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

PROBLEMAS "FLOW-SHOP" E "JOB-SHOP". COMPLEXIDADE DOS PROBLEMAS. MÉTODOS DE SOLUÇÃO.

M A R I A S O N I A N O G U E I R A

CAMPINA GRANDE, março de 1980

PROBLEMAS FLOW-SHOP E JOB-SHOP. COMPLEXIDADE DOS PROBLEMAS. MÉTODOS DE SOLUÇÃO.

M A R I A S O N I A N O G U E I R A

Tese submetida ao Corpo Docente do Programa de Pós-Graduação em Engenharia de Sistemas - Opção Pesquisa Operacional do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba, como requesito parcial para a obtenção do Grau de Mestre em Engenharia de Sistemas.

Aprovado por:

COMISSÃO EXAMINADORA

Prof. Ph.D. Peter Joachim Siegfried Brucker
Presidente

Prof. Ph.D. Hans Hermann Weber
Examinador

Prof. Dr. Rubens Leão de Andrade
Examinador

A meus pais: Samuel e Joanita

A meu esposo: Carlos Augusto

À minha sogra: M. Carmo

A G R A D E C I M E N T O S

A autora agradece:

À todos aqueles que contribuíram, de alguma forma, para a realização deste trabalho e em particular;

Ao seu orientador, Prof. Peter Brucker, pelo carinho e dedicação demonstrados durante todo o desenvolvimento do trabalho;

À Universidade Federal da Paraíba, especialmente ao corpo docente, pela importante função que desempenhou em parte de sua formação profissional;

Aos colegas e amigos Victor e Regina, pelo apoio, carinho e sugestões úteis ao aprimoramento deste trabalho;

À srta. Idelausi de B. Medeiros, pelo paciente e esmerado trabalho datilográfico;

À Secretária do Curso de Pós-Graduação em Engenharia de Sistemas da UFPb, Ana Lúcia, pelas inúmeras informações dadas sempre que solicitadas.

R E S U M O

Este trabalho consta de um estudo detalhado dos problemas "Flow-Shop" e "Job-Shop".

Algorítmos exatos são dados para estes problemas.

Dentre os algorítmos exatos, tem-se alguns algorítmos polinomiais para problemas muito particulares e alguns algorítmos enumerativos para problemas gerais. Estes problemas pertencem à classe NP-Completa, uma classe de problemas muito complicados e que agora não foram escritos algorítmos mais simples que os algorítmos mais simples que os algorítmos enumerativos para solucionar tais problemas.

Devido ao crescimento exponencial com o número de jobs e máquinas dos algorítmos enumerativos, outros algorítmos serão analisados. Estes são chamados Heurísticos, muito importantes na teoria "schedule" devido à simplicidade e viabilidade das soluções.

Finalmente, é feito um estudo sobre a pior solução de problemas "Flow-Shop" através de alguns métodos heurísticos.

A B S T R A C T

This work is about a study in details on "Flow-Shop" and "Job-Shop" problems.

Some of exact algorithms were studied for these problems.

Among these exact algorithms, we have some polynomial' algorithms for particular problems and some enumerative algorithms for general problems. These problems belong to the NP-Complete class. This is a class of problems very difficult for which we don't have simpler algorithms than those enumerative algorithms to solve such problems.

Due to exponencial growth with the number of jobs and machines of these enumeratives algorithms, some other algorithms ' will be analysed. These algoritms are called Heuristics and they are very important in "Schedule" theory due their simplicity and viability of solutions.

Finally a study has been done on the worst solution of "Flow-Shop" problems through some heuristic methods.

Í N D I C E

I N T R O D U Ç Ã O

Os problemas "Flow-Shop"[1] e "Job-Shop"[2] são temas de destaque neste trabalho. Estes problemas são dos mais importantes na teoria "scheduling"[3] como também dos primeiros a serem discutidos. A grande importância destes, se deve ao fato da aplicabilidade dentro do processo industrial, preocupação natural dos últimos tempos. Johnson [13] foi o primeiro cientista a dar resultados concretos deste problemas.

Um problema "Flow-Shop" envolve um conjunto de máquinas e

1,2 - Os termos "Flow-Shop" e "Job-Shop" são utilizados durante todo o contexto por falta de uma palavra em português que os traduza adequadamente.

3 - O termo "scheduling" é utilizado por falta de uma palavra que o traduza de maneira completa. Durante todo este trabalho, ele é definido como alocação de máquinas sobre tempo para executar uma coleção de tarefas.

"Jobs"[4]. Cada "Job" é formado de tarefas, sendo cada uma delas executada por uma diferente máquina. É importante salientar que em um problema "Flow-Shop" o conjunto de máquinas deverá ser ordenado e que o número de cada tarefa dos "Jobs" coincida com o número da máquina que a executará. Uma possível aplicabilidade do problema "Flow-Shop" é na Indústria Automobilística.

Define-se "Job-Shop" como um conjunto de "Jobs" que devem ser executados por máquinas. Cada "Job" é composto de tarefas, as quais serão processadas pelas máquinas. Aqui pode-se ter um número de tarefas diferente do número de máquinas podendo ocorrer diferentes sequências no processamento dos "Jobs" pelas máquinas. Por isso "Job-Shop" é um problema mais geral que "Flow-Shop". Uma possível aplicação para ele é em oficinas para reparo de carros.

O capitulo I deste trabalho, trata dos problemas "Flow-Shop" e "Job-Shop", definições das diversas variáveis que os compõem, como também, do diagrama de Gantt, utilizado na representação gráfica destes problemas.

Nos capítulos 2 e 3 serão desenvolvidos algorítmos polinomiais para a determinação de uma "schedule" ótima para alguns problemas "Flow-Shop" e "Job-Shop". Entretanto, estes algorítmos só se verificam para problemas muito específicos.

No capítulo 4 trabalha-se com algorítmo "Branch and Bound" para os problemas "Flow-Shop" e "Job-Shop". Estes algorítmos são algorítmos enumerativos, por isso, computacionalmente, não são aconselháveis, pois exigem muita memória e tempo de computação. Em geral

---

4 - Neste trabalho o termo "Job" representa um conjunto de tarefas ou operações que deverão ser processadas por máquinas.

eles crescem exponencialmente com o número de jobs e máquinas.

Existe uma classe de problemas chamados problemas NP-Completos muito complicados e que serão tratados no capítulo 5. Estes problemas são todos polinomialmente equivalentes, isto é, se exisitr um algorítmo polinomial para um destes problemas, é possível construir algorítmos polinomiais para outros problemas NP-Completos. Durante muitos anos, cientistas tentaram escrever algorítmos melhores para solucionar os problemas desta classe não alcançando grande sucesso, por isso eles pensam da não existência de tais algorítmos. Também será mostrado neste capítulo, que problemas "Flow-Shop" e "Job-Shop" mais gerais pertence a esta classe e, daí, provavelmente os melhores algorítmos para solucionar vários problemas sejam os algorítmos enumerativos.

O capítulo 6, trata de algorítmos heurísticos de grande importância na teoria "scheduling" devido à sua simplicidade e viabilidade nas soluções, especialmente para problemas NP-Completos.

No capítulo 7, analisa-se o pior caso para algorítmo de problemas "Flow-Shop" através de alguns métodos heurísticos.

CAPÍTULO I

PROBLEMAS "FLOW-SHOP" E "JOB-SHOP"

1.1 - O PROBLEMA "FLOW-SHOP"

Define-se "Flow-Shop" como sendo um conjunto de n jobs $J_1,\ldots,J_n$ e um conjunto ordenado de m ($m \geq 1$) máquinas $M_1,\ldots,M_m$. Cada job consiste de m operações, de modo que a i-ésima operação de cada job seja executada pela máquina M_i. Notando-se por O_{ji} a i-ésima operação do job J_j, t_{ji} representa o tempo de processamento desta operação na máquina M_i.

Uma "schedule" é definida como sendo uma função X que especifica o tempo de início de cada operação. Se x_{ji} é o tempo de início do job J_j na máquina M_i, isto é, o tempo de início da i-ésima operação do job J_j, então as propriedades seguintes devem se verificar numa "schedule";

a) Cada operação deve ser processada ininterruptamente;

b) Uma máquina não pode processar, em um

mesmo instante, mais de uma operação;

c) O processamento de uma operação $O_{j,i+1}$ só poderá ser iniciado após concluída a execução da operação O_{ji}.

Formalmente, uma "schedule" pode ser definida pela seguinte função:

$$X: O_{ji} \rightarrow x_{ji} \quad \text{com}$$

1. $x_{j,i+1} \geq x_{ji} + t_{ji}$

2. $[x_{ji}, x_{ji} + t_{ji}) \cap [x_{ki}, x_{ki} + t_{ki}) = \emptyset . k \neq j$

Exemplo 1.1 - Este problema será ilustrado com um exemplo onde m=3, n=2 e com os valores t_{ji} dados na tabela 1.1 abaixo.

M_j	t_{1i}	t_{2i}	
M_1	1	5	
M_2	5	1	
M_3	5	3	

Tabela 1.1

Determina-se uma "schedule" para este problema mediante uma representação gráfica conhecida por diagrama, de Gantt que é a alocação de máquinas sobre o tempo.

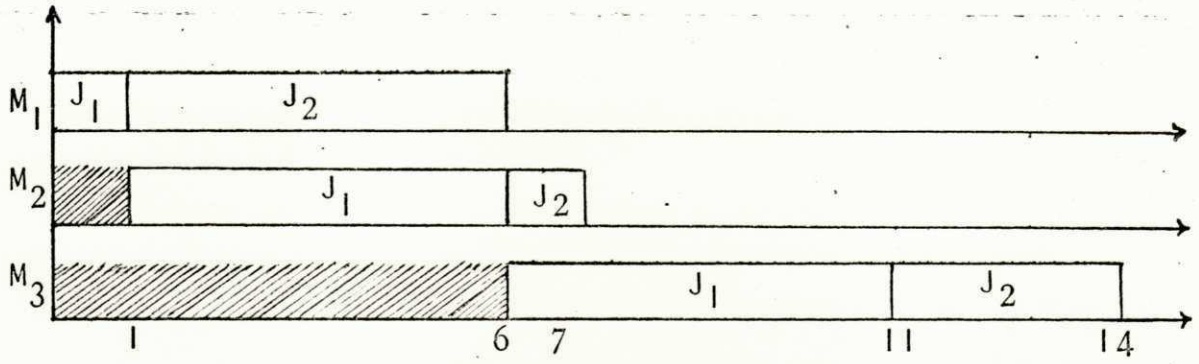


Fig. 1.1 - Uma possível "schedule" para o exemplo 1.1

Na determinação de uma "schedule" para um problema "Flow Shop", tem-se em mente determinados objetivos que serão executados de acordo com o interesse deste problema. Antes de explanar-se estes objetivos (funções objetivas), serão vistas algumas variáveis adicionais a este problema.

r_j, $j=1,\ldots,n$ - tempo em que o job J_j está disponível para ser processado.

d_j, $j=1,\ldots,n$ - tempo previsto para a conclusão do job J_j.

C_j, $j=1,\ldots,n$ - tempo de conclusão do job J_j, isto é, tempo em que a última operação do job J_j foi concluída; assim $C_j=x_{jn}+t_{jn}$.

w_j, $j=1,\ldots,n$ - peso atribuído ao job J_j de acordo com sua importância. Neste trabalho serão es

tudados os casos particulares $w_j=1$ e $w_j=1/n$.

As quantidades que avaliam uma "scheduling" são usualmente funções de C_j. Serão vistas algumas delas:

L_j, $j=1,\ldots,n$ - estabelece a diferença entre o tempo de conclusão e o tempo previsto de conclusão do Job J_j, isto é, $L_j=C_j-d_j$.

Assim, $L_j>0$ sempre que o Job J_j for concluído após o tempo previsto. $L_j=0$ sempre que o Job J_j for concluído antes do tempo previsto e $L_j<0$ sempre que o Job J_j for concluído antes do tempo previsto.

T_j, $j=1,\ldots,n$ - Detecta de quanto o Job J_j atrasou seu processamento. $T_j=\max\{0, C_j-d_j\}$. É claro que $T_j=0$ sempre que o Job J_j for processado antes ou na data prevista.

U_j, $j=1,\ldots,n$ - Detecta quando o Job J_j atrasou no seu processamento. $U_j=\begin{cases}0 & \text{se } C_j\leq d_j\\ 1 & \text{caso contrário}\end{cases}$

Com base nestas variáveis, pode-se agora destacar as funções objetivas de maior interesse para os problemas "Flow-Shop":

i) $f(C_1,\ldots,C_n)=\max_{j=1}^{n}\{C_j\}=C_{max}$. Nesta função, tem-se como propósito, minimizar o maior tempo de conclusão de todos os Jobs.

ii) $f(C_1,\ldots,C_n)=\max_{j=1}^{n}\{L_j\}=L_{max}$. Objetiva-se aqui, mi

nimizar a maior diferença ocorrida no processamento dos jobs.

iii) $f(C_1,\ldots,C_n) = \sum_{j=1}^{n} w_j C_j$. Partindo-se do princípio de que w_j representa o custo proporcional ao tempo de conclusão de cada job J_j, tem-se como objetivo para esta função, minimizar o custo total de processamento.

iv) $f(C_1,\ldots,C_n) = \sum_{j=1}^{n} w_j T_j$. Considerando-se que w_j representa o custo proporcional ao atraso ocorrido em cada job J_j, tem-se como objetivo para esta função, minimizar o custo total de penalidade pelo atraso de todos os jobs.

v) $f(C_1,\ldots,C_n) = \sum_{j=1}^{n} w_j U_j$. Tomando-se w_j como custo por atraso do job J_j, então tem-se como objetivo para esta função, minimizar o custo fixo total de atraso. Se $w_j=1$, $\forall_j$, então $\sum_{j=1}^{n} w_j U_j$ representa o número de jobs atrasados.

Exemplo 1.2 - Dar um exemplo para cada uma destas funções, utilizando os dados do exemplo 1.1 e a "schedule" da figura 1.1.

Obs: Nada garante que a "schedule" da figura 1.1 é ótima para cada função objetiva abaixo, entretanto, objetiva-se neste exemplo, apenas mostrar como utilizar estas funções. Para a determinação do ótimo, tem-se que usar algorítmos ou encontrar todas as possíveis "schedules" para então escolher-se a melhor.

Analisando-se a "schedule" da figura 1.1, tem-se:

i) $f(C_1,C_2)= \max\{C_1,C_2\}=14$ que representa o tempo mínimo de conclusão após o processamento dos jobs.

Para as funções objetivas seguintes, toma-se como tempo previsto para a conclusão dos jobs $d_1=12$ e $d_2=13$.

ii) $f(C_1,C_2)= \max_j \{L_j\} = 1$ que representa o maior atraso

ocorrido entre os jobs.

iii) Considerando-se que $w_1=w_2=1$ representa o custo proporcional ao tempo de conclusão de cada job, então $f(C_1,C_2)= \sum_{j=1}^{2} w_j C_j=25$ que representa o custo total de processamento dos jobs J_j e J_2.

iv) Considerando-se agora que o custo proporcional ao atraso pelo job J_1 é $w_1=2$ e pelo job J_2 é $w_2 = 3$ então $f(C_1,C_2) = \sum_{j=1}^{2} w_j T_j =3$.

v) Tomando-se $w_1=1$ como custo fixo por atraso do job J_1 e $w_2 = 3$ do job J_2, então $f(C_1,C_2 = \sum_{j=1}^{2} w_j U_j=3$ representa o custo fixo total pelo atraso dos jobs.

Por simplicidade, os problemas "Flow-Shop" serão representados na forma n/m/F/1/f onde:

n = representa o número de jobs.
m = representa o número de máquinas.
F = por se tratar de um problema "Flow-Shop"
1 = representa o valor de r_j.
f = representa a função objetiva em questão.

Exemplo 1.3 - Um problema "Flow-Shop" com 3 jobs, 2 máquinas, com os jobs disponíveis para serem processados no tempo 0 (zero) e com objetivo C_{max}, pode ser representado do seguinte modo : $3/2/F/r_j= 0/C_{max}$ para j=1,2,3.

Na secção seguinte, serão tratados problemas mais gerais, denominados problemas "Job-Shop".

****

1.2 - O PROBLEMA "JOB-SHOP"

Define-se "Job-Shop", como sendo um conjunto de n jobs $J_1,\ldots, J_n$ que devem ser processados por m máquinas $M_1,\ldots, M_m$. O job J_j $(j=1,\ldots,n)$ consiste de n_j operações O_{jki}, onde O_{jki} representa a k-ésima operação $(k=1,\ldots,n_j)$ do j-ésimo job que deve ser executada pela i-ésima máquina. Cada operação O_{jki} requer um tempo de processamento t_{jki}.

Uma "schedule" para um problema "Job-Shop" é definida do mesmo que para "Flow-Shop".

Nota-se por x_{jki} o tempo de início da k-ésima operação do job J_j na i-ésima máquina. Formalmente uma "schedule" para um problema "Job-Shop" pode ser definida pela função.

$$X : O_{jki} \longrightarrow x_{jki} \qquad \text{com}$$

1. $x_{j,k+1,j} \geq x_{jkl} + t_{jkl}$

2. $[x_{jki},\ x_{jki} + t_{jki}) \cap [x_{lhi},\ x_{lhi} + t_{lhi}) = \emptyset$

para diferentes operações O_{jki} e O_{lhi}.

O problema "Job-Shop" será ilustrado com o exemplo 1.2 a seguir:

Exemplo 1.2 - As tabelas 1.2 a seguir determinam os dados para um problema "Job-Shop" de 4 jobs, 3 máquinas e os seguintes números de operações:

$$n_1 = 5, \quad n_2 = 2, \quad n_3 = 4, \quad n_4 = 3$$

Tabela 1.2

Job \ Oper.	1	2	3	4	5
J_1	2	1	5	3	1
J_2	2	3	-	-	-
J_3	1	3	1	2	-
J_4	4	3	5	-	-

(a) Tempo de processamento das operações.

Job \ Oper.	1	2	3	4	5
J_1	1	3	2	3	1
J_2	2	3	-	-	-
J_3	2	1	3	2	-
J_4	3	2	1	-	-

(b) Máquinas onde as operações serão executadas.

Nota: Uma maneira mais formal e simplificada de representar-se os dados deste exemplo seria:

$t_{111} = 2$, $t_{123} = 1$ $t_{132} = 5$ $t_{143} = 3$ $t_{151} = 1$

$t_{212} = 2$, $t_{223} = 3$

$t_{312} = 1$, $t_{321} = 3$ $t_{333} = 1$, $t_{342} = 2$

$t_{413} = 4$, $t_{422} = 3$, $t_{431} = 5$

Determina-se uma "schedule" para este problema - exemplo mediante o diagrama de Gantt.

