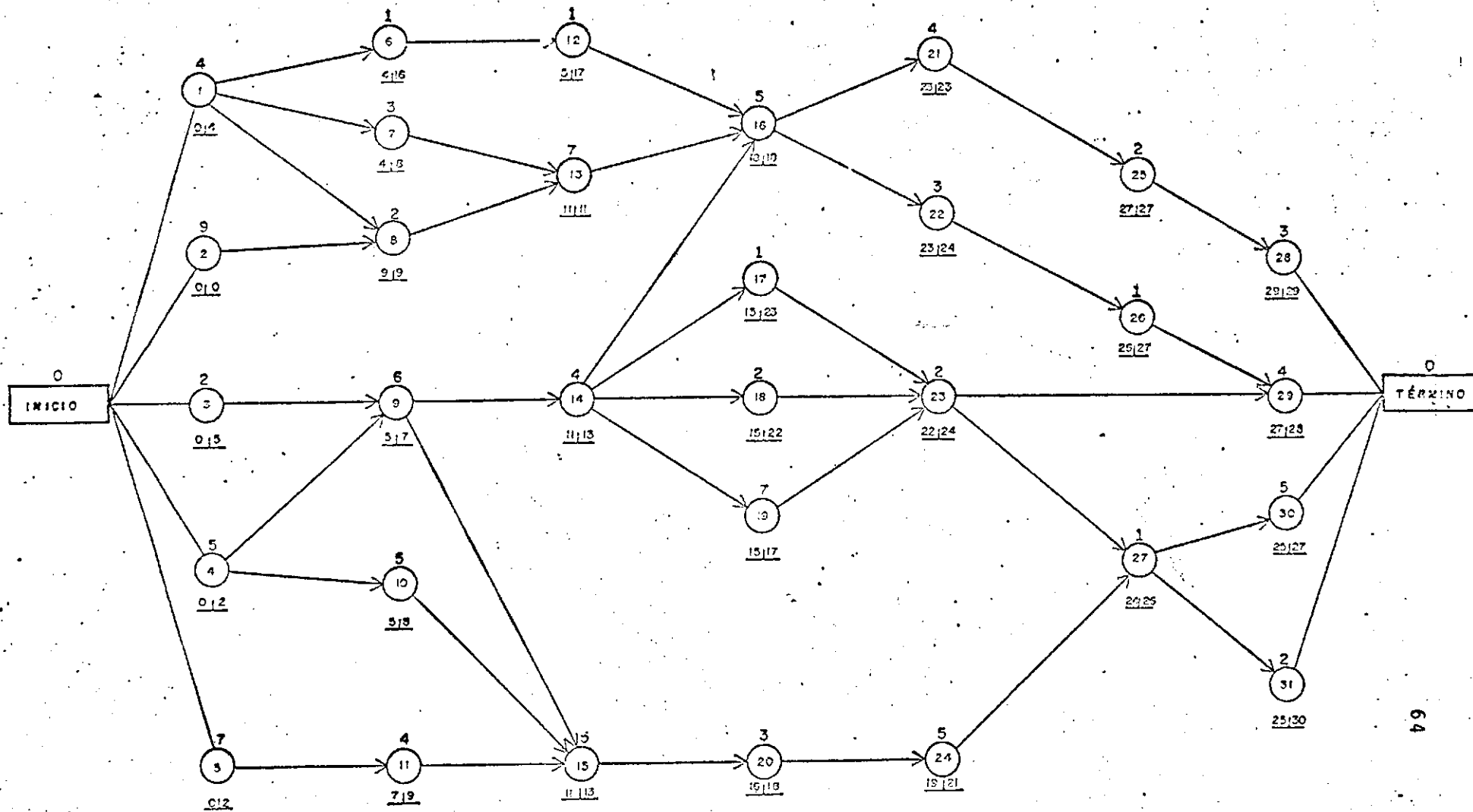


Fig. 4.4 - Network representando o projeto definido na tabela 4.2

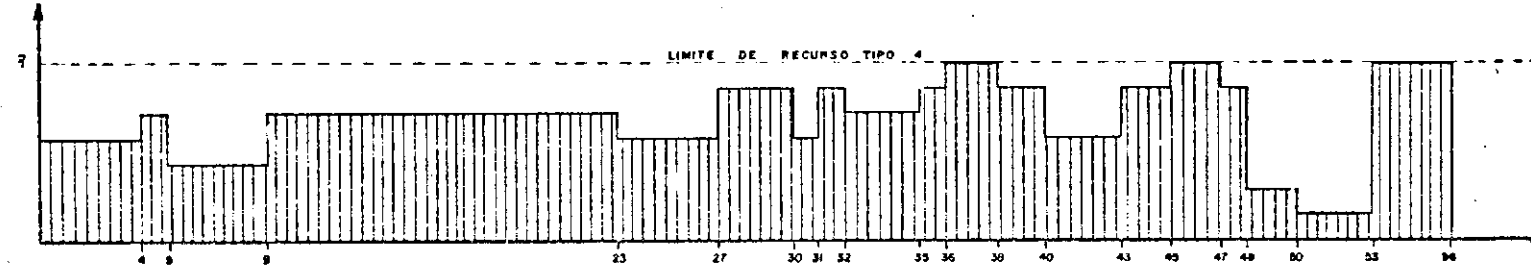