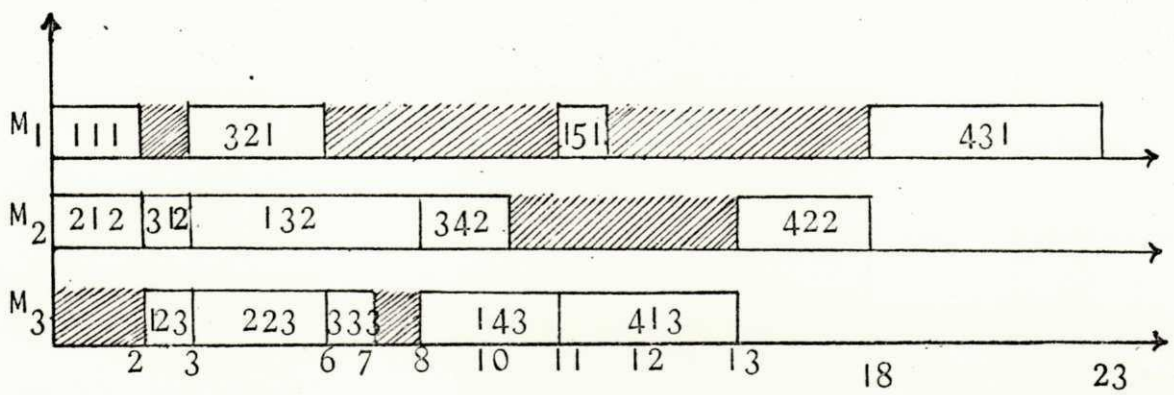


FIGURA 1.2 - Uma possível "schedule" para o exemplo 1.2

As funções objetivas e demais variáveis de um problema "Job-Shop" são definidas como no "Flow-Shop".

Os problemas "Job-Shop" podem ser expressos na forma n/m/J/I/f onde n, m, f são definidos como no "Flow-Shop", J indica que se trata de um problema "Job-Shop" e I pode representar restrições para r_j, n_j e t_{jki}.

Nota: Por simplicidade de notação podería-se usar em cada bloco do diagrama de Gantt, apenas o número do job.

C A P Í T U L O II

ALGORÍTMOS POLINOMIAIS PARA PROBLEMAS "FLOW-SHOP"

2.1 - "SCHEDULES" PERMUTAÇÃO

A maioria dos "papers" (1) que desenvolvem algorítmos para problemas "Flow-Shop", evidenciam as "schedules" permutação, isto é, "schedules" com a mesma sequência de jobs em todas as máquinas, devido à sua participação em busca da solução ótima. Entretanto, serão vistos através de exemplos, que nem sempre estas "schedules" dão uma solução ótima, a menos que se tratem de problemas muito específicos, os quais serão discutidos nos exemplos e teoremas seguintes.

Exemplo 2.1 - Verificar se uma schedule ótima para o problema $2/4/F/r_j=0/C_{max}$ com os dados da tabela 2.1 a seguir, ocorre ' em uma "schedule" permutação.

---

(1) trabalhos publicados.

Tabela 2.1

M_i	t_{1i}	t_{2i}
M_1	2	8
M_2	8	2
M_3	8	2
M_4	2	8

Algumas das possíveis "schedules" para o problema são dadas a seguir:

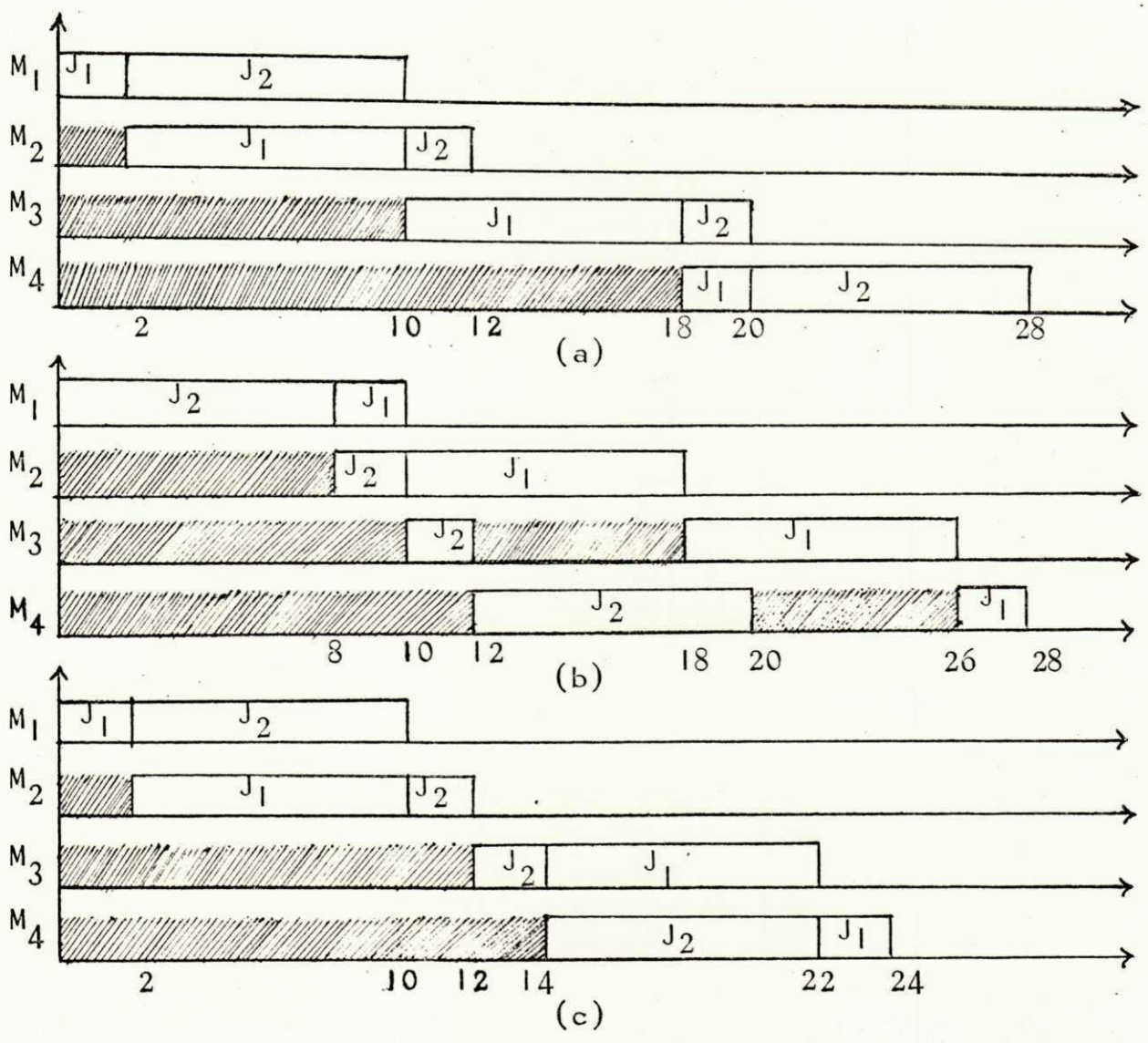


FIGURA 2.1

Portanto na figura 2.1(c), tem-se uma "schedule" melhor que as duas possíveis "schedules" permutação mostradas nas figuras 2.1(a) e (b).

TEOREMA 2.1 - Para minimizar-se um problema do tipo n/m/F/I/f onde f é qualquer função objetiva definida no capítulo I, basta considerar as "schedules" nas quais a mesma sequência de jobs ocorrerá nas máquinas M_1 e M_2.

Prova: Se uma "schedule" não tem a mesma sequência de jobs nas duas primeiras máquinas, então existirão dois jobs J_k e J_l de modo que o job J_l é imediatamente posterior ao Job J_k na máquina M_1 enquanto o job J_k é posterior ao job J_l (possivelmente com jobs intermediários) na máquina M_2 (ver figura 2.2)

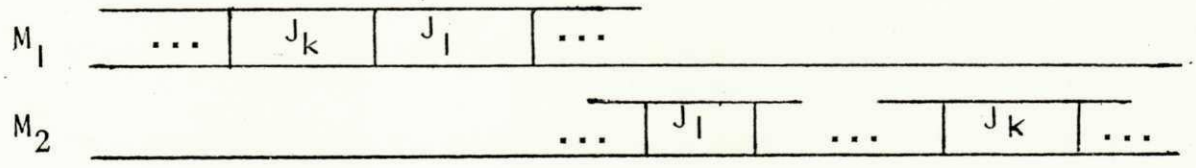


FIGURA 2.2

É óbvio que se pode trocar a ordem destes dois jobs, na máquina M_1 sem acarretar qualquer acréscimo no tempo de início dos jobs na máquina M_2 e consequentemente nas máquinas subsequentes. Deste modo, não haverá acréscimo no tempo de conclusão dos jobs e portanto em qualquer função objetiva.

Observação: Na realidade o que poderá ocorrer após a troca, é um decrescimento no tempo de início dos jobs da máquina M_2 daí, nas máquinas subsequentes e portanto, um decrescimento na função objetiva. Este fato é ilustrado pela figura 2.3.

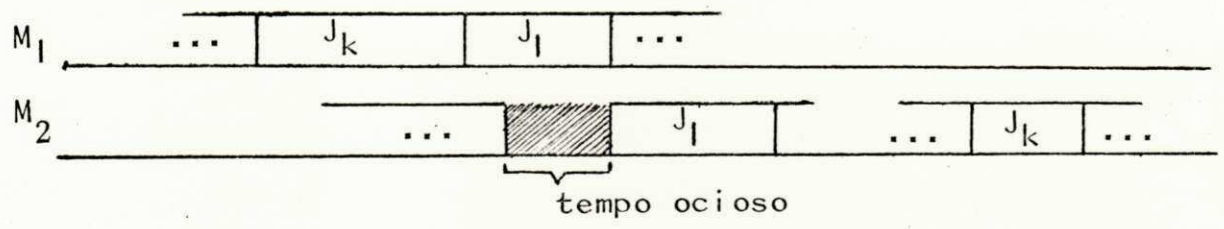


FIGURA 2.3

TEOREMA 2.2 - Em um problema do tipo $n/m/F/I/C_{max}$, basta considerar-se as "schedules" com a mesma ordenação dos jobs nas máquinas M_{m-1} e M_m.

Prova: Se uma "schedule" não tem a mesma sequência de jobs sobre as duas últimas máquinas, é porque existirão dois jobs J_k e J_l com J_k imediatamente posterior a J_l na máquina M_m enquanto o job J_l é posterior do job J_k (possivelmente com jobs intermediários) na máquina M_{m-1} (ver figura 2.4).

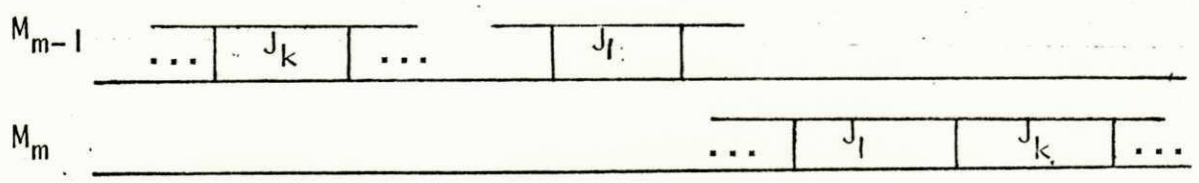


FIGURA 2.4

Obviamente a posição destes dois jobs pode ser trocada na máquina M_m de modo que o max $\{C_k, C_l\}$ não seja aumentado e consequentemente o tempo de conclusão dos demais jobs.

Observação: Na realidade o que poderá ocorrer após a troca é uma diminuição em max $\{C_k, C_l\}$ e consequentemente uma diminuição no valor C_{max}. Uma justificativa para este último fato é dada na figura 2.5 a seguir:

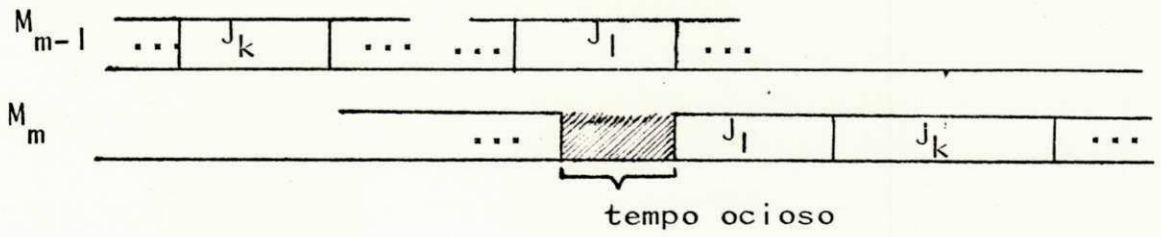


FIGURA 2.5

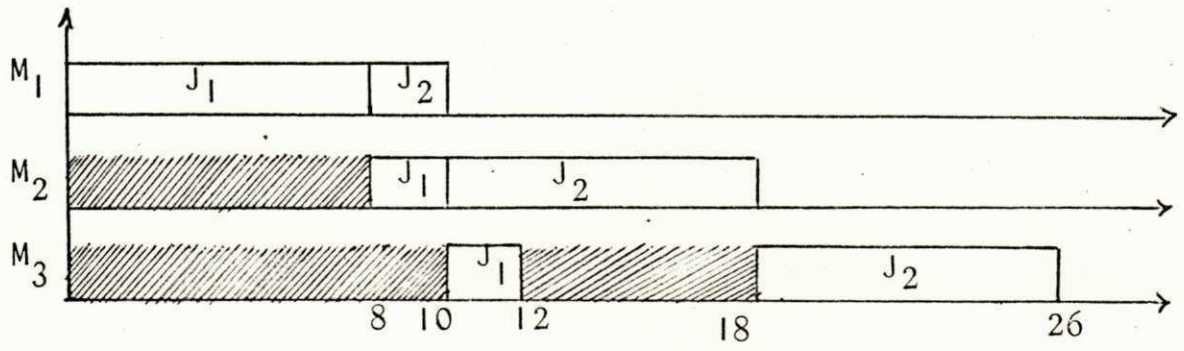
No exemplo 2.1 viu-se que uma "schedule" ótima para um problema "Flow-Shop" de 4 máquinas e com objetivo C_{max}, pode não ser uma "schedule" permutação. Portanto, da união dos teoremas 2.1 e 2.2, conclui-se que o valor C_{max} mínimo, ocorre em uma "schedule" permutação quando tratar-se de um problema "Flow-Shop" com $m \leq 3$.

Quando tratar-se de um problema de 3 máquinas com função objetiva diferente de C_{max}, a "schedule" ótima também poderá não ser uma "schedule" permutação. Para ilustrar este fato, desenvolve-se o exemplo abaixo:

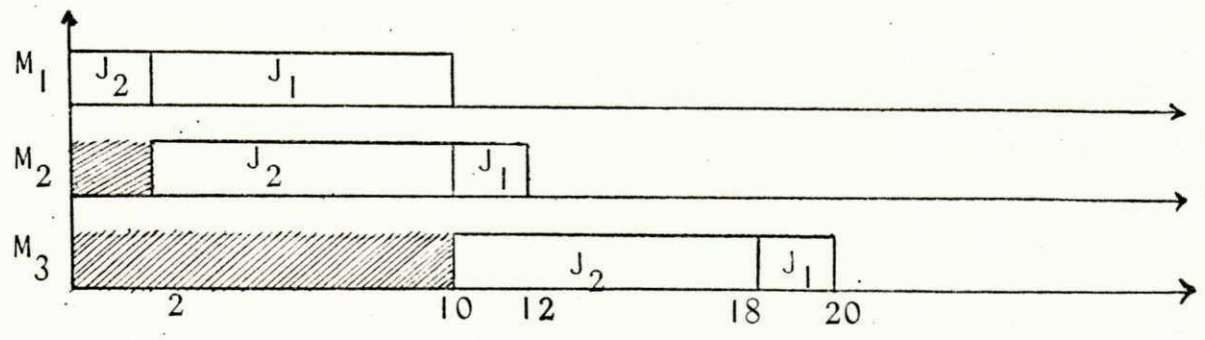
Exemplo 2.2 - Verificar se o problema $2/3/F/r_j=0/\sum_{j=1}^{2} C_j$.

com os dados da tabela 2.2 abaixo tem solução ótima em uma "schedule" permutação.

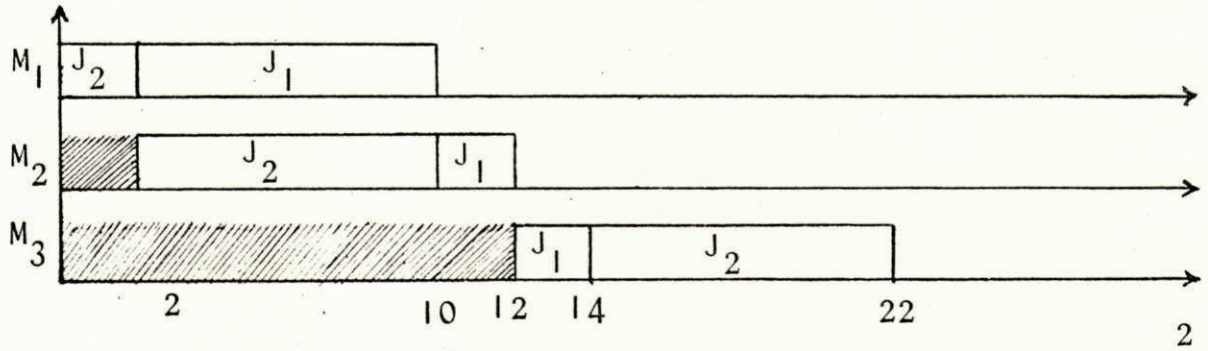
M_i	t_{1i}	t_{2i}
M_1	8	2
M_2	2	8
M_3	2	8



(a) "Schedule" permutação $\sum_{j-1}^{2} C_j = 38$



(b) "Schedule" permutação $\sum_{j-1}^{2} C_j = 36$



(c) "schedule" que não é uma "schedule" permutação. $\sum_{j=1}^{2} C_j = 36$

FIGURA 2.6

Viu-se portanto, que a "schedule" da figura 2.6(c) é melhor que as duas possíveis "schedules" permutação da figura 2.6(a) e (b).

Do Teorema 2.1 e exemplo 2.2, conclui-se que a solução ótima de um problema com uma função objetiva qualquer só ocorre em uma "schedule" permutação quando tratar-se de um problema com m=2.

****
****
****

2.2 - O PROBLEMA $n / 2 / F / r_j = 0 / C_{max}$

Johnson [14] foi o primeiro cientista a dar resultados concretos para problemas "Flow-Shop". Ele escreveu um algorítmo, através do qual, determina-se uma sequência ótima para os jobs de um problema de 2 máquinas e n jobs na ordem de nlogn passos, daí, ser chamado algorítmo polinomial, isto é, um algorítmo cuja ordem de complexidade (número de passos) é limitada por um polinômio que depende do número de jobs.

ALGORÍTMO 2.1 (ALGORÍTMO DE JOHNSON)

Os dados e variáveis deste algorítmo são os seguintes:

n - Número de jobs

$X = \{J_1,\ldots,J_n\}$ - Conjunto de jobs do problema

t_{ji}, $j=1,\ldots,n$; $i=1,2$ - representa o tempo de processamento requerido pelo job J_j na máquina M_i.

S_1 - Sequência de jobs cujo tempo de processamento na máquina M_1 é menor que o tempo de processamento na máquina M_2. Em caso contrário, tem-se a sequência S_2

$L = S_1S_2$ - representa a sequência ótima para o problema.

O desenvolvimento deste algorítmo se dá do seguinte modo:

Procura-se o mínimo tempo de processamento de todas as

operações dos jobs. Se este valor ocorrer em uma operação executada pela máquina M_1 respectivamente M_2, então o job em questão será o primeiro elemento da pesquisa S_1 respectivamente S_2 e elimina-se este job do conjunto X. Prossegue-se com a pesquisa até eliminar-se todos os jobs de X e tal que cada job em S_1 é colocado à direita dos demais e em S_2 à esquerda. A sequência ótima para o problema é a sequência L formada pela concatenação das sequências S_1 e S_2 na ordem em que se encontram.

1. (Inicialização) $X \leftarrow \{J_1, \ldots, J_n\}$; $S_1 \leftarrow \emptyset$; $S_2 \leftarrow \emptyset$;

2. While $X \neq \emptyset$ do

begin

3. if $t_{j^*1} = \min \{ t_{ji} / J_j \in X, i=1,2 \}$ then

4. $S_1 \leftarrow S_1 \; J_{j^*}$

else

5. if $t_{j^*2} = \min \{ t_{ji} / J_j \in X, i=1,2 \}$ then

6. $S_2 \leftarrow J_{j^*} \; S_2$;

7. $X \leftarrow X \setminus \{ J_{j^*} \}$

8. end;

9. $L \longleftarrow S_1 S_2$

10. end alg. 2.1.

Para efeito de programação, é fácil notar que este algorítmo não está definido de maneira única porque nos passos 3 e 5 podem ocorrer que o valor $\min \{ t_{ji} / J_j \in X, i = 1,2 \}$ não seja único. Para torná-lo único, certas regras de prioridade deverão ser introduzi

das no programa, como por exemplo:

i) Se $t_{j^*i} = t_{k^*i} = \min \{ t_{ji} / J_j \in X;\ i=1,2 \}$ com $j^* < k^*$, processar J_{j^*} antes de J_{k^*}

ii) Se $t_{j^*i} = t_{j^*2}$ colocar o job J_j no início da sequência S_2.

A ordem de complexidade para este algorítmo é nlogn considerando-se que os valores t_{ji} devem ser ordenados. Existe uma estrutura de dados que executa o while-loop deste algorítmo na O(n) [1].

Exemplo 2.3 - Utilizando-se do algorítmo 2.1, encontrar uma sequência ótima para o problema de 2 máquinas e 5 jobs cujos dados estão na tabela 2.3 a seguir:

tabela 2.3

op / maq	J_1	J_2	J_3	J_4	J_5
M_1	4	3	3	1	8
M_2	8	3	4	4	7

Nota: Determinar a sequência ótima com base nas regras de prioridade descritas após o algorítmo 2.1.

$X \leftarrow \{J_1, J_2, J_3, J_4, J_5\}$; $S_1 \leftarrow \emptyset$; $S_2 \leftarrow \emptyset$;

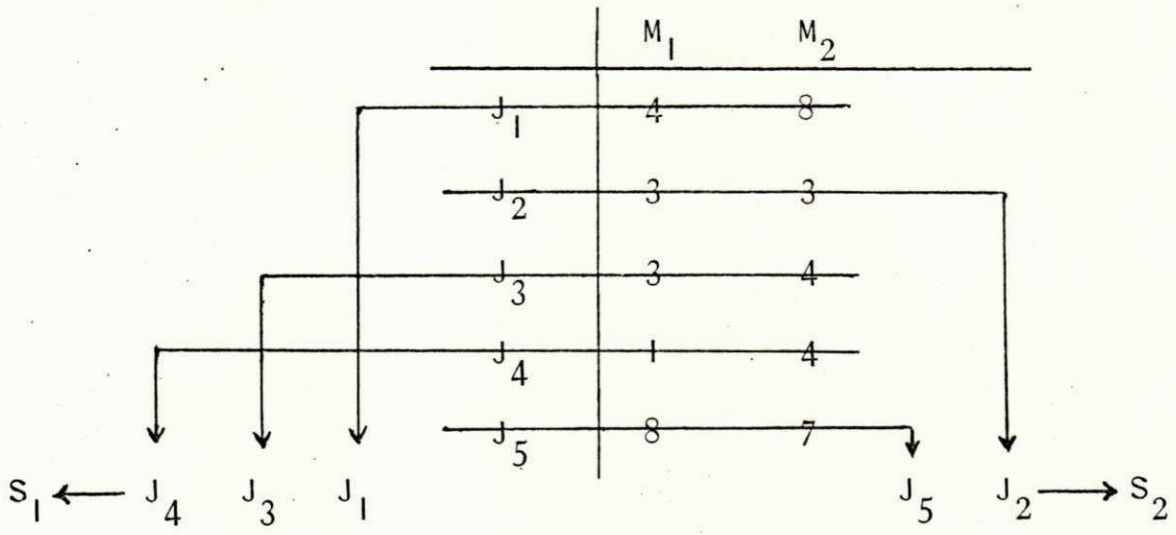
$t_{j^*i} = \min t_{ji} = t_{41}$. Como i=1 então colocar o Job $J_{j^*} = J_4$ na primeira posição da sequência, isto é, $S_1 \leftarrow J_4$.

Agora $X \leftarrow \{J_1, J_2, J_3, J_5\}$

$$t_{j^*i} = \min \{ t_{ji}, J_j \in X, \quad i=1,2 \} = t_{22}.$$

Como i=2 então colocar o job $J_{j^*}=J_2$ na última posição da sequência, S_2, isto é, $S_2 \leftarrow J_2$. Neste passo foram aplicadas as duas regras de prioridade.

Agora $X \leftarrow \{J_1, J_3, J_5\}$. $t_{j^*i} = \min \{t_{ji}, J_j \in X, i=1,2\} = t_{31}$.
Como i=1 então colocar o job $J_{j^*}=J_3$ à direita do Job J_4, logo $S_1 \leftarrow J_4 J_3$
Continuando-se com o processo, tem-se a sequência L abaixo:



$L = S_1 S_2 = J_4 J_3 J_1 J_5 J_2$ cuja representação gráfica será:

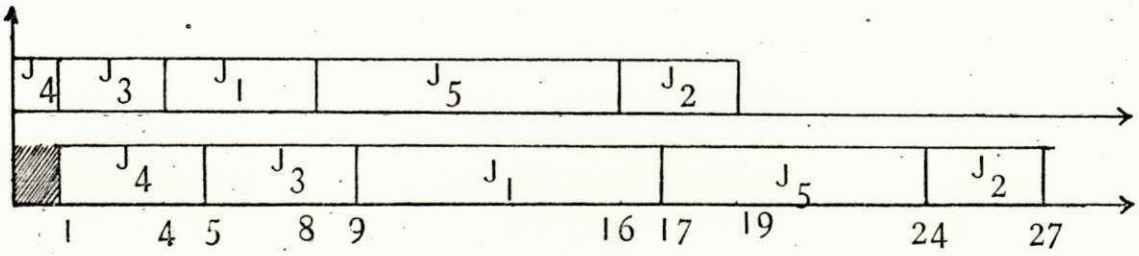


FIGURA 2.7

Gupta, após analisar o algorítmo de Johnson(alg.2.1), verifi - cou que era possível descrevê-lo do seguinte modo:

ALGORÍTMO 2.2

// Dados: os mesmos do algorítmo 2.1 //

// Variáveis: $S(J_j)$ - uma quantidade relativa ao job J_j e a partir da qual será determinada a posição do job J_j na se- quência ótima //.

1. (Iniciação) $X \leftarrow \{J_1, \ldots, J_n\}$;
2. For j=1 to n do
   begin
3. if $t_{j*1} < t_{j*2}$ then
4. $S(J_{j*}) \leftarrow 1/t_{j*1}$
   else
5. $S(J_{j*}) \leftarrow -1/t_{j*2}$;
6. end;
7. Ordenar J_j de maneira não crescente dos $S(J_j)$;
8. end alg 2.2 .

Este algorítmo também não está definido de maneira única, pois pode ocorrer o caso de $S(J_k)=S(J_l)$, $k \neq l$ e J_k, $J_l \in X$. Para efeito de programação, colocar J_k antes de J_l somente quando $k < l$.

Obviamente este algorítmo determina a mesma sequência do algorítmo 2.1.

A ordem de complexidade deste algorítmo é também nlogn. O do-loop de 2 até 6 é executado na O(n) e o passo 7 na ordem nlogn [1].

Exemplo 2.4 - Aplicando-se o algorítmo 2.2 nos dados da tabela 2.3, tem-se:

Como $t_{11} < t_{12}$ então $S(J_1)= \frac{1}{t_{11}} = \frac{1}{4}$. Os outros valores de $S(J_j)$ são:

$S(J_2) = -1/3$

$S(J_3) = 1/3$

$S(J_4) = 1$

$S(J_5) = -1/7$

Ordenando-se os valores de $S(J_j)$ de maneira não crescente, tem-se a sequência ótima $J_4J_3J_1J_5J_2$ que é a mesma obtida pelo algorítmo 2.1.

Agora, a otimalidade das sequências obtidas pelo algorítmo 2.1 serão provadas mediante os lemas e teorema seguintes:

Lema 2.1 - seja $L=L(J_1)\ldots L(J_n)$ a lista (sequência) construída pelo algorítmo 2.1. Se $\min\{t_{i1}, t_{j2}\} < \min\{t_{j1}, t_{i2}\}$ então o job J_i precede o job J_j na lista (sequência) L.

Prova: A prova deste lema consiste de dois casos:

i) $t_{i1} < \min\{t_{j1}, t_{i2}\}$ então $t_{i1} < t_{i2}$ logo J_i está em S_1, como $t_{i1} < t_{j1}$ então J_i precederá J_j. Se J_j estiver em S_2, como a

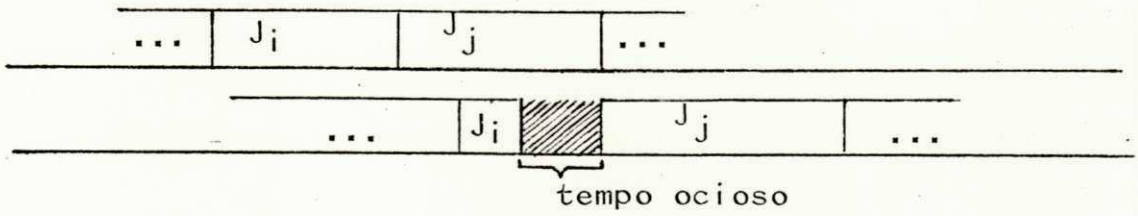
sequência dos jobs de S_2 são colocados à direita da sequência dos jobs de S_1 na sequência L então obviamente J_i precederá J_j.

ii) $t_{j2} \leq \min \{t_{ji}, t_{i2}\}$ então $t_{j2} < t_{ji}$ logo J_j está em S_2. Se J_i também estiver em S_2, como $t_{j2} < t_{i2}$ então J_i é predecessor de J_j. Se J_i estiver em S_1 por maior razão J_i precede J_j visto que a sequência S_i precede a sequência S_2 em L.

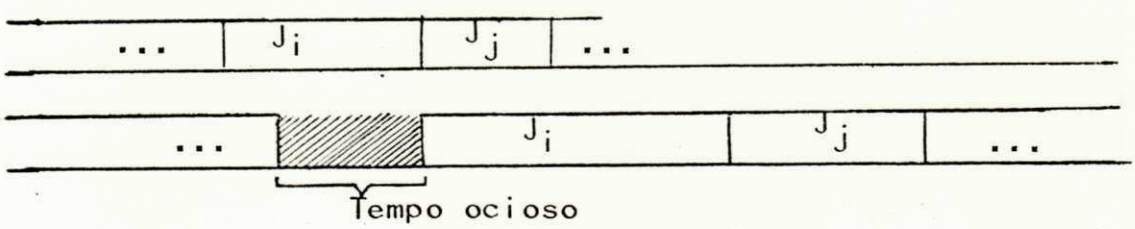
Lema 2.2 - Seja M uma lista (sequência) arbitrária e J_j sucessor imediato de J_i em k.

Se $\min \{t_{ji}, t_{i2}\} \leq \min \{t_{i1}, t_{j2}\}$ então é possível trocar a ordem de J_i e J_j sem crescer o valor da função objetiva.

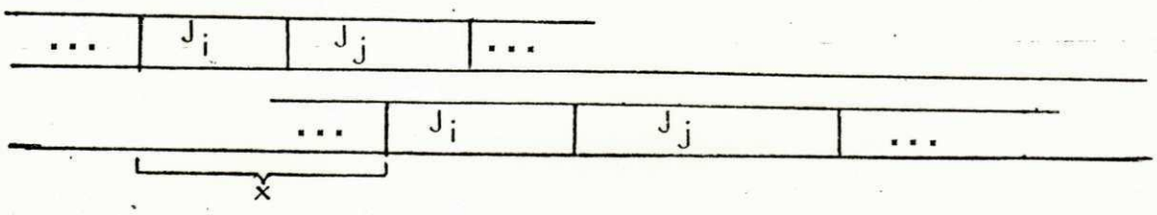
Prova: Se J_j é um sucessor de J_i então um dos seguintes casos poderá ocorrer.



(a)



(b)



(c)

FIGURA 2.8

Portanto o tempo de processamento W_{ij} dos jobs J_i e J_j se J_j é um sucessor de J_i é:

$$Wij = \max \{t_{i1} + t_{j1} + t_{j2},\ t_{i1} + t_{j2},\ x + t_{i2} + t_{j2}\} =$$

$$= \max \{t_{i1} + t_{j2} + \max \{t_{j1},\ t_{i2}\},\ x + t_{i2} + t_{j2}\}.$$

Similarmente,

$W_{ji} = \max \{t_{j1} + t_{i2} + \max \{t_{i1},\ t_{j2}\},\ x + t_{i2} + t_{j2}\}$. Vê-se que em ambos os casos, x independe dos jobs J_i , J_j .

Da condição do lema,

$\min \{t_{j1},\ t_{i2}\} \leq \min \{t_{i1},\ t_{j2}\}$ logo, $\max \{-t_{j1} - t_{i2}\} \geq \max \{-t_{i1},\ -t_{j2}\}$. Adicionando-se aos dois membros $t_{i1} + t_{i2} + t_{j1} + t_{j2}$, obtem-se:

$$t_{ji} + t_{i2} + \max \{t_{i1},\ t_{j2}\} \leq t_{i1} + t_{j2} + \max \{t_{j1},\ t_{i2}\},$$

ou seja $W_{ji} \leq W_{ij}$. Portanto, é sempre possível processar J_j antes de J_i (onde t_{jk}, t_{ik}, k=1,2 satisfazem a condição do lema 2.2) visto que, o tempo de processamento W_{ji} dos jobs nesta ordem é menor ou igual

a W_{ij} (tempo de processamento dos jobs J_i, J_j onde J_i é predecessor de J_j).

TEOREMA 2.3 - A lista (sequência) L=L(1)...L(n) construída pelo algorítmo 2.1 é ótima.

Prova: Seja S uma sequência ótima e L o conjunto de todas as sequências que podem ser construídas a partir de S mediante a operação de trocar jobs consecutivos sem que o valor da função objetiva seja aumentado. Portanto, se $L \in \mathfrak{L}$ então L é uma sequência ótima.

Supor que $L \notin \mathfrak{L}$, então existe uma sequência $K \in \mathfrak{L}$ com L(i) =K(i) para i=1,...,s-1 e $L(s) \neq K(s)$ $(s < n)$, onde K escolhido desta maneira é a sequência, que possui o maior número de elementos comuns, isto é, s é máximo. Seja $K(s)=J_j$ e $L(s)=J_i$, então J_i é um sucessor de J_i em K. Pelo lema 2.1 cada job de J_k de L entre J_i e J_j inclusive $j_k=J_j$ é tal que

$$\min\{t_{k1}, t_{i2}\} \geq \min\{t_{k2}, t_{i1}\}$$ pois em caso contrário J_k seria um predecessor de J_i.

Pelo lema 2.2 pode se trocar em K o job J_i por todos os jobs J_k sem crescer o valor da função objetiva, gerando-se assim uma sequência $K' \in L$ com K'(i)= L(i) para i=1,...,s, chegando-se deste modo a contradição da maximalidade de s. Logo $L \in \mathfrak{L}$ portanto é uma sequência ótima.

Na secção seguinte serão tratados alguns problemas especiais de 3 máquinas, para os quais o algorítmo Jonhnson[2] pode ser aplicado.

---

2 - O algorítmo dado na unidade 2.2

2.3 - EXTENSÃO DO PROBLEMA $n/2/F/r_j=0/C_{max}$

Viu-se na unidade I deste capítulo que em um problema do tipo $n/3/F/r_j=0/C_{max}$, o ótimo ocorre sempre em uma "schedule" permutação. Johnson [14] mostrou que é possível utilizar o algorítmo 2.1 (mediante algumas modificações) em certos problemas bem particulares de 3 máquinas.

1º) O algorítmo 2.1 pode ser aplicado em um problema do tipo $n/3/F/r_j=0/C_{max}$ [14] sempre que os tempos de processamento dos jobs forem tais que $\min_K \{t_{k1}\} \geq \max_K \{t_{k2}\}$ ou

$\min_K \{t_{k3}\} \geq \max_K \{t_{k2}\}$ [14]. Neste caso, o algorítmo 2.3 a seguir difere do algorítmo 2.1 nos seguintes passos:

ALGORÍTMO 2.3

3. if $t_{j*1} + t_{j*2} = \min \{t_{j1} + t_{j2}, t_{12} + t_{13} / J_j \in X\}$ then

$S_1 \leftarrow S_1 \quad J_{j*}$

else

if $t_{j*2} + t_{j*3} = \min \{t_{j1} + t_{j2}, t_{j2} + t_{j3} / J_j \in X\}$ then

$S_2 \leftarrow J_{j*} \quad S_2$;

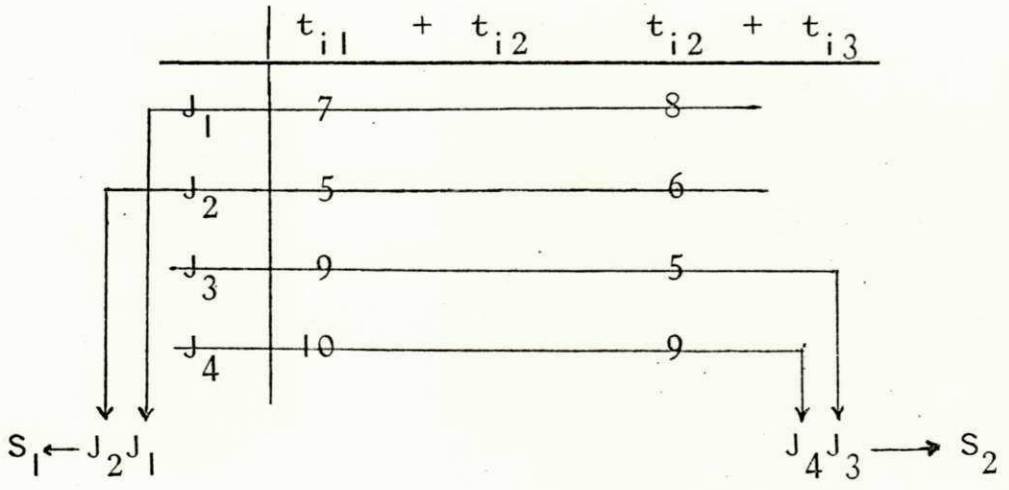
Os outros passos e as restrições para este algorítmo são iguais ao algorítmo 2.1, assim como, sua ordem da complexidade.