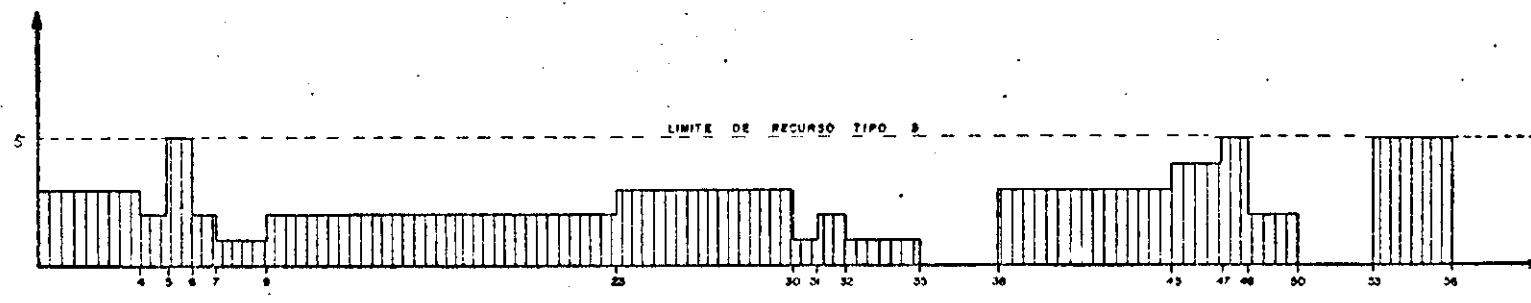
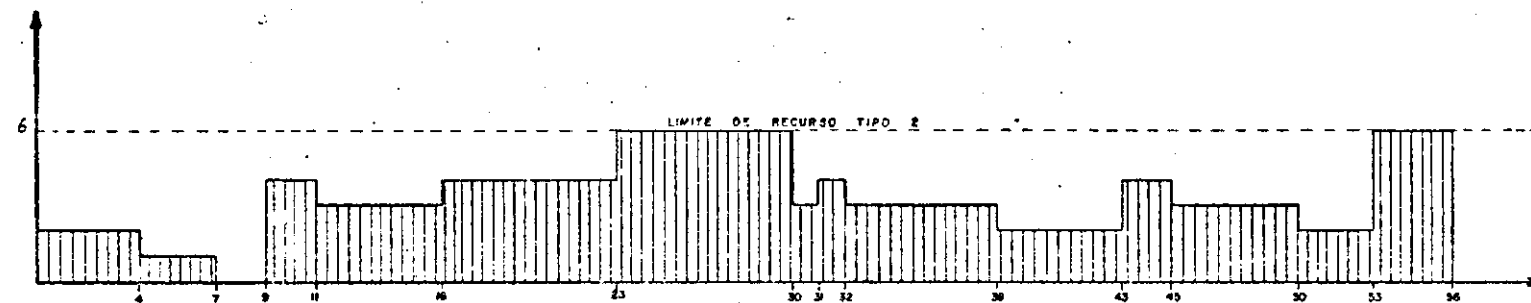
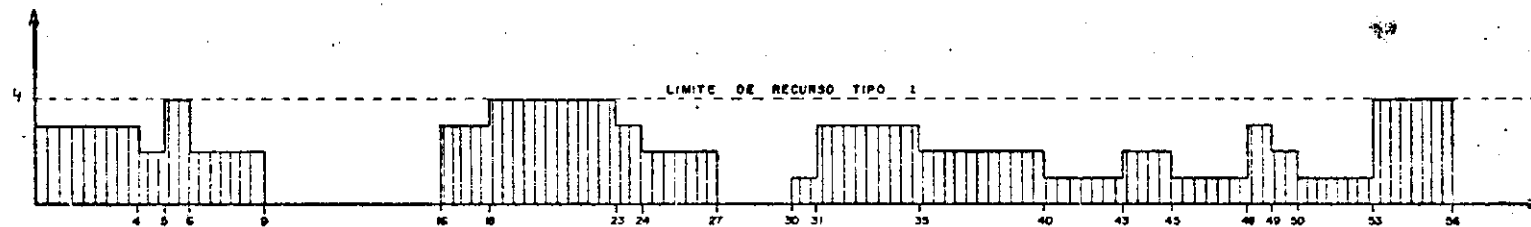
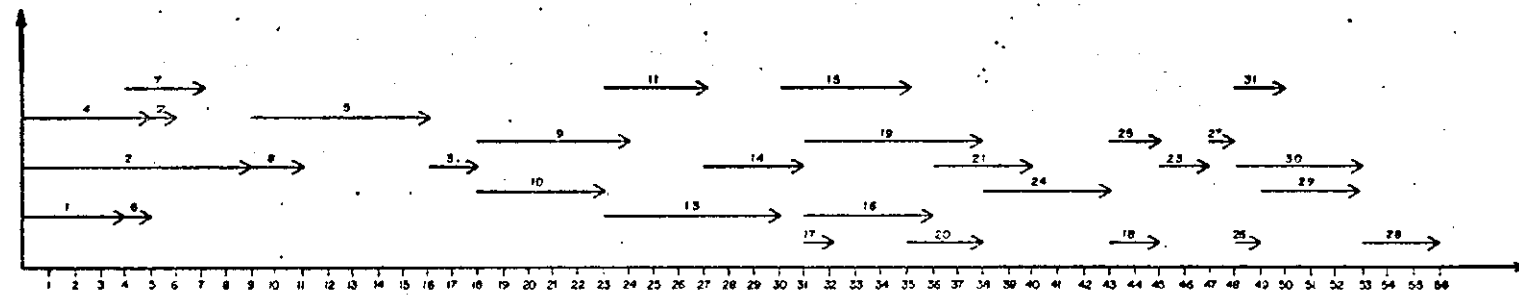