Exemplo 2.5 - Utilizando o algorítmo 2.3, encontrar uma seqüência ótima para os jobs de um problema de 3 máquinas, 4 jobs, cujos dados estão na tabela 2.4 a seguir:

Tabela 2.4

Job / maq.	J_1	J_2	J_3	J_4
M_1	5	4	6	8
M_2	2	1	3	2
M_3	6	5	2	7

Obs: $\max \{ t_{k2} \} \leq \min \{ t_{k1} \}$



A sequência ótima encontrada foi $L=J_2J_1J_4J_3$ cuja representação gráfica será:

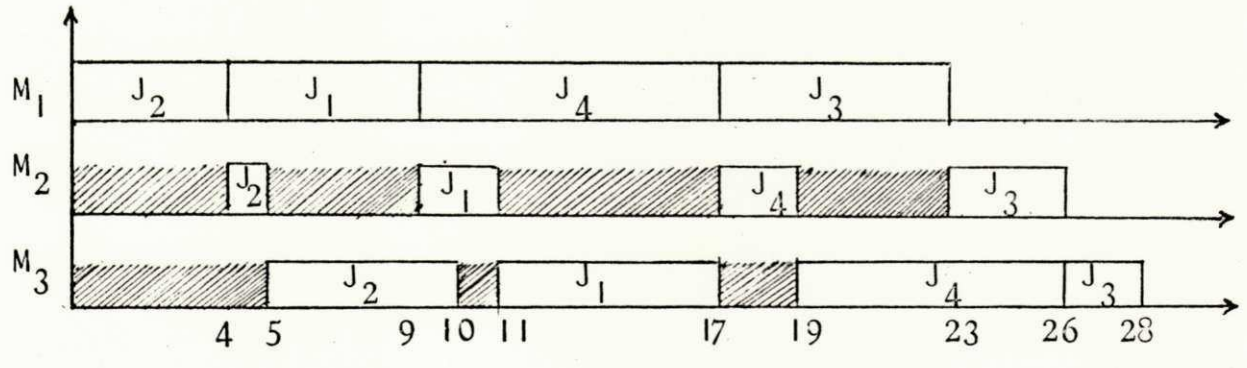


FIGURA 2.9 - "Schedule" ótima para o exemplo 2.5

Gupta expressou a Extensão do problema de Johnson através do algorítmo 2.2 mediante a modificação nos seguintes passos.

ALGORÍTMO 2.4

3. if $t_{j*i} < t_{j*3}$ then

4. $S(J_{j*}) \leftarrow -1/(t_{j*1} + t_{j*2})$

else

5. $S(J_{j*}) \leftarrow -1/(t_{j*2} + t_{j*3})$.

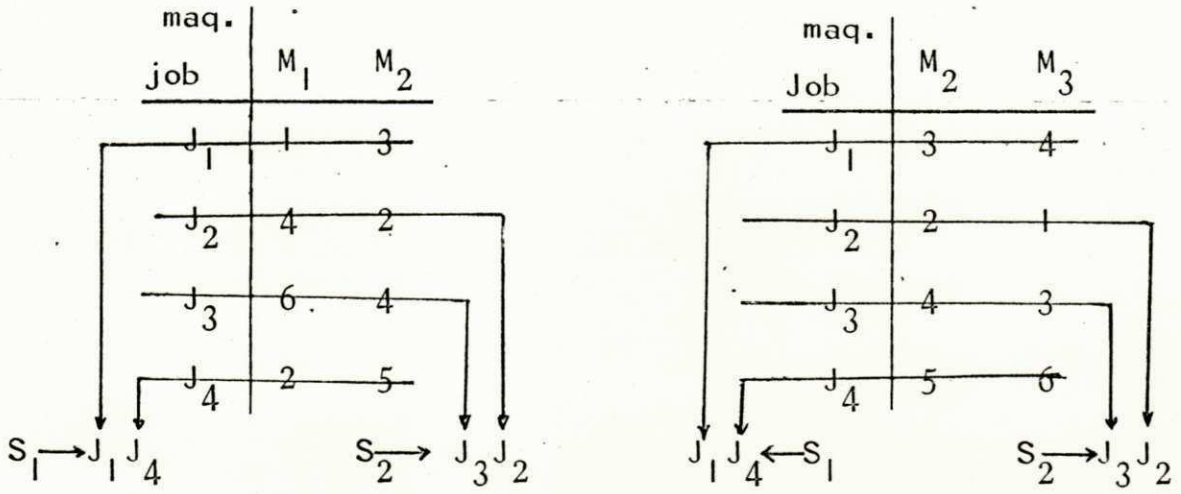
2º) Dado um problema do tipo $n/3/F/r_j=0/C_{max}$, se após aplicar-se o algorítmo 2.1 separadamente para as máquinas M_1, M_2 e M_2, M_3, constatar-se que ocorreu a mesma sequência ótima para os dois sub-problemas, então esta sequência será ótima para o problema global.

Exemplo 2.5 - Utilizando-se o algorítmo 2.1, encontrar uma sequência ótima para o problema de 3 máquinas, 4 jobs, cujos dados estão na tabela 2.5

Tabela 2.5

Job / maq.	J_1	J_2	J_3	J_4
M_1	1	4	6	2
M_2	3	2	4	5
M_3	4	1	3	6

Aplicando-se o algorítmo 2.1 para as máquinas M_1 e M_2 e depois para M_2 e M_3, tem-se:



Viu-se que ocorreu a mesma sequência ótima para os jobs nos dois sub_problemas, daí uma sequência ótima para o problema global. A representação gráfica será:

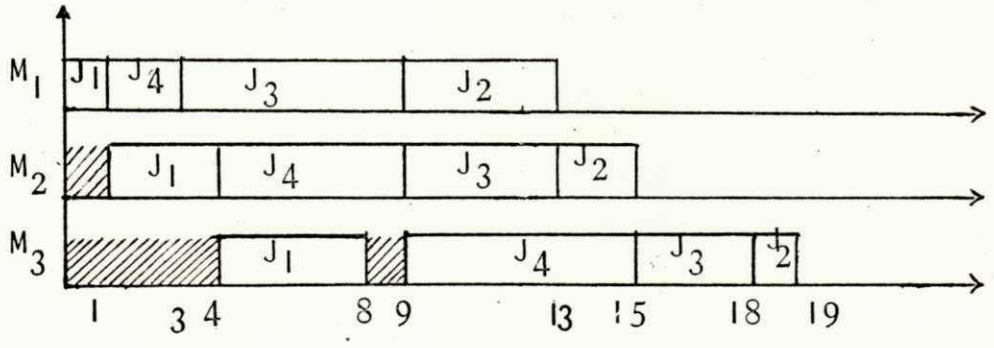


FIGURA 2.10 - "Schedule" ótima para o exemplo 2.5

C A P Í T U L O I I I

ALGORÍTMOS POLINOMIAIS PARA PROBLEMAS "J O B - S H O P"

O problema geral "Job-Shop" tem sido um constante desafio para os cientistas. Apesar da fácil exposição e visualização do que se pretende dele, uma solução ótima é muito difícil de ser alcançada. Até agora, pouquíssimos algorítmos polinomiais para a determinação de uma solução ótima foram desenvolvidos e além do mais, para problemas muito particulares. Dois deles serão vistos a seguir.

3.1 - O PROBLEMA $n/2/J/n_j \leq 2/C_{max}$

Jackson [13] mostrou que o problema $n/2/J/n_j \leq 2/C_{max}$ pode ser solucionado usando o algorítmo desenvolvido por ele, mediante algumas alterações:

O procedimento inicia particionando-se os n jobs em 4 diferentes conjuntos do seguinte modo:

I_1 - conjunto dos jobs com uma única operação e que deve ser executada pela máquina M_1.

I_2 - conjunto dos jobs com uma única operação e que deve ser executada pela máquina M_2.

I_1I_2 - conjunto dos jobs com duas operações de modo que a primeira seja executada pela máquina M_1 e a segunda pela máquina M_2.

I_2I_1 - conjunto dos jobs com duas operações de modo que a primeira seja executada pela máquina M_2 e a segunda pela máquina M_1.

Agora, ordenar cada conjunto separadamente. Para ordenar os conjuntos I_1I_2 e I_2I_1, usa-se o algorítmo 2.1 (Algorítmo de Johnson).

A disposição para estes conjuntos é que é fundamental na determinação da "Schedule" ótima e deve ser feita do seguinte modo:

Para a máquina M_1: as operações de I_1I_2, seguidas das de I_1, seguidas das de I_2I_1.

Para a máquina M_2: as operações de I_2I_1, seguidas das de I_2, seguidas das de I_1I_2.

TEOREMA 3.1 - O algorítmo descrito acima, é ótimo para todo problema do tipo $n/2/J/n_j \leqslant 2/C_{max}$.

PROVA:

1º caso: Quando não existir tempo ocioso nas máquinas. Neste caso a "schedule" é ótima.

2º caso: É claro que não é possível ociosidade nas duas máquinas ao mesmo tempo, visto que, podem ocorrer somente uma das si

tuações abaixo com relação aos jobs I_1I_2 e I_2I_1.

i) O tempo de processamento dos jobs de I_1I_2 pela máquina M_1 é maior (ou igual) que o tempo de processamento dos jobs de I_2I_1 pela máquina M_2 (ver figura 3.1 abaixo).

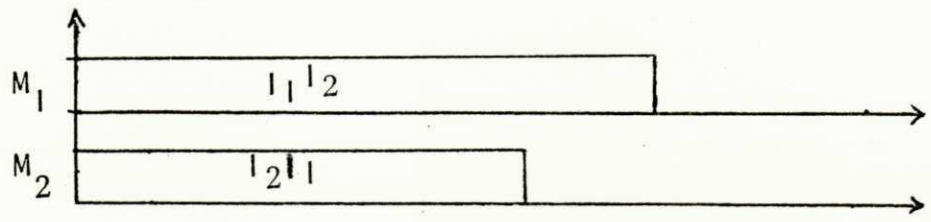


FIGURA 3.1

Neste caso a máquina M_1 não terá ociosidade.

ii) Em caso contrário (ver figura 3.2 abaixo), não haverá ociosidade na máquina M_2.

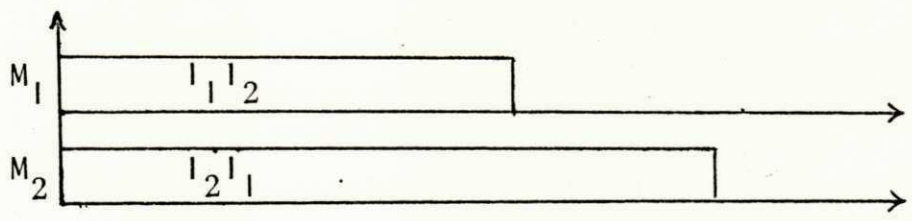


FIGURA 3.2

Neste caso, o valor C_{max} foi alcançado unicamente pelos jobs de $I_1 I_2$, cuja sequência foi obtida pelo algorítmo de Johnson. É óbvio, que não seria possível processar mais cedo os jobs de $I_1 I_2$ na máquina M_2 (por exemplo no lugar dos jobs de I_2), visto que já existe ociosidade na máquina M_2, após os jobs de I_2 (entre J_i e J_j). Porisso T é o menor valor possível para o tempo de conclusão dos jobs do problema geral, logo esta "schedule" também é ótima.

Este problema será ilustrado a seguir:

Exemplo 3.1 - Representar, através do diagrama de Gantt, uma "schedule" ótima para um problema "Job-Shop" de 2 máquinas, 10 jobs, cada job com no máximo 2 operações e com os dados das tabelas abaixo:

TABELA 3.1

Job / oper.	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
1	1	1	2	2	2	1	2	1	1	1
2	-	2	1	-	-	2	1	-	2	2

(a) máquinas onde as operações serão executadas.

Job / oper.	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
1	2	8	4	2	1	8	3	2	3	4
2	-	2	3	-	-	1	2	-	4	1

(b) tempos de processamento requeridos pelas operações.

Particionando-se os jobs em seus respectivos conjuntos,

tem-se:

$I_1 = \{1,8\}$

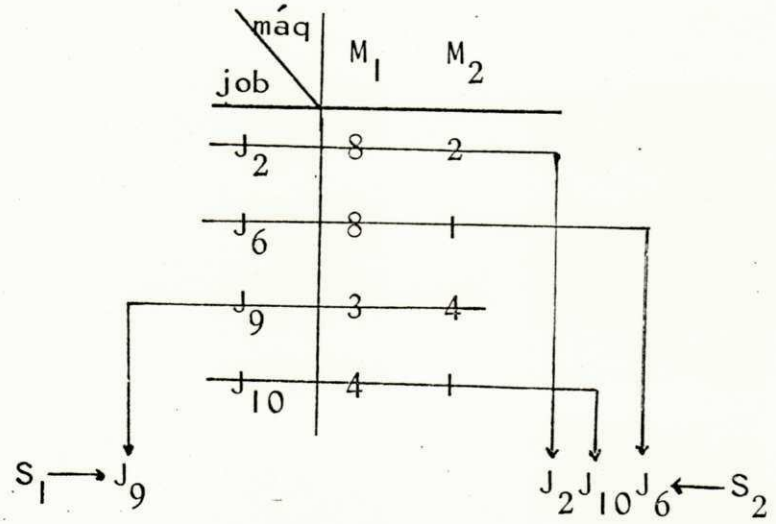
$I_2 = \{4,5\}$

$I_1 I_2 = \{2,6,9,10\}$

$I_2 I_1 = \{3,7\}$

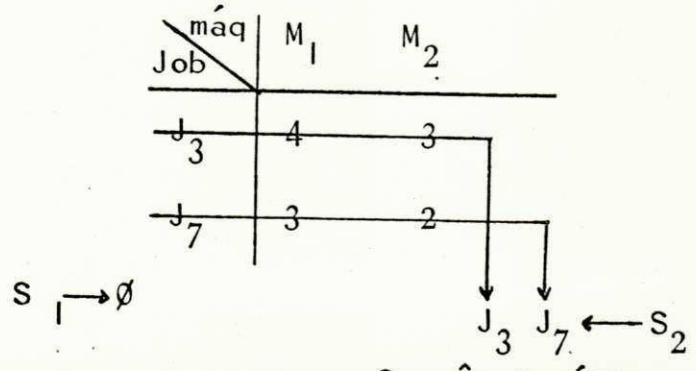
Aplicando-se o algorítmo 2.1 para a ordenação dos jobs de $I_1 I_2$ e $I_2 I_1$, tem-se:

i) Para o conjunto $I_1 I_2$:



Sequência ótima $J_9 J_2 J_{10} J_6$

ii) Para o comjunto $I_2 I_1$:



Sequência ótima $J_3 J_7$

Sem perda de generalidade, vai-se considerar o caso em que ocorre ociosidade na máquina M_2. Neste caso podem ocorrer as seguintes situações:

i)

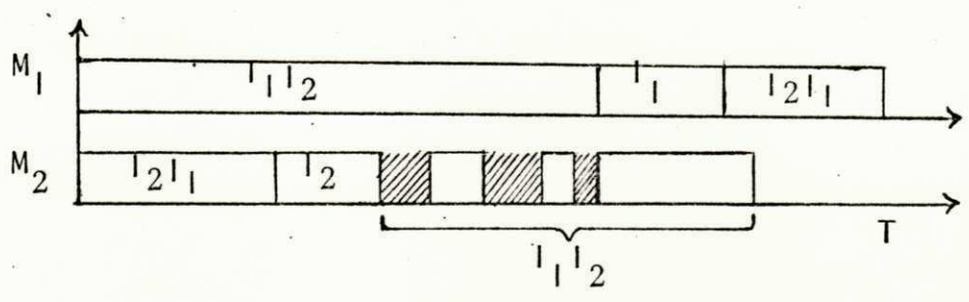


FIGURA 3.3 (a)

Pela figura 3.3 (a), é claro que esta "schedule" é ótima, visto que, $T=C_{max}$ foi dado pela máquina M_1, na qual não houve ociosidade.

ii)

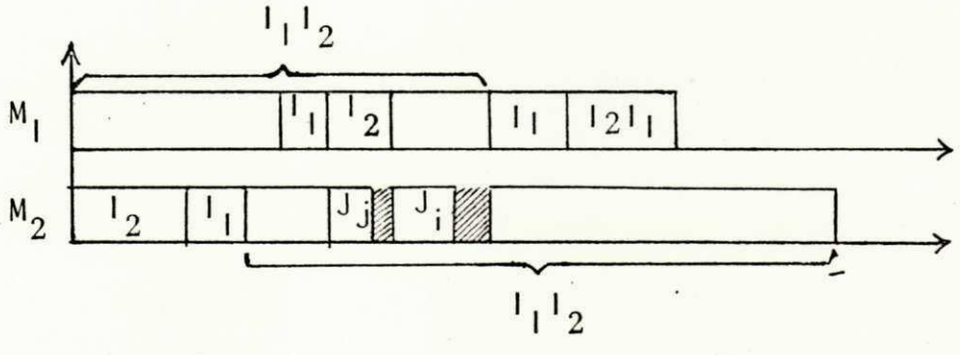


FIGURA 3.3 (b)

iii) Para I_1 e I_2, serão tomadas as seguintes sequências arbitrárias:

$I_1 \longrightarrow J_1 J_8$

$I_2 \longrightarrow J_4 J_5$.

Representando-se estes conjuntos graficamente, conforme a disposição porposta por Jackson, tem-se a seguinte "schedule" ótima:

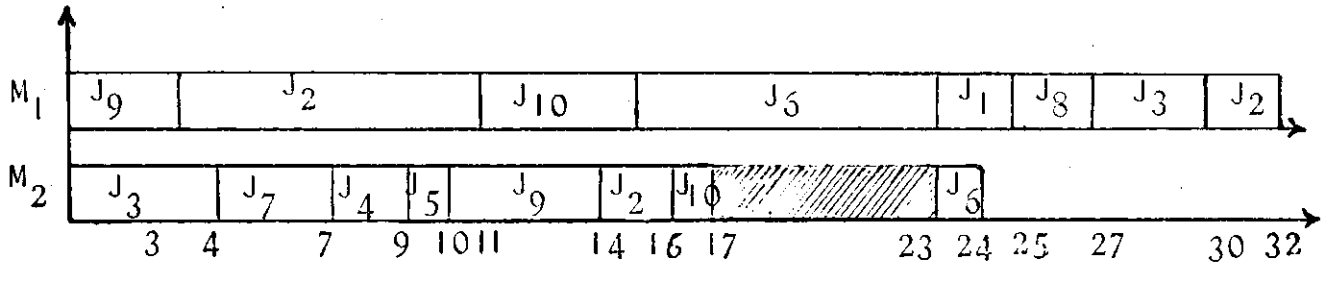


FIGURA 3.5 - "Schedule" ótima para o problema $10/2/J/n_j \leq 2/C_{max}$.

3.2 - O PROBLEMA $n/2/J/t_{jki}=1/C_{max}$

Viu-se na unidade 3.1, o algorítmo sugerido por Jackson para o problema $n/2/J/n_j \leq 2/C_{max}$ e agora será discutido um algorítmo para o problema $n/2/J/t_{jki}=1/C_{max}$ que embora sendo também polinomial é bem mais difícil que o anterior.

O problema $n/2/J/t_{jki}=1/C_{max}$ é formado de n jobs J_1, ..., J_n e 2 máquinas M_1, M_2. Cada job J_j consiste de n_j operações, as quais deverão ser processadas alternadamente pelas 2 máquinas.

Como neste problema, se for conhecida a máquina que processa a primeira operação de cada job, as demais também o serão, usa-se, por simplicidade, a notação O_{jk} em vez de O_{jki}:

Uma operação O_{jk} precede uma operação O_{jl} se $k < l$. Uma operação está disponível para ser processada quando todas as operações precedentes a ela já o foram.

As operações dos jobs deste problema serão processadas de acordo com o algorítmo a seguir:

ALGORÍTMO 3.2

1 - Rotular cada operação O_{jk} segundo a fórmula

$\alpha(O_{jk})=n_j - k+1$ ($\alpha(O_{jk})$ é o rótulo da k-ésima operação do job J_j).