Fig. 4.5 - Schedule e perfis dos tipos de recursos conseguidos para o projeto definido na Tabela 4.2 aplicando o critério LST

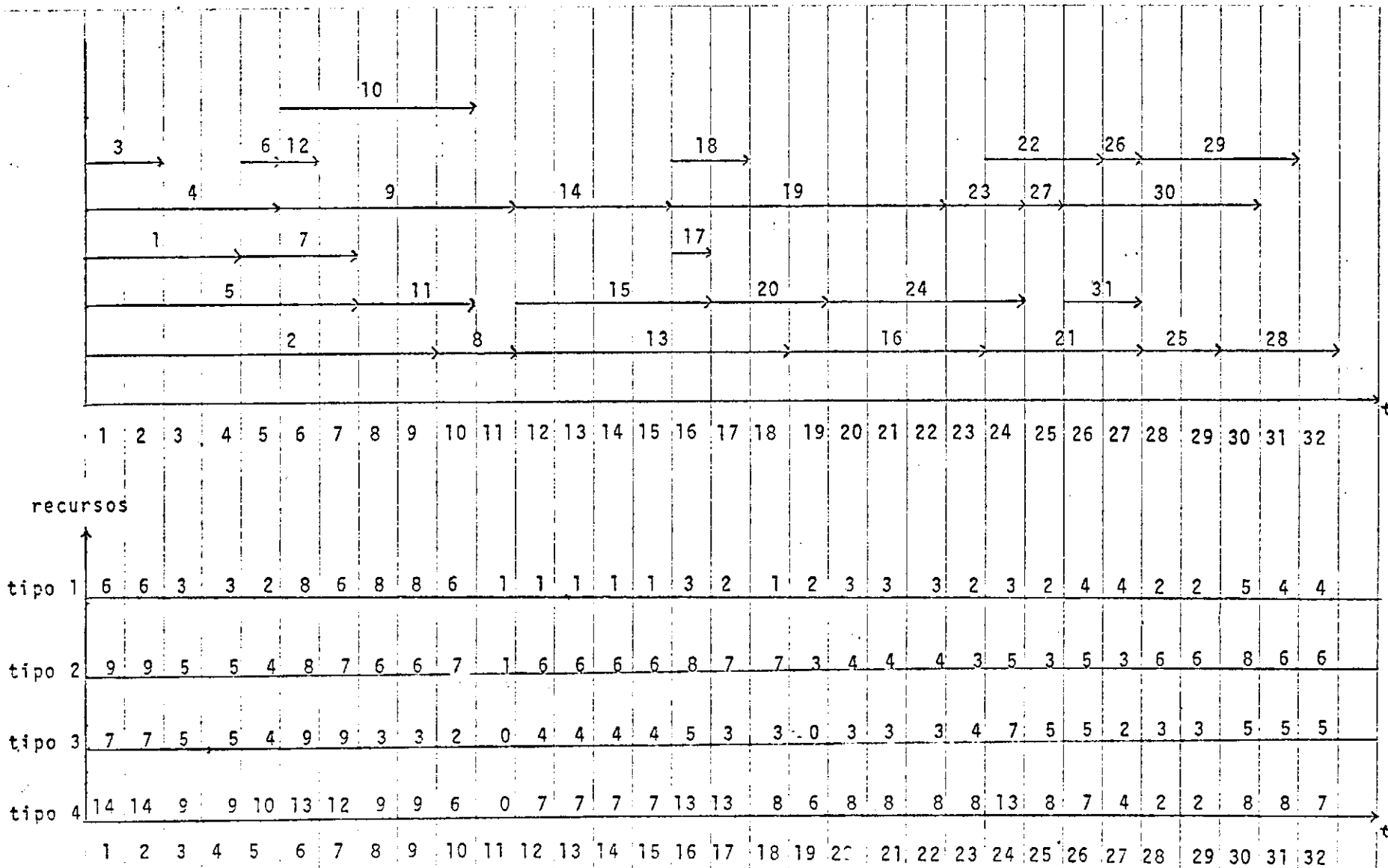


Fig. 4.6 - Schedule e perfis de recursos conseguidos para o projeto definido na Tabela 4.2 aplicando o critério LST e considerando recursos disponíveis em quantidades ilimitadas.

schedule mostra os jobs críticos (jobs que pertencem ao caminho de maior duração do início para o fim do projeto, caminho este assinalado no network da Figura 4.4).

Como já foi assinalado anteriormente, as estratégias dos métodos serial e paralelo constituem a abordagem heurística básica para a resolução destes problemas. O Algoritmo 2, utilizando uma lista de prioridade, processa os jobs em série. Segue-se em linhas gerais como funciona a estratégia do método serial ilustrado pela Figura 4.7 e que é identificado pelo "While loop" no passo 2.