2 - Processar primeiro a operação de mais alto rótulo na máquina requerida pela mesma e no tempo mais cedo possível. (No caso de operações com o mesmo rótulo, escolher arbitrariamente a operação que será processada primeiro).

3 - Remover do problema, a operação processada, Retornar ao passo 2 até que todas as operações sejam processadas.

Pare.

Este algorítmo pode ser implementado em O(N) passos através de uma estrutura de dados apropriada. Aqui N é o número de operações de todos os jobs J_j $j=1,\ldots,n$ ($N=\sum_{j=1}^{n} n_j$) [4,11].

Sejam:

$t_{max}+1$ o tempo de processamento da "schedule"' do algorítmo 3.2,

$A(0),\ldots,A(t_{max})$ as operações dos jobs J_j, $j=1,\ldots,n$ processadas pela máquina M_1.

$B(0),\ldots,B(t_{max})$ as operações dos jobs j_j, $j=1,\ldots,n$ processadas pela máquina M_2.

(A,B) a "schedule" construída pelo algorítmo 3.2.

Sejam: $A(t)=\emptyset$ respectivamente $B(t)=\emptyset$ se nenhum job é processado no tempo t pelas máquinas M_1, respectivamente M_2 e T_1 respectivamente T_2 o número de operações dos jobs para serem processadas pela máquinas M_1 respectivamente M_2.

TEOREMA 3.2 - A "Schedule" (A,B) é ótima.

PROVA:

1 - Se $A(t)\neq\emptyset$ e $B(t)\neq\emptyset$ para todo $t=0,\ldots,t_{max}$, en

tão $t_1=t_2=t_{max}+1$ logo (A,B) é ótima.

2 - Se $A(0)=\emptyset$ e $A(t)\neq\emptyset$, $B(t)\neq\emptyset$ para $t=1,\ldots,t_{max}$, tem-se $B(0)\neq\emptyset$ e portanto $T_2=t_{max}+1$, logo (A,B) é ótima. Similarmente, (A,B) é ótima para o caso em que $B(0)=\emptyset$ e $A(t)\neq\emptyset$, $B(t)\neq\emptyset$ para $t=1,\ldots,t_{max}$.

Considerar agora o caso onde:

3 - $A(t)=\emptyset$ ou $B(t)=\emptyset$ para algum $1 \leq t \leq t_{max}$. Escolher o mínimo $t \geq 1$ que satisfaz 3. Sem perda de generalidade supor que $A(t)=\emptyset$. Deste modo, 2 casos devem ser considerados:

caso 1: $A(t+1)=\emptyset$.

Mostra-se neste caso que $A(\nu)=\emptyset$ para todo $\nu \geq t$. Seja $B(t)=0_{jk}$, então $\alpha(0_{jk})=1$, pois caso contrário $0_{j,k+1}$ seria processada pela máquina M_1, no tempo t+1. Seja agora $A(t+2)\neq\emptyset$ então B(t+1) e A(t+2) devem pertencer ao mesmo job pois em caso contrário, A(t+2) deveria ser processada mais cedo. Assim $\alpha(B(t+1))\geq 2$ que é uma contradição com $\alpha(0_{jk})=1$. Logo tem-se $A(t+2)=\emptyset$. Da mesma maneira obtem-se $A(\nu)=\emptyset$ para todo $\nu > t+2$ e portanto $B(\nu)\neq\emptyset$ para todo $\nu=t,\ldots,t_{max}$.

Qualquer schedule neste caso toma uma das duas formas mostradas na figura 3.4 a seguir:

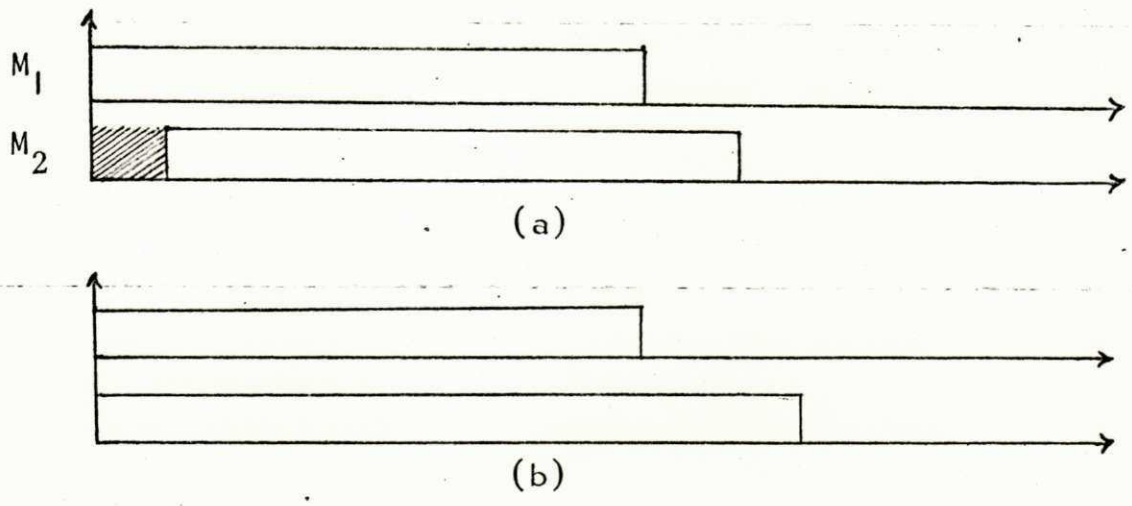


FIGURA 3.4

No caso (a), cada job requer para execução de sua primeira operação a máquina M_1. Assim o tempo de processamento T_2+1 é ótima . No caso (b) tem-se como tempo de processamento T_2 que também é ótima.

CASO 2: $A(t+1)\neq\emptyset$

Neste caso Job J_j é processado ininterruptamente do tempo 0 (zero) até t_{max} e assim a schedule (A,B) é ótima (ver prova' a seguir:

$A(t+1)$ e $B(t)$ devem pertencer ao mesmo Job J_j, pois em caso contrário, $A(t+1)$ seria processada mais cedo. É claro que $B(t-1)$ não terá um sucessor, porque em caso afirmativo esta seria processada pela máquina M_1, no tempo t. Assim $\alpha(B(t-1))=1$. Também $\alpha(B(t))\leq 2$. $A(t-1)$ deve pertencer também ao Job J_j, pois em caso contrário $B(t)'$ deveria ser processada no lugar de $B(t-1)$.

Tem-se também $\alpha(B(t+1)) \leq 1$, porque em caso contrário $B(t+1)$ seria processada no tempo t-1. Finalmente $A(t+2)=\emptyset$, pois um

job para ser processado pela máquina M_1 no tempo t+2, nem pode pertencer ao job J_j, nem pode ser um sucessor de B(t+1) e consequentemente seria processada pela máquina M_1, no tempo t. E agora surge uma situação idêntica à situação do tempo t em M_1, ou seja B(t+2) e A(t+1), devem pertencer ao mesmo job, pois em caso contrário B(t+2), seria processado antes. Assim o raciocínio empregado em t e t+2 deverá ser repetido até t_{max} e o job J_j será processado ininterruptamente até $t_{max}+1$.

Por outro lado, tem-se $\alpha(A(t-2)) \leq 2$, pois em caso contrário, existiria uma operação deste job para ser processada pela máquina M_1 no tempo t e $\alpha(A(t-1)) \geq 3$, porque viu-se que este job será processado nos tempos t,t+1 e t+2 respectivamente pelas máquinas' M_2, M_1, M_2. Da desigualdade $\alpha(A(t-1)) \geq 3$ segue-se que B(t-2) deve pertencer a J_j e portanto $\alpha(B(t-2)) \geq 4$. Continuando-se com este raciocínio, conclui-se que o job J_j é processado ininterruptamente do tempo 0 (zero) até o tempo $t_{max}+1$, daí, $t_{max} = n_j$.

O algorítmo 3.2 será ilustrado com o exemplo a seguir:

Exemplo 3.2 - Utilizar o algorítmo 3.2 na determinação de ' uma "schedule" ótima para o problema do tipo $3/2/J/t_{jki}=1/C_{max}$ com $n_1=6$, $n_2=1$, $n_3=1$ e com as operações O_{11} e O_{21} executada na máquina ' M_1 e a operação O_{31} na máquina M_2.

Rotulando-se cada operação, tem-se :

$\alpha(O_{11})=n_1 - 1+1 = n_1=6$. Analogamente,

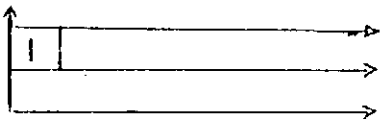
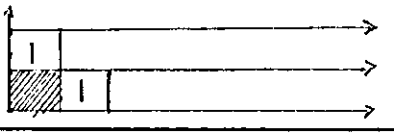
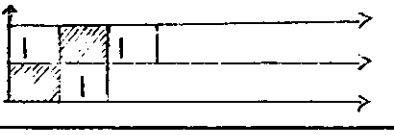
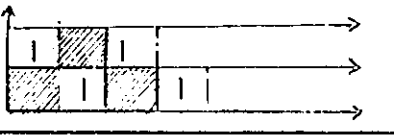
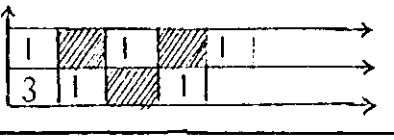
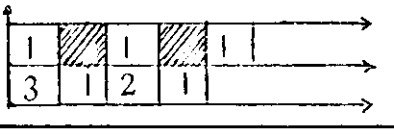
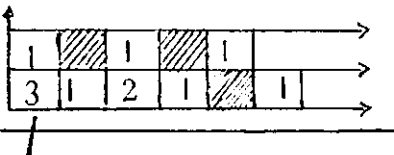
$\alpha(O_{12})=5$, $\alpha(O_{13})=4$, $\alpha(O_{14})=3$, $\alpha(O_{15})=2$, $\alpha(O_{16})=2$

$\alpha(O_{21})=1$

$\alpha(O_{31})=1$

Processando-se as operações na ordem não crescente de seus rótulos, encontra-se a "schedule" ótima mostrada na tabela a seguir:

Tabela 3.2 - Esquema utilizado na determinação da "schedule" ótima para o exemplo 3.2

assos	Rótulos (O_{jk}) ordenados de maneira não crescente	Operaçoes correspondentes ao rótulo (O_{jk})	Índice da máq. para estas operações.	Schedules parciais
1	6	O_{11}	1	
2	5	O_{12}	2	
3	4	O_{13}	1	
4	3	O_{14}	2	
5	2	O_{15}	1	
6	1	O_{31}	1	
7	1	O_{21}	1	
8	1	O_{16}	2	

"Schedule" ótima

C A P Í T U L O I V

ALGORÍTMOS ENUMERATIVOS PARA PROBLEMAS "FLOW-SHOP" E "JOB-SHOP"

Em capítulos anteriores foram desenvolvidos algorítmos polinomiais para problemas "Flow-Shop" e "job-Shop". Entretanto estes algorítmos só se verificam para problemas de 2 máquinas e para alguns problemas "Flow-Shop" muito particulares de 3 máquinas. Portanto, neste capítulo, serão desenvolvidos algorítmos para problemas mais gerais. Estes algorítmos são chamados algorítmos "Branch and Bound" e consistem de um processo enumerativo.

Como todo algorítmo enumerativo, estes também têm uma inevitável desvantagem: O tempo computacional e a memória requerida crescem exponencialmente com o número de jobs e máquinas.

No capítulo 2 viu-se que uma "schedule" ótima só ocorrerá em uma "schedule" permutação quando se tratar de um problema "Flow-Shop" com m 3. O procedimento "Branch and Bound" é aplicado em problemas mais gerais (m 3) e determina a sequência de jobs, para a qual, a "schedule" permutação embora não sendo a "schedule" ótima para o problema

está muito próxima desta. É importante salientar que na prática sempre estar-se interessados em"schedules"permutação.

4.1 - ALGORÍTMO "BRANCH AND BOUND" PARA PROBLEMAS "FLOW-SHOP"

Os primeiros cientistas a desenvolverem o procedimento "Branch and Bound" para problemas "Flow-Shop" foram Ignall e Schrage. Este procedimento consiste na determinação de uma árvore, onde cada nodo é representada por uma sequência parcial de jobs. Para cada nodo desta árvore é associado um limite inferior para o valor C_{max}. Este depende da sequência dos jobs já processados e dos tempos de processamentos dos demais jobs. O procedimento termina quando for encontrada uma sequência de todos os jobs, de modo que o limite inferior desta, seja menor ou igual aos demais limites das sequências parciais dos nodos finais.

Este algorítmo será desenvolvido para um problema "Flow-Shop" de m máquinas $M_1,\ldots,M_m$ e n jobs $J_1,\ldots,J_n$. O nodo inicial é representado pela letra λ e significa que a sequência de jobs é vazia. A partir desta, serão determinadas sequências mediante a operação $\sigma \leftarrow \sigma J_i$ onde J_i é um job do conjunto ainda não presente na sequência σ. O limite inferior para o valor C_{max} de cada sequência σ é denotado por B_σ onde $B_\sigma = \max_{i=1}^{m} (b_i(\sigma))$ e $b_i(\sigma)$ é calculado como segue:

$u(\sigma)$ = último elemento da sequência

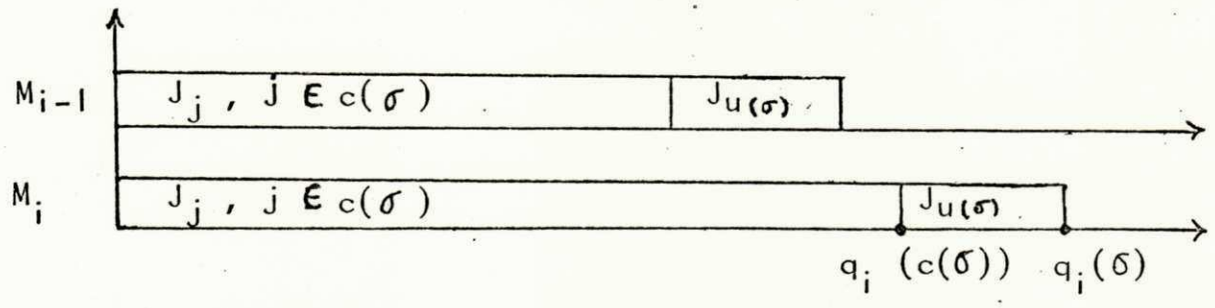
$c(\sigma)$ = sequência excluindo-se o último elemento ($c(\sigma)=\sigma \setminus u(\sigma)$)

σ' = conjunto de todos os jobs do problema e que não estão na sequência σ. ($\sigma' = \{J_1,\ldots,J_n\} \setminus \sigma$).

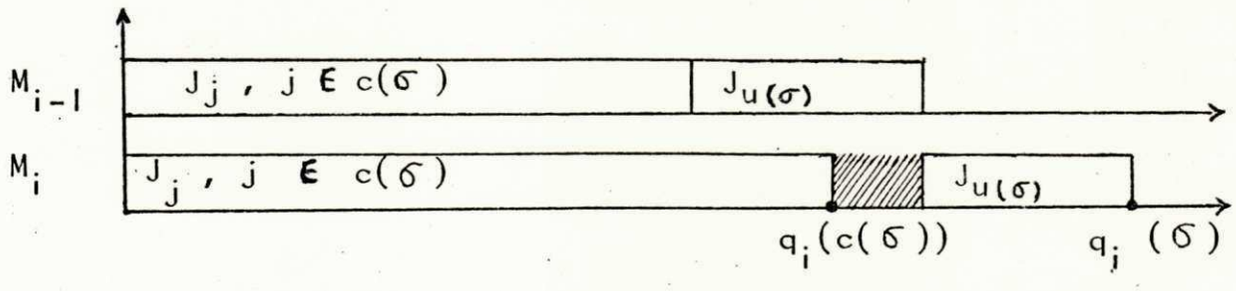
$q_i(\sigma) = \sum_{j\in\sigma} t_{ji}$ é o tempo de conclusão dos jobs de σ na maquina M_i.

$q_i(\sigma) = \max\ q_i(c(\sigma)),\ q_{i-1}(\sigma) + t_{u(\sigma)i}$ é o tempo de conclusão dos jobs de σ na máquina M_i.

No valor de $q_i(\sigma)$ é necessário verificar-se o máximo, porque duas possibilidades podem ocorrer com relação ao processamento dos jobs de σ (ver figura 4.1 (a) e (b)).



(a)



(b)

FIGURA 4.1

Portanto

$$b_i(\sigma) = q_i(\sigma) + \sum_{j \in \sigma'} t_{ji} + \min_{j \in \sigma'} \left\{ \sum_{k=i+1}^{m} t_{jk} \right\}$$

onde $b_i(\sigma)$ representa um limite inferior para C_{max} baseado nos jobs já processado na máquina M_i.

Este fato pode ser mostrado na figura 4.2 abaixo

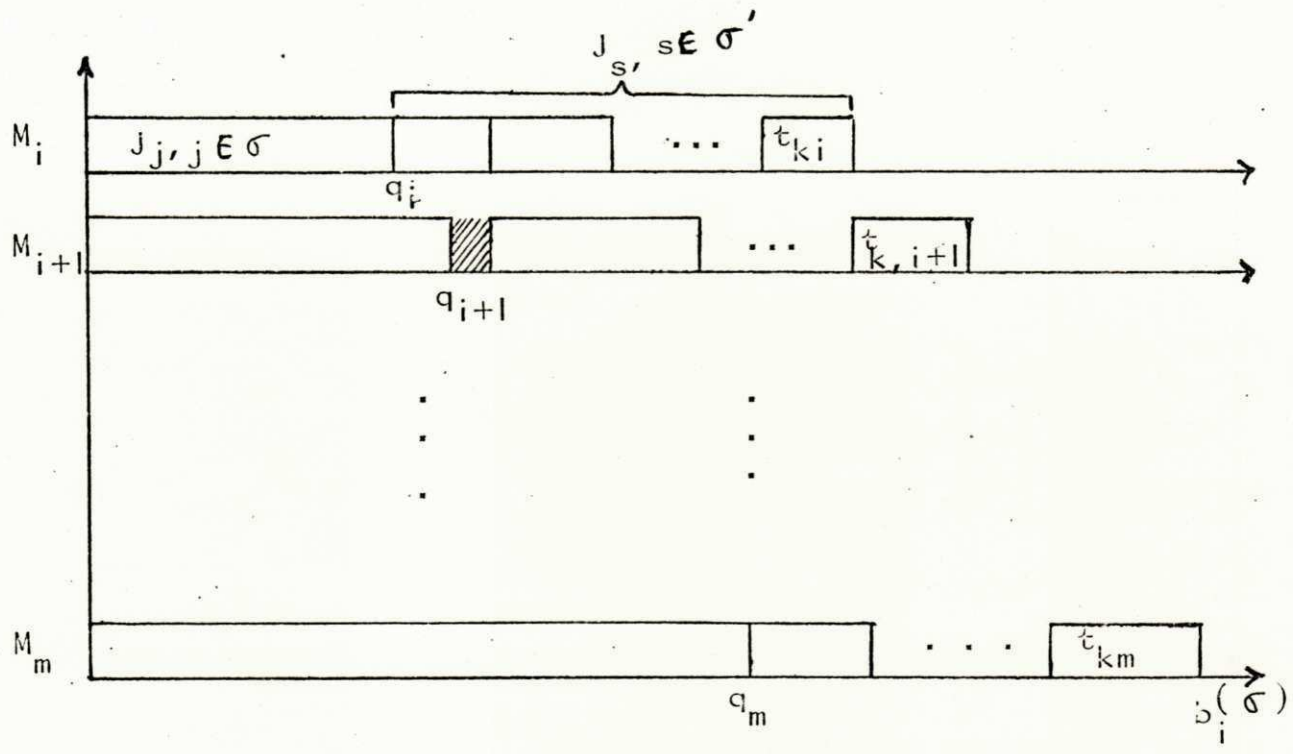


FIGURA 4.2

O algorítmo usa uma lista L formada pelas sequências parciais criadas durante o desenvolvimento do algorítmo.