Suponha que, para serem processados, os jobs são classificados em ordem não decrescente dos EST conforme dados da Tabela 4.2. No tempo 0 os jobs em condições de processamento formam o conjunto, em ordem, {3,1,4,5,2}. Assim, os jobs 3,1 e 4 são inicializados no tempo 0, não dando condições durante os dois primeiros períodos de tempo de qualquer outro job do conjunto ser inicializado, por causa do recurso disponível. Quando o job 3 é concluído o conjunto em condições para o processamento torna-se igual a {5,2} e, então, apenas o job 5 tem condições de ser processado, não dando condições ao job 2 até o período de tempo 4, quando o job 1 é concluído. No instante do tempo 4 o conjunto em condições torna-se {2,6,7,11}. O próximo job a ser processado seria o 2 mas, como a disponibilidade de recursos não permite, fica em condições apenas o job 6. Com isso o job 2 fica num estado de espera até a conclusão dos jobs 6 e 4. Neste ponto o conjunto em condições torna-se {2, 7, 12, 10, 9, 11}, e continuando desta maneira obtém-se  $C_{max}$  igual a 55 unidades de tempo como uma

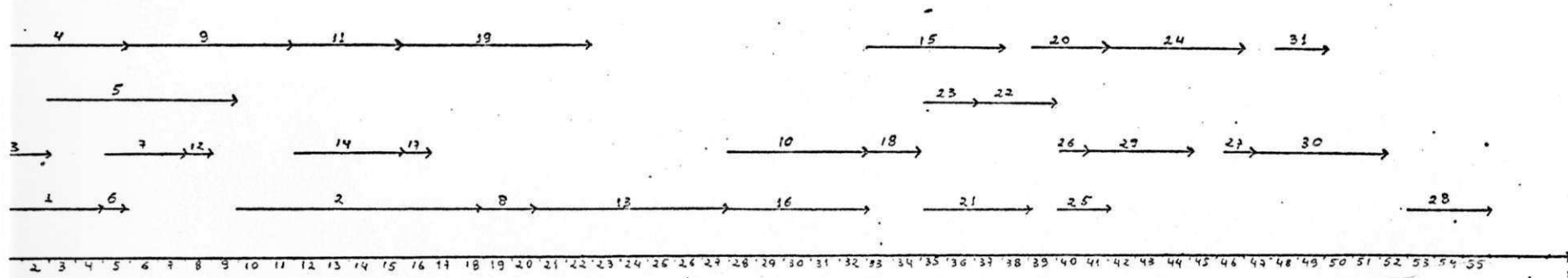


Fig. 4.7 - Schedule conseguida para o projeto definido na tabela 4.2 aplicando o critério EST.

solução heurística. Observe que este método é serial pelo fato de que uma possível schedule pode ser construída considerando os jobs na ordem de suas aparições na lista e colocando-os na schedule um de cada vez, tão cedo quanto possível, desde que as restrições de precedência e recursos permitam.

Além das regras de scheduling, utilizando listas de prioridade de acordo com os EST e LST dos jobs, para o exemplo definido com os dados da Tabela 4.2, foram testadas outras regras utilizando-se de cada job, tempo de processamento, folga total e soma de folga e tempo de processamento. Em todos os casos foram conseguidas schedule atingindo diferentes valores para Cmax. Esses valores aparecem na tabela a seguir a frente dos respectivos fatores de prioridades utilizados nas listas.

Fatores de prioridades	Cmax
Tempo de início mais tarde (LST)	56 unidades de tempo
Tempo de início mais cedo (EST)	55 unidades de tempo
Tempo de processamento dos jobs	58 unidades de tempo
Folga total	66 unidades de tempo
Folga total + tempo de processamento dos jobs.	57 unidades de tempo

Tabela 4.3 - Construída com os resultados das implementações do Algoritmo 2.

Dentre estas várias regras de scheduling usadas se  
rá então escolhida como solução heurística a melhor schedule, ou  
seja, a schedule conseguida utilizando o fator EST e que produ  
ziu  $C_{max} = 55$  unidades de tempo.

## CAPÍTULO V

### CONCLUSÕES

Após a descrição do modelo geral, no Capítulo II, foram mostrados ainda possíveis situações reais onde podem aparecer problemas de scheduling. Na descrição do modelo geral foi possível apresentar uma notação para problemas de scheduling, introduzir medidas quantitativas para avaliação de schedules, mostrar diferentes funções objetivas e descrever sobre o diagrama de Gantt para representação de schedules. A fim de familiarizar o leitor como o problema de scheduling, principalmente no que diz respeito a restrições de recursos e precedência, foram apresentados diversos exemplos com características próprias de problemas reais, isto é, problemas facilmente encontrados no nosso dia-a-dia.

No Capítulo III, após definir, de maneira informal,



as classes de problemas polinomialmente solucionáveis (classe P) e de problemas intratáveis por procedimentos exatos (classe NP-completo), foram listados vários tipos de problemas de scheduling atualmente provados como sendo P ou NP-completos e as referências correspondentes. Nesta parte foram apresentadas tabelas para orientação do leitor tentando mostrar inclusive uma linha limite entre as classes P e NP-completo. Na Tabela 3.1 foram classificados os mais conhecidos tipos de problemas de scheduling conforme a complexidade dos mesmos, e na Tabela 3.2 foram feitas referências a algoritmos para aqueles solucionáveis polinomialmente. Como resultado dessas abordagens nota-se que somente os problemas mais simples de scheduling são solucionáveis polinomialmente por algoritmos exatos, daí a motivação dos heurísticos para os demais. No final deste capítulo, Secção III.3, foi mostrado um problema de scheduling com uma estrutura de precedência especial (estrutura de árvore), onde os jobs requerem tempos iguais para processamento. Para este problema conseguiu-se solução exata com a aplicação de um procedimento baseado no algoritmo de Hu. Fazendo uma simples mudança em um dos parâmetros do problema (neste trabalho a mudança foi feita com relação aos tempos de processamento dos jobs) este tornou-se NP-completo e, conseqüentemente, foi necessário fazer modificações na estrutura do algoritmo de Hu tornando-o heurístico para conseguir soluções aproximadas.

Na sua forma geral, um problema de scheduling com restrições de precedência e recursos é um problema combinatorial de tal magnitude que todos os procedimentos existentes para encontrar soluções exatas são impraticáveis para problemas de dimensões rea

lísticas. Somente para problemas relativamente pequenos, isto é , com pequena quantidade de jobs, existem procedimentos viáveis para determinar schedules ótimas, como programação inteira e "branch and bound".

Este trabalho se propõe a mostrar procedimentos heurísticos em problemas de scheduling com restrições de precedência e recursos. Para isso foi escolhido um problema combinatorial que fosse motivante, onde estas restrições aparecem mais claramente, e que, ao mesmo tempo, abrangesse todas as demais situações apresentadas neste trabalho. Para isso abordou-se, no Capítulo IV, o problema de scheduling em planejamento de projetos com restrições de recursos. Após descrever aspectos de planejamento de projetos e abordar procedimentos heurísticos em planejamento de projetos com restrições de recursos foi apresentado um procedimento heurístico (Algoritmo 2) para problemas de scheduling mais gerais. Este procedimento foi testado num projeto simulado e conforme o critério de prioridade dos jobs para a scheduling conseguiu-se resultados heurísticos adversos, entre os quais, foi escolhido "o melhor" como solução do problema. O Algoritmo 2 pode ser implementado para um projeto real bastando apenas associar as restrições de recursos e precedência indicadas na Tabela 4.2, dados de um problema real. Além do mais, este procedimento, devido a sua generalidade, pode ser aplicado a todos os demais tipos de problemas apresentados nos capítulos anteriores. É verdade porém que, nesse caso, os resultados obtidos podem não ser suficientemente representativos se comparados com resultados obtidos da aplicação de algoritmos desenvolvidos especificamente para problemas com estruturas especiais, como por

exemplo o de Hu desenvolvido para problemas com estruturas de árvore. Para uma abordagem mais abrangente sobre procedimentos heurísticos específicos para problemas de scheduling com estruturas especiais, ver Backer (31).

Procedimentos heurísticos, estruturalmente similares aos abordados neste trabalho (isto é, baseados em listas de prioridades), podem ser aplicados em diversos tipos de problemas. Exemplos são: controle de produção em organizações de manufaturamento, scheduling de operações em construções, scheduling de programas de rádio e televisão, etc. Convém registrar porém que, esperar que todos esses problemas pudessem ser resolvidos, integralmente, pelos procedimentos aqui implementados, seria muita pretensão. A propósito, em se tratando de problemas de scheduling, convém citar Kelley ( ) quando ele, referindo-se à escolha de um algoritmo para um determinado problema, disse: "... O único modo de responder realmente à pergunta 'até que ponto isto é bom?' é aplicar o algoritmo em algum problema prático e ver o que acontece. Se os resultados forem racionais, existe algum mérito para o algoritmo. Isto, todavia, não nos autoriza a generalizar que (mesmo que as circunstâncias intuitivamente indiquem) o algoritmo seja convenientemente aplicável (isto é, com possibilidades de sucesso) a outros problemas.

A luz destas conclusões, pode-se dizer que é de grande importância a análise de resultados obtidos quando da utilização de procedimentos heurísticos. Com esta análise, às vezes referida como "análise do pior caso", pode-se determinar até que ponto um procedimento produz, ou não, um "bom" resultado para um problema. Neste trabalho não tivemos a preocupação de proceder a uma análise

de pior caso para os procedimentos implementados. Isto, deixamos como sugestão para um próximo trabalho.