Com base nestes resultados, pode-se agora, escrever o algorítmo de Ignall e Schrage.

ALGORÍTMO 4.1

1. $L \longleftarrow \lambda$
2. $B \longleftarrow \infty$
3. while $L \neq \emptyset$ do
   Begin
4. Remova de L a sequência σ com o menor valor B_σ ;
5. If σ tem n elementos
   then
6. Saia com a sequência ótima
   else
7. Adicione em L, para cada $J_i \in \sigma'$ a sequência σJ_i e calcule o correspondente valor $B_{\sigma J_i}$;
8. end.

O número de passos deste algorítmo cresce exponencialmente com o número de jobs.

Para ilustrar o algorítmo 4.1, será desenvolvido o exemplo seguinte:

Exemplo 4.1 - Utilizando-se o algorítmo 4.1, determinar a "schedule" permutação ótima para um problema "Flow-Shop" de 3 máquinas, 4 jobs com objetivo C_{max} e cujos tempos de processamento das operações são dados na tabela 4.1 - abaixo:

Tabela 4.1

	J_1	J_2	J_3	J_4
M_1	2	8	10	11
M_2	3	9	1	13
M_3	9	14	4	3

Aplicando-se o algorítmo 4.1 nestes dados, tem-se:

$L \longleftarrow \lambda$ $\quad B_\lambda = \infty \quad \sigma' = \{J_1, J_2, J_3, J_4\}$

$L \leftarrow J_1\ J_2\ J_3\ J_4$ onde para $\sigma = J_1$, tem-se $\sigma' = \{J_2, J_3, J_4\}$. O valor de B_{J_1} é calculado do seguinte modo:

$q_1(J_1) = t_{11} = 2$

$q_2(J_1) = \max\ \{0, 2\} + t_{12} = 2+3=5$

$q_3(J_1) = \max\ \{0, 5\} + t_{13} = 5+9=14$

$b_1(J_1) = q_1(J_1) + t_{21} + t_{31} + t_{41} + \min_{j \in \sigma'} \{t_{j2} + t_{j3}\} = 2+29+5=36$

Analogamente $b_2(J_1) = 31$ e $b_3(J_1) = 35$

Logo $B_{J_1} = \max\ \{36, 31, 35\} = 36$. Procedendo-se deste modo, tem-se:

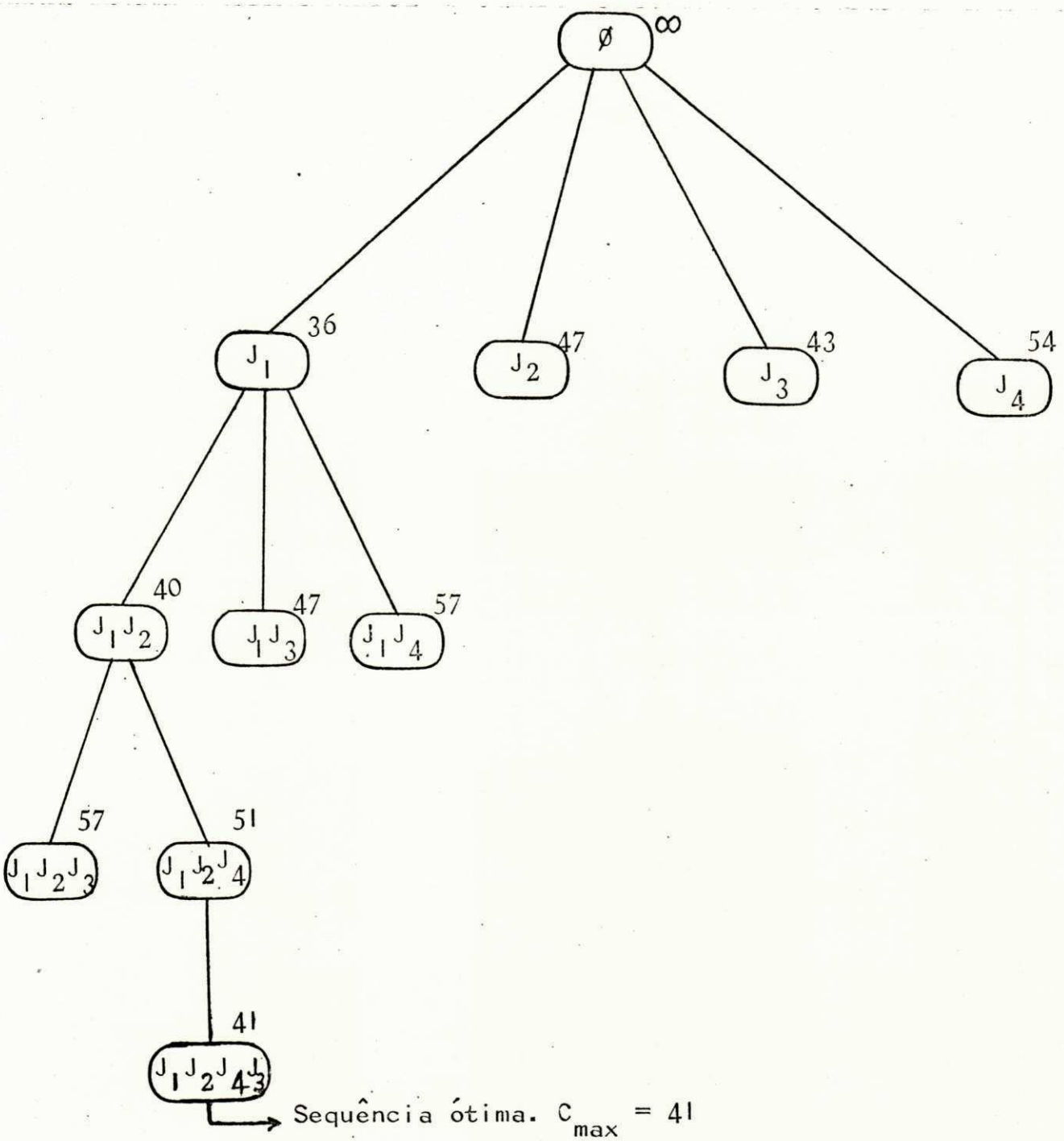
$B_{J_2} = 47 \quad B_{J_3} = 43 \quad B_{J_4} = 54$

A sequência que deverá ser removida de L é a sequência $\sigma = J_1$ pois B_{J_1} é o minimo, e portanto a lista L tem as seguintes sequências parciais.

$L \longleftarrow J_2\ J_3\ J_4 \quad J_1\ J_2 \quad J_1\ J_3 \quad J_1\ J_4$

Continuando-se com o processo, tem-se a árvore mostrada a seguir, na qual a sequência ótima será indicada. O número acima de cada nodo, representa o limite inferior B_σ , onde σ é a sche-

dule parcial do nodo em questão.



4.2 - ALGORÍTMO "BRANCH AND BOUND" PARA PROBLEMAS "JOB-SHOP"

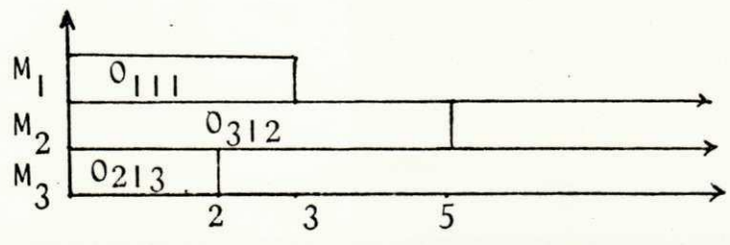
Os primeiros cientistas a darem resultados concretos na determinação de algorítmos enumerativos para problemas "Job-Shop", foram Giffler e Thompson que construíram um algorítmo de enumeração total [2] .

Neste trabalho, foram feitas modificações tornando este algorítmo da mesma linha do Algorítmo de Ignall e Schrage, ou seja, construção de uma árvore onde cada nodo representa uma "Schedule" parcial S e associado a cada um deles está um limite inferior para o tempo de processamento.

Este algorítmo é desenvolvido para um problema de m máquinas $M_1,\ldots,M_m$ e n jobs $J_1,\ldots,J_n$ onde cada job J_j consta de n_j operações O_{jki}. O número total de operações do problema é dado por N ($N= \sum_{j=1}^{n} n_j$).

Seja σ_{jki} o tempo mais cedo no qual a operação O_{jki} poderá ser processada. Este tempo é determinado pelo máximo entre o tempo de conclusão das operações $O_{j,k-1,l}$ onde O_{rsi} é a última operação processada pela máquina M_i. Nota-se por ϕ_{jki} o tempo mais cedo possível no qual a operação O_{jki} poderá ser concluída, isto é, $\phi_{jki} = \sigma_{jki} + t_{jki}$ onde t_{jki} é o tempo de processamento requerido pela operação O_{jki}. Para aclarar estes fatos, o seguinte exemplo será analisado:

Considerar a "schedule" parcial abaixo



Seja $O_{jki}=O_{221}$ com $t_{221}=2$ a operação para ser processada nesta "schedule", então $\sigma_{221}=\max\{2,3\}=3$, ou seja, o processamento da operação O_{221} deverá ser iniciado no tempo 3 e concluído no tempo $\phi_{221}=3+2=5$.

O algorítmo usa uma lista L formada por uma sequência de "schedules" na ordem não decrescente de seus limites inferiores B_S para tempo de processamento e O(S) o conjunto das operações O_{jki} processáveis em cada uma destas "schedules" S.

A inicialização deste algorítmo se dá do seguinte modo :

Coloca-se na lista L a "schedule" vazia e associado a λ o valor $B^1_\lambda = \max\{\sum_{\gamma=1}^{n_j} t_{j\gamma}\}$.

Em um passo geral, este algorítmo se comporta do seguinte modo:

Remove-se de L a primeira "schedule" S. Se S tiver N operações processadas, então esta é a "schedule" ótima; se não, para cada operação $O_{jki'} \in O(S)$ com $\sigma_{jki'} < \phi^*_{j'k'i'}$, onde $\phi^*_{j'k'i'} = \min_{O_{jki} \in S} \{\phi_{jki}\}$, será criada uma nova "schedule" parcial S^* mediante o processamento desta operação no tempo $\sigma_{jki'}$. Determinar seu correspondente valor $B_S{}^*$. Inserir estas "schedules" em L atentando para os valores $B_S{}^*$, após ter verificado a não existência de "Schedules" parciais iguais em L.

---

1 - B representa o limite inferior para o tempo de conclusão da "schedule" λ.

Nota: Viu-se que a condição para se criar uma nova "schedule" parcial é que $\sigma_{jki'} < \phi_{j'k'i'}$. Isto se explica pelo fato de que se $\sigma_{jki'} \geq \phi^*_{j'k'i'}$, então existe uma outra operação ' que deverá ser processada neste espaço, pois em caso contrário, a máquina M_i, ficaria ociosa durante este tempo. Para melhor esclarecimento, ver o exemplo abaixo:

Considerar a "schedule" parcial S abaixo e seja O(S) = $= \{O_{312}, O_{123}, O_{222}\}$ com $t_{312}=2$, $t_{123}=3$ e $t_{222}=2$

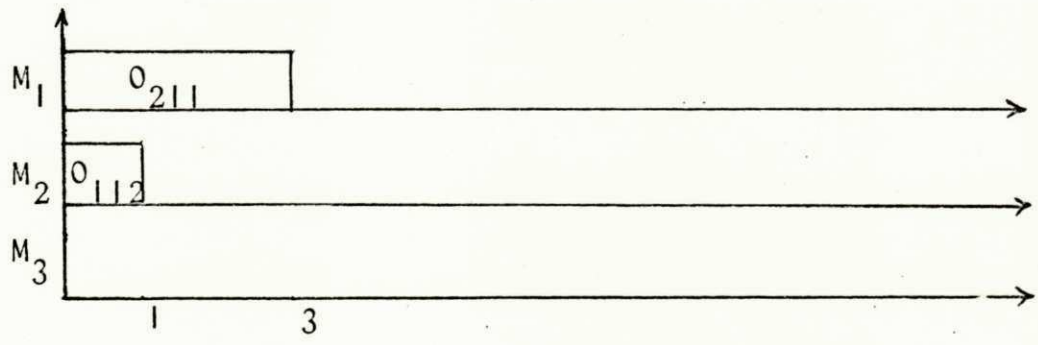


FIGURA 4.3 - "SCHEDULE" S

Obviamente, $\sigma_{312}=1$, $\sigma_{123}=1$, $\sigma_{222}=3$ e

$\phi_{312}=3$, $\phi_{123}=4$, $\phi_{222}=5$

Portanto: $\phi^*_{j'k'i'} = \min \{3,4,5\} = 3$ logo $i'=2$

como $\sigma_{312}=1 < 3$ então criar uma nova "schedule" S*, processando na "schedule" da figura 4.3, a operação O_{312} no tempo 1.

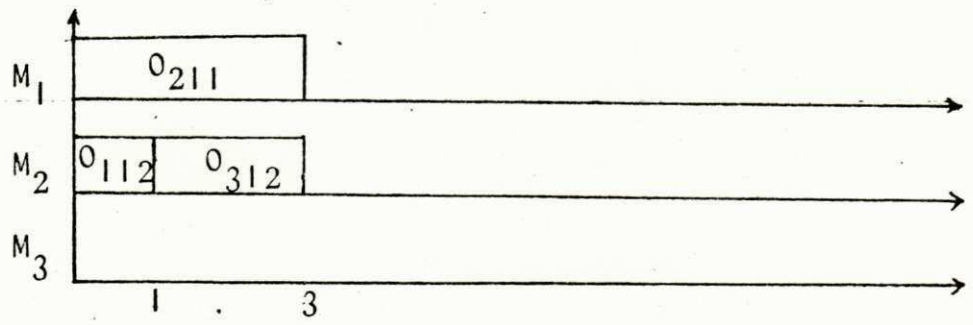


FIGURA 4.4 -"SCHEDULE"S*.

Como $\sigma_{222}=3 = \emptyset^*_{j'k'i'}$, não se deve processar esta operação, pois em caso contrário, teria-se a"schedule"(ver fig. 4.5)com um espaço vazio no qual seria possível processar a operação O_{312}

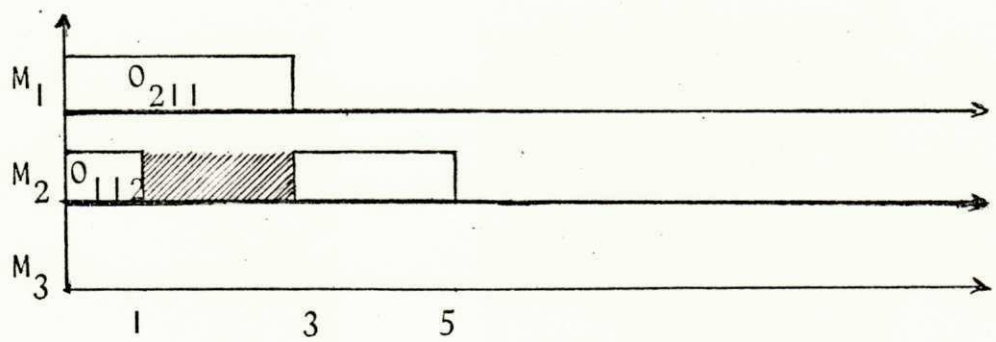


FIGURA 4.5 - "Schedule" impossível de ser criada pelo alg 4.2

O algorítmo em questão pode ser formalizado do seguinte modo:

ALGORÍTMO 4.2

1. $L \longleftarrow \lambda$
2. $B_{\lambda} \longleftarrow \max \{ \sum_{\nu=1}^{n_j} t_{j\nu} \ / j=1,\ldots,n \}$
3. While $L \neq \emptyset$ do

   begin

4. Remova de L a primeira "schedule" S. (obviamente B_S é o menor valor das "schedules" S de L)
5. if S tem N elementos

   then

6. Saia com a "schedule" ótima S.

   else

   begin

7. Forme o conjunto O (S) com todas as operações O_{jki} processáveis em S;
8. $\emptyset^*_{j'k'i'} \quad \min_{O_{jki} \in O(S)} \{ \emptyset_{jki} \}$;
9. While O (S) $\neq \emptyset$ do

   begin

10. escolher $O_{jki'} \in O(S)$;
11. if $\sigma_{jki'} < \emptyset^*_{j'k'i'}$ then

   begin

12. criar uma nova "schedule" parcial S^* onde a operação $O_{jki'}$ é processada no tempo $\sigma_{jki'}$;
13. $B_S \longleftarrow \max \{ B_S , \sigma_{jki'} + \sum_{\nu=k}^{n_j} t_{j\nu i(\nu)} \}$;
14. Inserir as schedules S^* na lista ordenada L;

```
15                              end;
16.                         end;
17.                     end;
18.                 end.
```

Para efeito de programação, é fácil notar que este algorítmo não está definido de modo único. No passo 8 pode ocorrer de $\min\{\emptyset_{jki}\}$ não ser único. Neste caso, a seguinte regra de prioridade poderá ser implantada:

Se $\emptyset_{pqr} = \emptyset_{suv} = \min\{\emptyset_{jki}\}$ onde O_{pqr} e $O_{suv} \in O(S)$ e $r < v$ então escolher primeiro O_{pqr}.

Para ilustrar este algorítmo, o seguinte exemplo será desenvolvido.

Exemplo 4.2 - Utilizando o algorítmo 4.2, determinar a "schedule" ótima para o problema "Job-Shop" de 3 máquinas, 3 jobs, $n_1=3$ $n_2=1$, $n_3=2$ e com os dados das tabelas 4.2 (a) e 4.2 (b) abaixo:

Tabela 4.2

job \ op	1	2	3
J_1	2	3	1
J_2	1	-	-
J_3	2	1	-

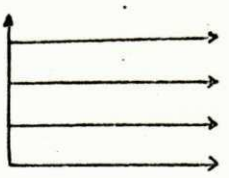
a) Máquinas onde as operações serão processadas.

Job \ op	1	2	3
J_1	1	4	2
J_2	3	-	-
J_3	2	4	-

b) Tempos de processamento das operações.