APENDICE

```

0000 1-- BEGIN
0001 -- COMMENT VICTOR HUGO DE SOUZA LISBOA
0001 -- *****
0001 -- *
0001 -- *
0001 -- *
0001 -- * PROCEDIMENTO EXATO PARA
0001 -- * MINIMIZAR O TEMPO TOTAL RE-
0001 -- * QUISTIÇÃO PARA O PROCESSAMEN-
0001 -- * TO DE TODOS OS JOBS
0001 -- *
0001 -- *
0001 -- * ESTES JOBS ORNAM UMA ES-
0001 -- * TRUTURA INTRE F REQUEREM
0001 -- * IGUAIS TEMPOS DE PROCF.SSAMENTO *
0001 -- *
0001 -- *
0001 -- * *****
0001 -- *
0001 -- * INTEGER M,N,TPROC,AUX,K,J,AUX1,T
0002 -- * INTERFSIZE =3
0003 -- * READ(M,N,TPROC)
0004 -- * WRITE (NUMERO DE JOBS,N= *M)
0005 -- * WRITE (NUMERO DE MAQUINAS,M= *M)
0006 -- * WRITE (TEMPO DE PROCESSAMENTO DE QUALQUER JOB = *TPROC)
0007 2-- BEGIN
0008 -- * INTEGER ARRAY SUCS, TINI, PRD (1 N)
0009 -- * INTEGER ARRAY VTAUX(1 M)
0010 -- * RECORD IJSTA(INTEGER JOB, ROT REFERENCE(IJSTA) LINK)
0012 -- * REFERENCE(IJSTA) APONT, NOVO, NOVO2
0013 -- * REFERENCE(IJSTA) ARRAY TEMP(1 M)
0014 -- * INTEGER PROCEDURE ROTUA(INTEGER VALUE I)
0015 -- * IF SUCS(I)=0 THEN 0
0015 -- * ELSE
0016 -- * ROTUA(SUCS(I)) + 1
0017 -- * PROCEDURE INSRD(REFERENCE(IJSTA)VAI DE RESULT APONT
0018 -- * REFERENCE(IJSTA) VALUE NOVO)
0019 3-- BEGIN
0020 -- * REFERENCE(IJSTA) TEMP,TEMP1
0021 -- * IF APONT= NULL THEN
0021 -- * BEGIN
0022 -- * APONT =NOVO
0023 -- * LINK(APONT) =NULL
0024 -- * END
0024 -- * ELSE
0025 4-- BEGIN
0026 -- * IF ROT(NOVO) =ROT(APONT) THEN
0026 5-- BEGIN
0027 -- * LINK(NOVO) =APONT
0028 -- * APONT =NOVO
0029 -- * END
0029 -- * ELSE
0030 5-- BEGIN
0031 -- * TEMP =APONT
0032 -- * WHILE ROT(NOVO) < ROT(TEMP) (
0032 -- * BEGIN
0033 -- * TEMP =TEMP
0034 -- * END
0035 -- * LINK(NOVO) =TEMP
0036 -- * LINK(TEMP) =NOVO
0037 -- *

```

```

0038 -5      END
0039 -4      END
0040 -3      END INSDRD
0041 --      FOR I =1 UNTIL N DO
0042 --      READLN(SUGS(I),PRFD(I))
0043 --      FOR J =1 UNTIL M DO
0044 --      READLN(VFTAUX(I))
0045 --      APONT =NULI
0046 --      FOR I =N STEP -1 UNTIL 1 DO
0047 --      BEGIN
0048 --      NCV0 =LISTA
0049 --      JPR(NOV0) =I
0050 --      ROT(NOV0) =ROTJWA(I)
0051 --      TINC(I) =0
0052 --      INSDRD(APONT,NOV0)
0053 --      END
0054 --      T =0
0055 --      FOR I =1 UNTIL M DO
0056 --      TEMP(I) =NULI
0057 --      WHILE APONT =NULI DO
0058 --      BEGIN
0059 --      FOR K =1 UNTIL M DO
0060 --      BEGIN
0061 --      NOV01 =APONT
0062 --      NOV02 =NULI
0063 --      NOV01 =LINK(NOV01)
0064 --      WHILE (NOV01 =NULI) AND (PRFD(JOB(NOV01)) =0) DO
0065 --      BEGIN
0066 --      NOV02 =NOV01
0067 --      NOV01 =LINK(NOV01)
0068 --      IF NOV01 =NULI THEN
0069 --      BEGIN
0070 --      VFTAUX(K) =SUCC(JOB(NOV01))
0071 --      NOV01 =LINK(NOV01)
0072 --      IF TEMP(K) =NOT THEN
0073 --      BEGIN
0074 --      LINK(NOV01) =NULI
0075 --      TEMP(K) =NOV01
0076 --      END
0077 --      ELSE
0078 --      BEGIN
0079 --      LINK(NOV01) =TEMP(K)
0080 --      TEMP(K) =NOV01
0081 --      END
0082 --      UNTIL (JOB(NOV01)) =I
0083 --      IF NOV01 =APONT THEN
0084 --      APONT =NOV0
0085 --      ELSE
0086 --      LINK(NOV02) =NOV0
0087 --      END
0088 --      T = T + TPRIC
0089 --      FOR I =1 UNTIL M DO
0090 --      IF ((VFTAUX(I) =-99) AND (VFTAUX(I) =0)) THEN
0091 --      BEGIN
0092 --      PRFD(VFTAUX(I)) =PRFD(VFTAUX(I)) -1
0093 --      VFTAUX(I) =-99
0094 --      END
0095 --      END

```

```

0082 --3 FND
0083 -- WRITE( )
0084 -- WRITE( )
0085 -- WRITE( OS JOBS COM SEUS RESPECTIVOS SUCESSORES DIFERIS CIRCULADO
0086 -- S ENTRE PARENTESSES SAO )
0087 -- WRITE( 1, ( * SUCS(1), ) )
0088 -- FOR K =2 UNTIL N DO
0089 -- WRITE( )
0090 -- WRITE( )
0091 -- ACAN AS TAREFAS ABAIXO ESTAO LISTADAS EM ORDEM INVERSA EM REI
0092 -- ACAN AN SEU PROCESSAMENTO EM CADA MAQUINA )
0093 -- WRITE( )
0094 -- WRITE( )
0095 -- FOR K =1 UNTIL M DO
0096 -- BEGIN
0097 -- WRITE( OS JOBS DA MAQUINA * K, SAO )
0098 -- NUMO =TE4PK)
0099 -- WHILE NUMO =NULO DO
0100 -- BEGIN
0101 -- WRITE( )
0102 -- FND
0103 --3 FND
0104 -- WRITE( )
0105 -- WRITE( TEMPO TOTAL DE PROCESSAMENTO E .T)
0106 --1 FND.