1. $L \leftarrow \lambda$

$B_\lambda = 7$



← "Schedule" vazia S

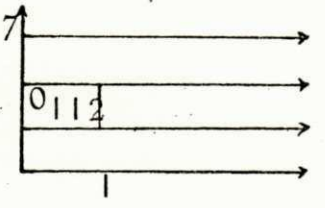
$O(\lambda) = O(S) = \{ O_{112}, O_{211}, O_{312} \}$

$\phi^*_{j'k'i'} = \min \{ \phi_{112}, \phi_{211}, \phi_{312} \} = \min \{1, 3, 2\} = 1$

$i' = 2$

para O_{112}, tem-se

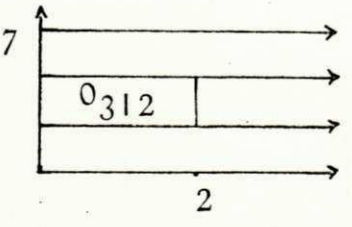
$\sigma_{112} = 0 < \phi^*_{j'k'i'}$ logo $S^* =$



$B_{S^*} = 7$

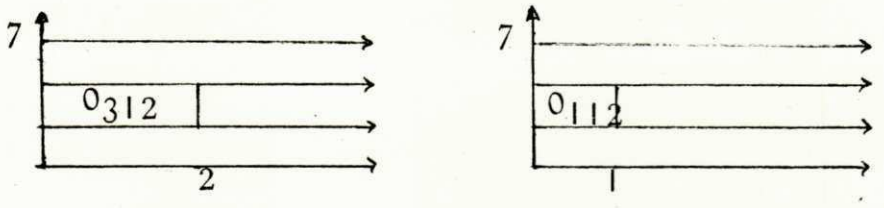
para O_{312}, tem-se

$\sigma_{312} = 0 \quad \phi^*_{112}$ logo $S^* =$

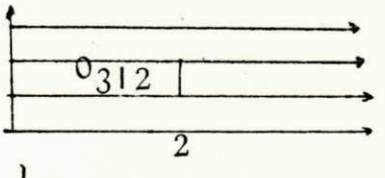


$B_{S^*} = 7$

L :



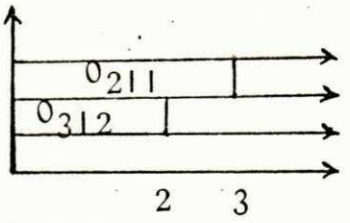
Remove-se a primeira "schedule" S de L



$O(S) = \{ O_{112}, O_{211}, O_{321} \}$

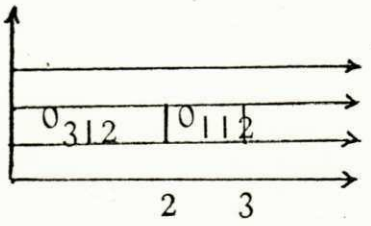
$\phi^*_{j'k'i'} = \min \{ \phi_{112}, \phi_{211}, \phi_{321} \} = \min \{3, 3, 6\} = 3$

para i' = 1, tem-se a operação O_{211} e

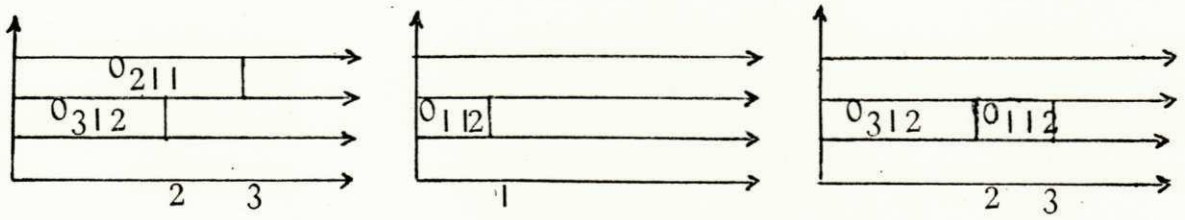
$\sigma_{211} = 0 < \phi^*_{j'k'i'}$ logo S^* = 

$B_S^* = 7$

para i' = 2, tem-se a operação O_{112} e

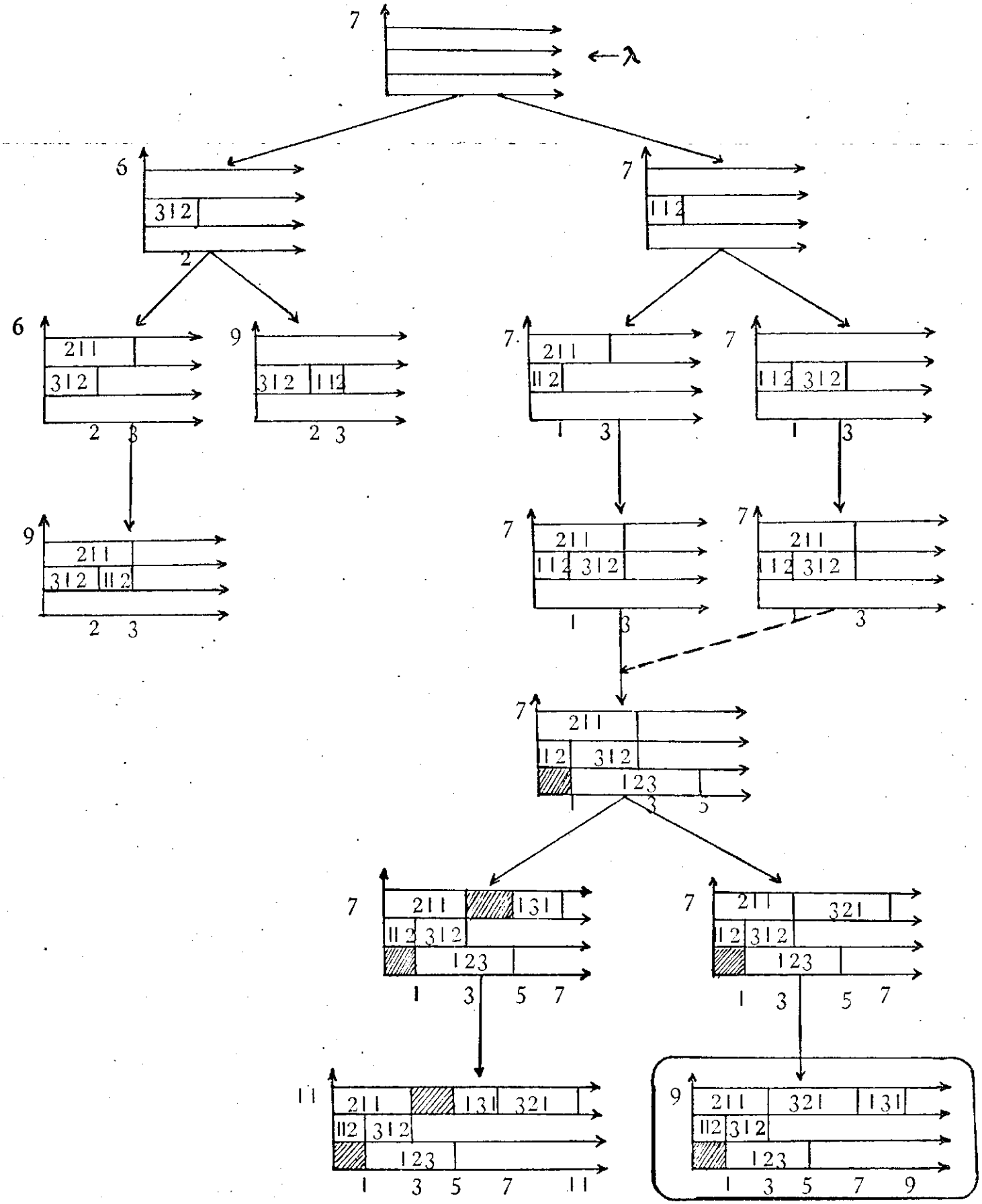
$\sigma_{112} = 2 < \phi_{j'k'i'}$ logo S^* = 

$B_S^* = 9$

L : 

Continua-se com o processo até ser removida uma schedule com N operações processadas.

A árvore resultante é a seguinte:



C A P Í T U L O V

COMPLEXIDADE DOS PROBLEMAS "F L O W - S H O P" E "J O B - S H O P"

5.1 - TEORIA DOS PROBLEMAS NP-COMPLETOS.

Nos capítulos 2 e 3 foram desenvolvidos alguns algorítmos polinomiais para problemas "Flow-Shop" e "Job-Shop" bem particulares e no capítulo 4, analisou-se alguns algorítmos enumerativos para problemas mais gerais. Estes algorítmos são também chamados ALGORÍTMOS DETERMINÍSTICOS por gozarem da propriedade de cada operação ser unicamente definida. Todos os algorítmos que podem ser escritos numa linguagem acessível a um computador comum são ditos determinísticos.

A classe de todos os problemas que podem ser solucionados através de um algorítmo determinístico em tempo polinomial é chamada classe "$\mathcal{P}$".

Sabe-se que os algorítmos enumerativos crescem exponencialmente com o número de jobs e máquinas, tornando-se às vezes, intratáveis em computadores comuns. A seguinte pergunta poderia ser lançada: Existe algum algorítmo capaz de resolver estes problemas em

tempo polinomial num computador comum? Durante décadas, cientistas tentaram dar uma resposta afirmativa para esta pergunta não alcançando bom resultado, daí, eles acreditarem na não existência destes algoritmos, principalmente pelo fato da existência de uma teoria chamada NP-Completa na qual argumentos mais fortes são dados para tal afirmação. Antes de se desenvolver a teoria NP-Completa, vai-se introduzir o conceito de um algorítmo "NÃO DETERMINÍSTICO" muito importante nesta teoria. Para isto, eles pensaram em uma máquina fictícia, isto é, uma máquina não determinística capaz de resolver problemas através de algorítmos não determinísticos em tempo polinomial.

Um algorítmo é dito NÃO DETERMINÍSTICO quando se fizerem presentes, as seguintes funções:

i) Choice (σ) onde σ é um conjunto finito. Esta operação choice (σ) escolhe arbitrariamente um elemento de σ. Seja por exemplo o conjunto $\sigma = \{J_1, \ldots, J_n\}$. Esta operação pode ser representada do seguinte modo: $X \leftarrow$ choice $\{J_1, \ldots J_n\}$.

ii) Success, se a conclusão foi bem sucedida.

iii) Failure, se a conclusão foi mal sucedida.

Uma máquina não determinística é uma máquina ficitícia que executa algorítmos não determinísticos do seguinte modo:

Se dentre os elementos das funções "choice" existir uma sequência que conduz para um sucesso, então o algorítmo deverá seguir esta sequência para concluir o problema com sucesso. O algorítmo termina com insucesso, se não existir tal sequência. Para melhor entendimento de um algorítmo não determinístico e consequentemente de uma máquina não determinística, o problema de Knapsack, definido a seguir, será discutido.

O problema de Knapsack é definido do seguinte modo:

Dados inteiros positivos b, n, a_i para i= 1,...,t encon

trar $x_i \in \{0,1\}$ para i=1,...,t de tal maneira que $a_1x_1 + \ldots + a_tx_t = b$.

Um algorítmo para este problema seria:

```
begin
1.      For i ← 1 to t do
            x_i ← choice {0,1} ;
2.      if Σ_{i=1}^{t} a_i x_i = b      then
3.          Success
                                        else
4.          failure ;
5.      end.
```

Obviamente neste algorítmo, todas as funções de um algorítmo não determinístico, dadas anteriormente, estão presentes nele.

Tomando-se como base este algorítmo, vai-se agora explicar com mais detalhes uma maneira de encontrar os resultados de um algorítmo não determinístico em uma máquina não determinística:

Para i=1, a função "choice" tem duas possibilidades ' 0 ou 1 para o valor de x_1 então duas máquinas idênticas são criadas. Numa o número 0 (zero) é o escolhido e na outra o número 1. Ambas começam a trabalhar paralelamente executando o mesmo algorítmo. Para ' i=2, cada uma das duas máquinas criadas no passo 1 têm novamente ' duas possibilidades, sendo criadas portanto neste passo, 4 máquinas. O algorítmo continua até que uma das máquinas execute a função '

"success". Se nenhuma máquina executar tal função, o algorítmo termina após todas as máquinas terem executado a função "failure". Este procedimento poderá ser representado pela seguinte árvore:

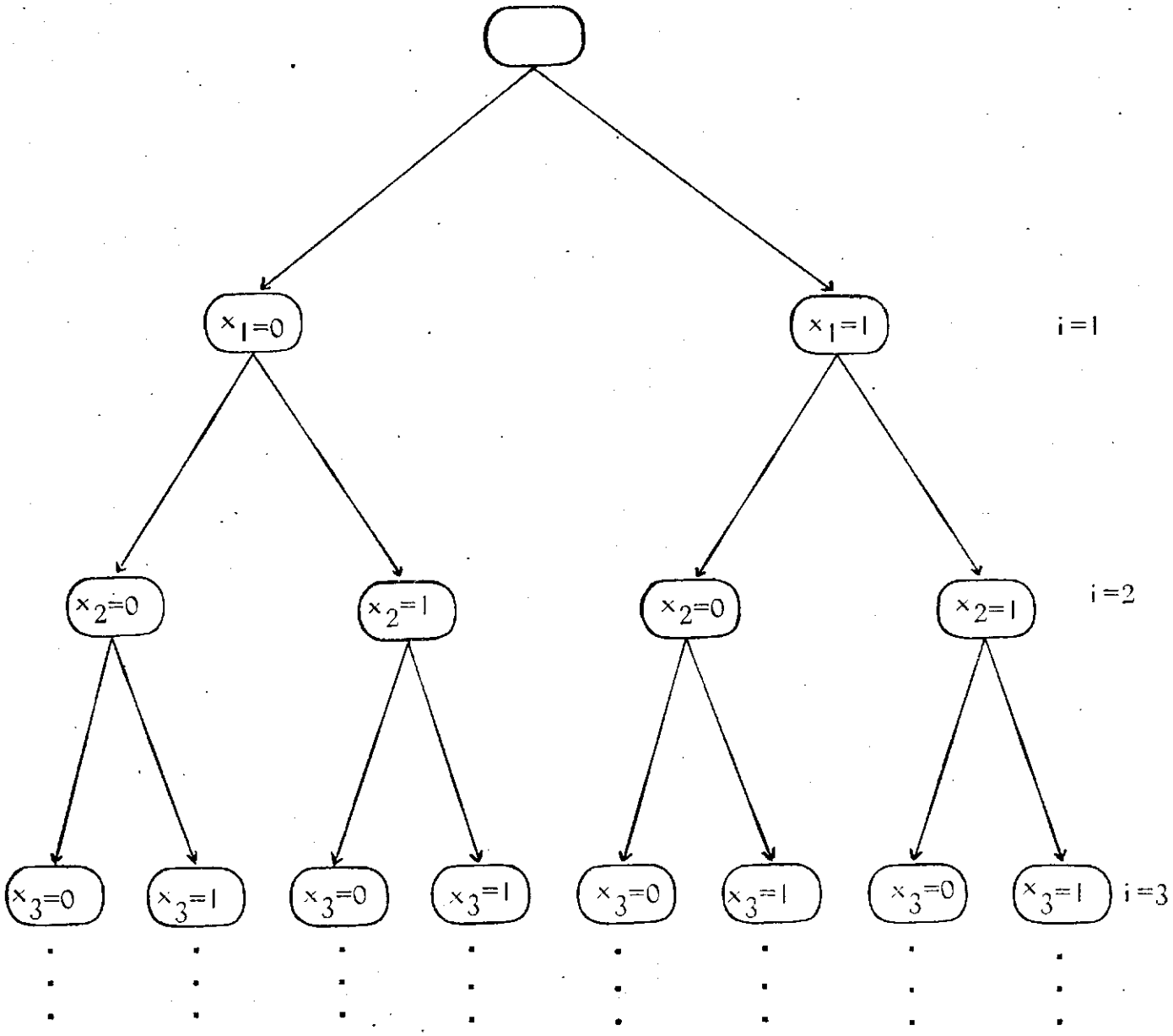


FIGURA 5.1

Obviamente no passo i=t tem-se 2^t máquinas trabalhando em paralelo. Deste modo tem-se todas as possíveis sequências dos valores de x_i, verificando-se aquela (se existir) que tornará

$$\sum_{i=1}^{t} a_i x_i = b$$

A complexidade deste algorítmo é definido por t (nº de passos) mais o tempo gasto pelas 2^t máquinas, trabalhando em paralelo, para determinar a sequência ótima. Consequentemente, trata-se de um algorítmo não determinístico solúvel em tempo polinomial.

Genericamente, a ordem de complexidade de um algorítmo não determinístico é dada pelo menor número de passos necessários para atingir uma solução "success". Se não exisitir uma sequência que conduza a tal solução, a complexidade é dada pelo maior número de passos empregados na determinação de todas as soluções "failure".

A classe de todos os problemas que poderem ser resolvidos através de um algorítmo não determinístico em tempo polinomial é chamada "classe NP". Como exemplo de um problema NP, tem-se o problema de Knapsack, visto acima. Outro problema NP é o problema "satisfiability" o qual determina se uma expressão boolean é verdadeira.

Um conceito de vital importância na teoria da complexidade é o conceito da redutibilidade definido a seguir:

P_1 é redutível para P_2 se e somente se existir um algorítmo A que resolve P_1 com as seguintes propriedades:

1. O algorítmo A usa como subrotinas, um algorítmo que resolve P_2.
2. A complexidade do algorítmo A é polinomial sem se contar o trabalho utilizado pelas subrotinas que resolvem P_2.

Portanto se for possível resolver P_2 em tempo polinomial, a complexidade de A será também polinomial.

Outro conceito importante desta teoria é o conceito da equivalência. Dois problemas P_1 e P_2 são ditos equivalentes (notação $P_1 \sim P_2$) se $P_1 \propto P_2$ e $P_2 \propto P_1$.

A partir destes conceitos, pode-se agora definir os problemas fundamentais da teoria da complexidade que são chamados "NP-Completos". Todos os problemas equivalentes ao problema "satisfiability", definido anteriormente, são chamados "NP-Completos". Os problemas "NP-Completos" são todos equivalentes entre sí, isto é, se exisitir um algorítmo polinomial para um deles, é possível construir algorítmos também polinomiais para cada um, mas como foi dito anteriormente, os cientistas acreditam na não existência destes algorítmos.

O teorema a seguir é o mais importante da teoria NP-Completa. Este teorema foi formulado por S.A Cook [6].

TEOREMA 5.1 - Todos os problemas em NP são redutíveis para o problema "satisfiabiiity", (ver demonstração [1], [6]).

A partir deste Teorema, surge o importante corolário.

Corolário 5.1 - Um problema P é NP-Completo se e somente se:

i) $P \in NP$

ii) Existe um problema NP-Completo P' com $P' \propto P$.

$\Rightarrow$ P é NP-Completo então $P \sim$ "satisfiability". Portanto P satisfaz i) e ii)

$\Leftarrow$ Partindo-se do princípio de que i) e ii) são satisfeitos, então por i) e teorema 5.1, $P \propto$ "satisfiability" e por ii) "satisfiability" $\sim P' \propto P$, logo $P \sim$ "satisfiability" então P é NP-Completo.

Nesta unidade, os problemas até agora discutivos foram problemas de decisão ou seja problemas que têm como solução uma das respostas "success" ou"failure". Este fato é justificável visto que estar-se interessados principalmente na classe NP cujos problemas são solucionados por algorítmos não determinísticos, os quais têm estas palavras como palavras chave. A seguir serão tratados problemas "Flow-Shop" e "Job-Shop" que comumente são formulados como problemas de otimização. Aqui eles deverão ser formulados como problemas de decisão, para poder-se demonstrar certos fatos de interesse deste trabalho. Esta formulação pode ser feita do seguinte modo:

Dado um problema de minimização $f(x) \leftarrow \min$, o problema de decisão correspondente seria: existe um X possível, tal que $f(x) \leq Y$[1]

Na unidade 2 a seguir, serão relacionados alguns probelmas "Flow-Shop" e "Job-Shop" pertencentes às classes P e NP-Completa.

Também nesta unidade alguns problemas "Flow-Shop" e "Job-Shop" com diferentes funções objetivas são demonstrados serem NP-Completos. Para isto, usa-se o fato de que Knapsack é redutível neles. A demonstração de que Knapsack é um problema é NP-Completo [15] será omitida deste trabalho.