```

EXECUTION OPTIONS DEVRG:1 TIME=5 SECONDS PAGES=5

002.65 SECONDS IN COMPILATION. (33660.01792) BYTES OF CODE GENERATED





```

0061 -6      END
0062 -6      ELSE
0063 -6      BEGIN
0064 -6      APOINT2 = LINK(APOINT)
0065 -6      LINK(APOINT) = NUVO
0066 -6      LINK(LINK(APOINT)) = APOINT2
0067 -6      END
0068 -6      ELSE
0069 -6      BEGIN
0070 -6      APOINT2 = APOINT
0071 -6      WHILE (APOINT2 = NUVO) AND (CRI(NUVO) CRI(APOINT2)) DO
0072 -6      BEGIN
0073 -6      APOINT3 = APOINT2
0074 -6      APOINT2 = LINK(APOINT2)
0075 -6      END
0076 -6      IF APOINT2 = NUVO THEN
0077 -6      BEGIN
0078 -6      LINK(APOINT3) = NUVO
0079 -6      LINK(LINK(APOINT3)) = NUVO
0080 -6      END
0081 -6      ELSE
0082 -6      BEGIN
0083 -6      IF (CRI(NUVO)) CRI(APOINT2) OR (TPRG(NUVO)) =
0084 -6      TPRG(APOINT2) THEN
0085 -6      BEGIN
0086 -6      LINK(APOINT3) = NUVO
0087 -6      LINK(LINK(APOINT3)) = APOINT2
0088 -6      END
0089 -6      ELSE
0090 -6      BEGIN
0091 -6      LINK(NUVO) = LINK(APOINT2)
0092 -6      LINK(APOINT2) = NUVO
0093 -6      END
0094 -6      END
0095 -6      END
0096 -6      END
0097 -6      END
0098 -6      END
0099 -6      END
0100 -6      END
0101 -6      END
0102 -6      END
0103 -6      END
0104 -6      END
0105 -6      END
0106 -6      END
0107 -6      END
0108 -6      END
0109 -6      END
0110 -6      END
0111 -6      END
0112 -6      END
0113 -6      END
0114 -6      END
0115 -6      END
0116 -6      END
0117 -6      END
0118 -6      END
0119 -6      END
0120 -6      END
0121 -6      END
0122 -6      END
0123 -6      END
0124 -6      END
0125 -6      END
0126 -6      END
0127 -6      END
0128 -6      END
0129 -6      END
0130 -6      END
0131 -6      END
0132 -6      END
0133 -6      END
0134 -6      END
0135 -6      END
0136 -6      END
0137 -6      END
0138 -6      END
0139 -6      END
0140 -6      END
0141 -6      END
0142 -6      END
0143 -6      END
0144 -6      END
0145 -6      END
0146 -6      END
0147 -6      END
0148 -6      END
0149 -6      END
0150 -6      END
0151 -6      END
0152 -6      END
0153 -6      END
0154 -6      END
0155 -6      END
0156 -6      END
0157 -6      END
0158 -6      END
0159 -6      END
0160 -6      END
0161 -6      END
0162 -6      END
0163 -6      END
0164 -6      END
0165 -6      END
0166 -6      END
0167 -6      END
0168 -6      END
0169 -6      END
0170 -6      END
0171 -6      END
0172 -6      END
0173 -6      END
0174 -6      END
0175 -6      END
0176 -6      END
0177 -6      END
0178 -6      END
0179 -6      END
0180 -6      END
0181 -6      END
0182 -6      END
0183 -6      END
0184 -6      END
0185 -6      END
0186 -6      END
0187 -6      END
0188 -6      END
0189 -6      END
0190 -6      END
0191 -6      END
0192 -6      END
0193 -6      END
0194 -6      END
0195 -6      END
0196 -6      END
0197 -6      END
0198 -6      END
0199 -6      END
0200 -6      END

```



```

0122 -- WRITE ( J )
0123 -- WRITE ( ABAIXO ESTAO OS JORN COM OS SEUS RESPECTIVOS VETORES DEMANDA )
0124 -- FOR I = 1 UNTIL N DO
0125 -- BEGIN
0126 -- WRITE ( I , )
0127 -- FOR J = 1 UNTIL T-1 DO
0128 -- WRITE ( KFC( I , J ) , )
0129 -- WRITE ( KFC( I , J ) , )
0130 -- END
0131 -- WRITE ( ) WRITE ( )
0132 -- WRITE ( ABAIXO ESTAO OS JORN E, ENTRE PARENTESIS, SEUS RESPECTIVOS TEMP
0133 -- OS DE INICIO E DE PROCESSAMENTO )
0134 -- WRITE ( )
0135 -- FOR I = 1 UNTIL N DO
0136 -- WRITE ( I , TIME( I ) , TIME( I ) , ) )
0137 -- WRITE ( ) WRITE ( )
0138 -- WRITE ( O TEMPO NECESSARIO PARA A CONCLUSAO DO PROJETO E .T )
0139 -- END
0140 -- END

```

EXECUTION OPTIONS DEBUG,1 TIME=5 SECONDS PAGES=5  
 002.01 SECONDS IN COMPIATION. (05052.02120) BYTES OF CODE GENERATED

B I B L I O G R A F I A

- 1 - AHO, A.V.; Hopcroft, J.E.; Ullman, J.D. - "The Design and Analysis of Computer Algorithms". Addison Wesley, Reading, Mass. (1974).
- 2 - Cook, S.A. - "The Complexity of Theorem - Proving Procedures". Proc. 3rd Annual ACM Symp. Theory Comp. (1971).
- 3 - Lenstra, J.K.; Rinnooy Kan, A.H.G. - "Complexity of Scheduling under Precedence Constraints". Ann. Discret Math. (sendo publicado).
- 4 - Lenstra, J.K.; Rinnooy Kan, A.H.G.; Brucker, P. - "Complexity of Machine Scheduling Problems". Annals of Discrete Math. (sendo publicado).
- 5 - Lawler, E.L. - "Optimal Sequencing of a Single Machine Subject to Precedence Constraints". Management Sci. (1973).

- 6 - Smith, W.E. - "Various Optimizers for Single-State Production". Naval Res. Logist. Quart. (1956).
- 7 - Sidney, J.B. - "Decomposition Algorithms for Single-Machine Sequencing with Procedure Relations and Deferral Costs". Operations Res. (1975).
- 8 - Karp, R.M. - "Reducibility Among Combinatorial Problems". Plenum Press (1972).
- 9 - Moore, J.M. - "An n Job, one Machine Sequencing Algorithms for Minimizing the Number of Late Jobs". Management Sci. (1968).
- 10 - Lawler, E.L. - "A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize Total Tardiness Ann. Discrete Math. (sendo publicado)".
- 11 - Bruno, J.; Coffman, E.G.; Sethi, R. - "Scheduling Independent Tasks to Reduce Mean Finishing Time". Comm. ACM (1974)
- 12 - Coffman, E.G.; Graham, R.L. - "Optmal Scheduling for Two-Processor Systems". Acta Informat. (1972).

- 13 - Ullman, J.D. - "NP-Complete Scheduling Problems".  
J. Comput. System Sci. (1975).
- 14 - Hu, T.C. - "Parallel Sequencing and Assembly Line Problems". Operations Research (1961).
- 15 - Lageweg, B.J.; Lenstra, J.K.; Rinnooy Kan, A.H.G.  
"Minimizing Maximum Lateness on one Machine:  
Computational Experience and Some Applications".  
Statistica Neerlandica (1976).
- 16 - Lawler, E.L. - "On Sequencing Problems with Deferral  
Costs" Management Sci. (1964).
- 17 - Lageweg, B.J.; Lawler, E.L. - "Private Communication"  
(1975).
- 18 - Johnson, S.M. - "Optimal Two-And Three-Stage Production  
Schedules with Setup times Included". Naval Res.  
Logist. Quart. (1954).
- 19 - Szwarc, W. - "Solution of the Akers-Friedman Scheduling  
Problem". Operations Res. (1960).
- 20 - Conway, R.W.; Maxwell, W.L.; Miller, L.W. - "Theory of  
Scheduling". Addison-Wesley Publishing Company  
(1967).

- 21 - Brucker, P.; Garey, M.R.; Johnson, D.S. - "Scheduling Equal-Length Tasks Under Treelike Precedence Constraints to Minimize Maximum Lateness" (sendo publicado).
- 22 - Kelley, Jr. - "Industrial Scheduling" - Prentice-Hall (1963).
- 23 - Mueller-Merbach, H. - "Experience with Methods for Resource Scheduling In CPM Networks" Zeitschrift, Fur Wirtschaftliche Fertigung (1967).
- 24 - Weist, J.D. - "Heuristic Programs For Decision Making" Harvard Business Review (1965).
- 25 - Davis, E.W. - "Project Scheduling Under Resource Constraints - Historical Review and Categorization of Procedures" - A Transactions (1973).
- 26 - McLaren, K.G.; Buesnel, E.L. - "Network Analysis in Project Management" Cox & Wyman LTD. (1968).
- 27 - Crwston, Wallace B.S. - "Decision Network Planning Models" - Management Sciences Research Report Nº 138 (1968).
- 28 - Reingold, E.M.; Nievergeet, J.; Deo, N. - "Combinatorial Algorithms: Theory and Practice" - Prentice-Hall, Inc., E. Cliffs (1977).



- 29 - Battersly, A. - "Network Analysis for Planning and Scheduling" - Macmillan and Co Limited (1971).
- 30 - Johnson, L.A.; Montgomery, D.C. - "Operations Research in Production Planning, Scheduling and Inventory Control" - John Wiley & Sons, Inc. (1974).
- 31 - Backer, K.R. - "Introduction to Sequencing and Scheduling" John Wiley and Sons. (1974).