5.2 - RESULTADOS DA COMPLEXIDADE DE PROBLEMAS "FLOW-SHOP" E "JOB-SHOP".

Aqui será listada uma série de problemas "Flow-Shop" e "Job-Shop" pertencentes às classes P e NP-Completa. De acordo com a lista a seguir, de todos os problemas "Flow-Shop" mais importantes, o problema $n/2/F/r_j=0/C_{max}$ é o único pertencente à classe P. Qualquer alteração em um dos seus parâmetros o tornará NP-Completo.

---

1 - y é um limite superior para f(X).

Quanto aos problemas "Job-Shop", também são poucos os pertencentes à classe $\mathcal{P}$ e pequenas alterações também os transforma - rão em problemas NP-Completos. As referências para cada problema são dadas ao lado deles.

Classe $\mathcal{P}$	Classe NP-Completa
$n/2/F/r_j=0/C_{max}$ [14]	$n/2/F/r_j\ \ 0/C_{max}$ [18]
	$n/3/F/r_j=0/C_{max}$ [18]
	$n/2/F/r_j=0/\sum C_j$ [7]
	$n/2/F/r_j=0/\sum T_j$ [18]
	$n/2/F/r_j=0/L_{max}$ [18]
	$n/2/F/r_j=0/\sum U_j$ [18]
$n/2/J/r_j=0, n_j \leq 2/C_{max}$ [13]	$n/2/J/r_j=0,\ n_j \leq 3/C_{max}$ [18]
	$n/3/J/r_j=0,\ n_j \leq 2/C_{max}$ [18]
$n/2/J/r_j=0, t_{jki}=1/C_{max}$ [11]	$n/2/J/r_j=0, t_{jki}=1,2/C_{max}$ [19]
	$n/3/J/r_j=0, t_{jki}=1/C_{max}$ [19]
$2/m/J/r_j=0/C_{max}$ [20]	

Agora vai-se escolher os problemas "Flow-Shop"e "Job--Shop" $n/3/F/r_j=0/C_{max}$, $n/2/J/n_j \leq 3/C_{max}$ e $n/2/F/r_j=0/L_{max}$ da lista NP-Completa para demonstrar-se tal fato.

Nota: Por conveniência vai-se representar estes problemas pela letra P.

Esta demonstração envolve 2 provas:

i) $P \in \mathcal{NP}$.

ii) Existe um problema P' NP-Completo com $P' \propto P$.

Prova de i) É possível construir um algorítmo de enumeração total para resolver P. Como já foi visto antes, no desenvolvimento de um algorítmo enumerativo para problemas "Flow-Shop" e "Job-Shop" é gerada uma árvore, onde cada nodo representa uma "Schedule" parcial. Pensando-se em termos de um algorítmo não determinístico, sabe-se que estes problemas são solúveis em tempo polinomial conforme' definição dada para ordem de complexidade de algorítmos não determinísticos.

Prova ii) Considerando-se o problema NP-Completo P' como sendo o problema de Knapsack, então deve-se provar que Knapsack é redutível para os seguintes problemas:

a) $n/2/J/n_j \leq 3/C_{max}$

b) $n/3/F/r_j = 0/C_{max}$

c) $n/2/F/r_j = 0/L_{max}$

A idéia desta demonstração consiste em para cada problema "Flow-Shop" e "Job-Shop" acima, formular um correspondente problema particular P baseado nos dados do problema de Knapsack e determinar também um limite superior y para a função objetiva f de P. Então Knapsack tem solução $\Longleftrightarrow$ P tem uma solução S com $f(S) \leq y$.

Seja $A = \sum_{j=1}^{t} a_j$, onde a_j é definido no problema de Knapsack na unidade 5.1.

Para simplificar a notação, vai-se considerar o conjunto dos j's tais que $x_j=1$ como sendo um conjunto U e o conjunto de todos os j's como sendo $T(T=\{J_1,\ldots,J_t\})$.

a) Knapsack $\propto$ $n/2/J/n_j \leq 3/C_{max}$.

Sejam $n = t + 1$

$t_{j11} = a_j \ (j \in T)$

$t_{n12} = b,\ t_{n21}=1,\ t_{n32} = A-b$

$y = A + 1$

$\Longrightarrow$ Se Knapsack tem solução, uma "schedule" ótima para este problema será a "schedule" da fig. 5.2 com valor $C_{max}=A+1=y$

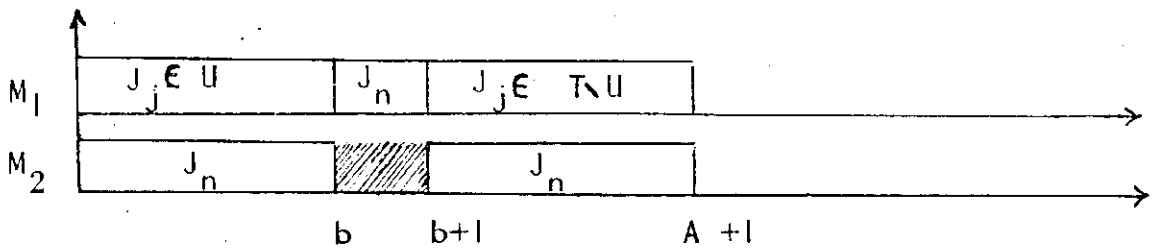


FIGURA 5.2

$\Longleftarrow$ Se Knapsack não tem solução então não existe uma "schedule" com valor $C_{max} \leq y$.

Considere uma "schedule" ótima e seja U o conjunto dos jobs processados pela máquina M_1 antes do job J_n, então

$\sum_{j\in U} a_j \neq b$ logo $\sum_{j\in U} a_j - b = c \neq 0$ e portanto :

caso 1 : $c > 0$

$$C_{max} \geq \sum_{j \in U} t_{j11} + t_{n21} + t_{n32} = b+c+1+A-b=A+c+1 > y .$$

caso 2 : $c < 0$

$$C_{max} \geq t_{n12} + t_{n21} + \sum_{j \in T-U} t_{j11} = b+1+A-b-c =$$

$$= A - c + 1 > y .$$

b) Knapsack $\propto$ $n/3/F/r_j=0/C_{max}$

Sejam $n=t+1$

$t_{j1}=1$, $t_{j2}= ta_j$, $t_{j3}=1$ $(j \in T)$

$t_{n1} = tb$, $t_{n2}=1$, $t_{n3} = t(A-b)$

$y = t(A+1) + 1$.

$\Longrightarrow$ Se Knapsack tem solução , uma "schedule" ótima para este problema será a "schedule" da figura 5.3 com valor $C_{max} = t(A+1)+ 1= y$

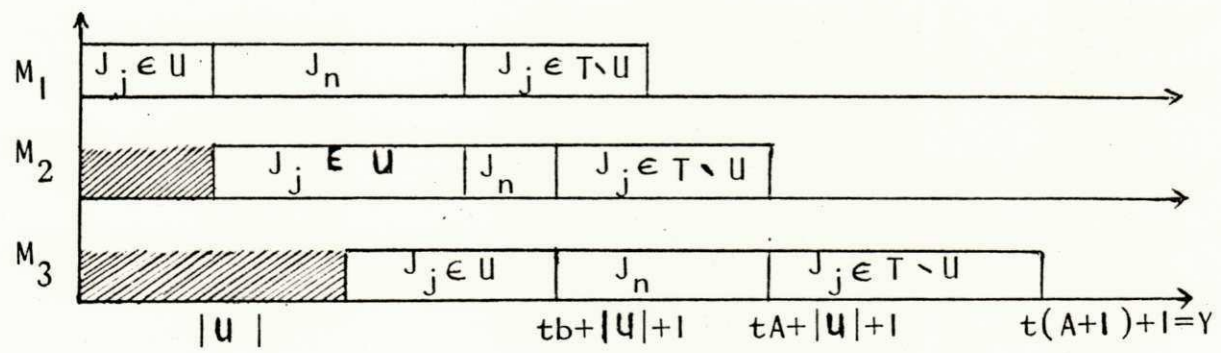


FIGURA 5.3

$\Longleftarrow$ Knapsack não tem solução, então não existe uma "schedule" com valor $C_{max} \leq y$.

Considere uma "schedule" ótima e seja U o conjunto dos jobs processados pela máquina M_2 antes de J_n, então:

$$\sum_{j \in U} a_j \neq b \quad \text{logo} \quad \sum_{j \in U} a_j - b = c \neq 0 \text{ e portanto:}$$

caso 1 : $c > 0$

$$C_{max} > \sum_{j \in U} ta_j + t_{n2} + t_{n3} = t(b+c) + 1 + t(A-b) =$$

$$= t(A+c) + 1 \geq y.$$

caso 2 : $c < 0$

$$C_{max} > t_{n1} + t_{n2} + \sum_{jET-U} ta_j = tb+1+1+t(A-b-c) = t(A-c)+1 \geq y.$$

c) Knapsack $\propto$ $n/2/F/r_j=0/L_{max}$

Sejam $n=t+1$ $\quad b > 1$ (para $b=1$ o problema é trivial)

$t_{j1}=1$, $t_{j2}=ta_j$, $d_j = t(A+1) + 1 \quad j \in T$

$t_{n1} = tb$, $t_{n2}=1$, $d_n = t(b+1)+1$

$y=0$

$\Longrightarrow$ Se Knapsack tem solução, então uma "schedule" ótima para este problema será a "schedule" da figura 5.4

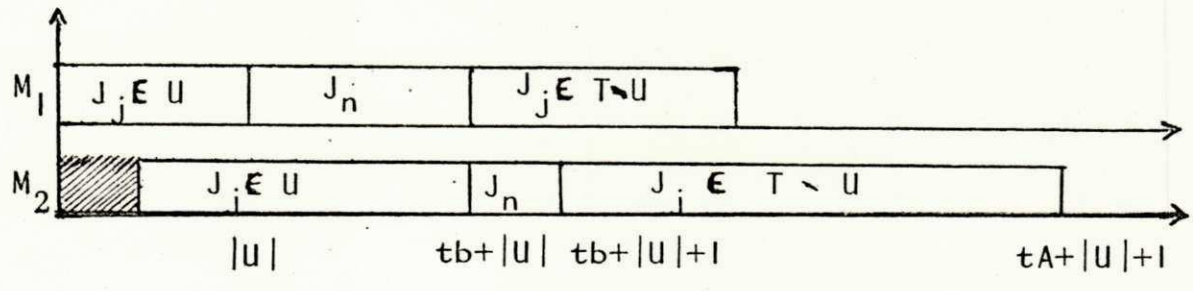


FIGURA 5.4

Para J_n, tem-se $C_n = \sum_{j \in U} t_{j1} + t_{n1} + t_{n2} = |U| + tb + 1 \leq t + tb + 1 = d_n$

Para J_j, tem-se $C_j \leq \sum_{j \in U} t_{j1} + t_{n1} + t_{n2} + \sum_{j \in T-U} t_{j2} = |U| + tb + 1 + t(A-b) \leq t + tb + 1 + t(A-b) = t + 1 + tA = t(A+1) + 1 = d_j$.

$\Leftarrow$ Se Knapsack não tem solução, então não existe uma "Schedule" com $L_j \leq 0, j = 1, \ldots, n$

Considere uma "schedule" ótima e seja U o conjunto dos jobs processados antes de J_n na máquina M_2, então $\sum_{j \in U} a_j - b = c \neq 0$ e daí,

caso 1 : $c > 0$

$$\sum_{j \in U} t_{j1} + t_{n1} + t_{n2} = t(b+c) + tb + 1 \geq t(b+1) + 2 > d_n$$

logo $L_n > 0$ - contradição

caso 2 : $c < 0$ e $U \neq \emptyset$

$$\sum_{j \in U} t_{j1} + t_{n1} + t_{n2} + \sum_{j \in T-U} t_{j2} = |U| + tb + 1 + t(A-b-c) \geq 1 + b + 1 + t(A-b-c) = t(A-c) + 2 > d_j \text{ para todo } j$$

então $L_j > 0$ - contradição

Se $U = \emptyset$ não exisitirá nenhuma "schedule" ótima, visto que, para qualquer ordem em que os jobs J_j forem processados tem-se sempre uma "schedule" (ver fig.5.5) onde $C_j = tb + 1 + tA = t(A+b) + 1 > d_j$ para algum $j \in T$.

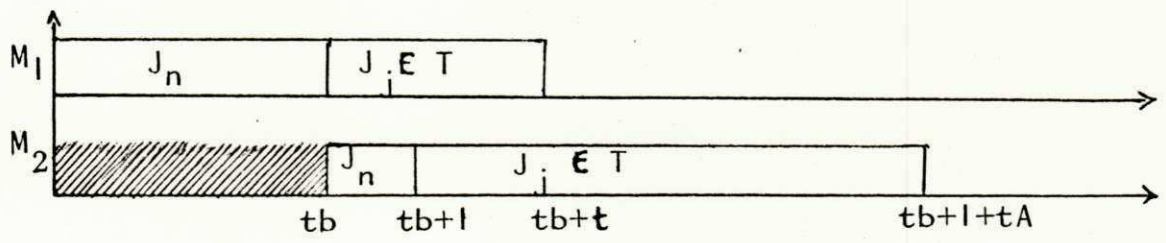


FIGURA 5.5

C A P Í T U L O V I

APROXIMAÇÕES HEURISTÍCAS

6.1 - MÉTODOS HEURÍSTICOS PARA PROBLEMAS "FLOW-SHOP".

No capítulo 4, viu-se o método enumerativo "Branch and Bound" e com ele, vitais desvantagens.

Aqui, serão tratados os métodos Heuristícos para pro - blemas "Flow-Shop", os quais evitam tais desvantagens, contudo apresentam outras, como por exemplo, a incerteza quanto à qualidade da solução. Aqui só se estar interessado nas "Schedules" permutação.

Serão abordados nesta unidade, três algorítmos heurísticos. O primeiro foi desenvolvido por Palmer. Os dois últimos se baseam no algorítmo 2.1, sendo que o segundo foi abordado por Gupta e o terceiro por Campbell, Dudek e Smith, também conhecido por algorítmo CDS.

i) Palmer seguiu a seguinte linha na descrição de seu algorítmo: associar a cada job J_j um peso a partir da expressão

$$S(j) = \sum_{k=1}^{m} (m+1-2k)t_{j,m+1-k} \qquad \text{onde } j=1,\ldots,n$$

são os números dos jobs e m a quantidade de máquinas. Após determinar S(j) para cada job J_j, sequenciá-los utilizando os valores de S(j) de maneira não crescente. Este algorítmo não está definido de maneira única, pois pode ocorrer de S(k)=S(l)k≠l. Para torná-lo único, deve-se colocar o job J_k na sequência antes do job J_l, somente quando k < l.

A determinação de todos os valores S(j) é feita em O(m.n) operações. A ordenação dos jobs em O(nlogn) operações e portanto a complexidade deste algorítmo é da ordem max{m.n, nlogn} .

Este algorítmo será ilustrado através do exemplo 6.1 abaixo:

Exemplo 6.1 - Utilizando-se do algorítmo de Palmer, determinar uma solução para os problemas com os dados especificados na tabela 6.1 abaixo:

TABELA 6.1

job / maq	J_1	J_2	J_3	J_4	J_5
M_1	1	4	2	6	3
M_2	4	3	1	3	5
M_3	2	2	3	2	5
M_4	5	3	4	5	1

Como m=4, S(j) toma a seguinte forma:

$$S(j)= \sum_{k=1}^{4} (5-2k)t_{j,5-k}$$ e portanto para j=1,...,5, tem-se:

$$S(1)=3t_{14}+t_{13}-t_{12}-3t_{11}=10$$

Analogamente,

S(2)=-4, S(3)=8, S(4)=-4 e S(5)=-6

A sequência ótima para este exemplo é $J_1 J_3 J_2 J_4 J_5$ cuja representação gráfica é mostrada na figura 6.1 abaixo:

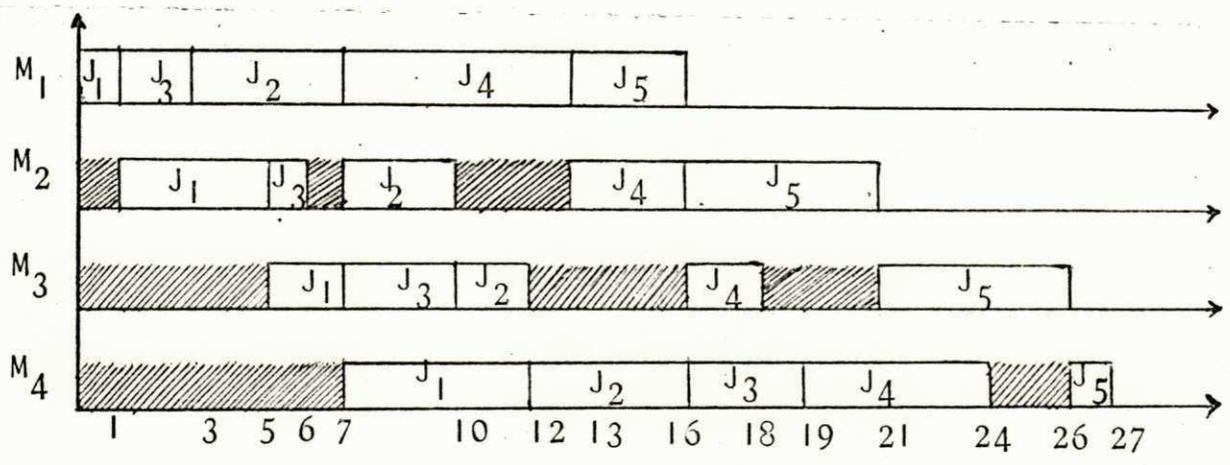


FIGURA 6.1 - "Schedule" construída a partir da sequência obtida pelo algorítmo de Palmer.

ii) O algorítmo aqui discutido é o algorítmo de Gupta. Gupta aplicou seu algorítmo e o algorítmo de Palmer em um número muito grande de problemas, e na maioria dos casos seu algorítmo demonstrou maior eficiência.

Gupta para desenvolver seu algorítmo heurístico, tomou como base a extensão do algorítmo 2.4 para 3 máquinas.

Com base neste fato, Gupta generalizou o algorítmo 2.4 do seguinte modo:

$$S(j) = \frac{e_j}{\min\limits_{1 \leq k \leq m-1} \{t_{jk} + t_{j,k+1}\}} \quad \text{onde}$$

$$e_j = \begin{cases} 1 & \text{se } t_{j1} < t_{jm} \\ -1 & \text{se } t_{j1} \geq t_{jm} \end{cases} \qquad j=1,\ldots,n$$

A sequência dos jobs é obtida utilizando-se os valores de S(j) de maneira não crescente. As restrições para a ordenação dos S(j) são as mesmas feitas no algorítmo de Palmer.

A ordem de complexidade deste algorítmo, também é dada por max (m.n, nlogn).

Exemplo 6.2 - Aplicando-se o algorítmo de Gupta nos dados da tabela 6.1 tem-se

$$S(1) = \frac{e_1}{\min\{t_{11}+t_{12},\ t_{12}+t_{13},\ t_{13}+t_{14}\}} = \frac{1}{5}$$

Analogamente,

$S(2) = -1/5$, $S(3)=1/3$, $S(4)=-1/5$ e $S(5)=-1/6$.

A sequência para os jobs é $J_3J_1J_5J_2J_4$ cuja representação gráfica é:

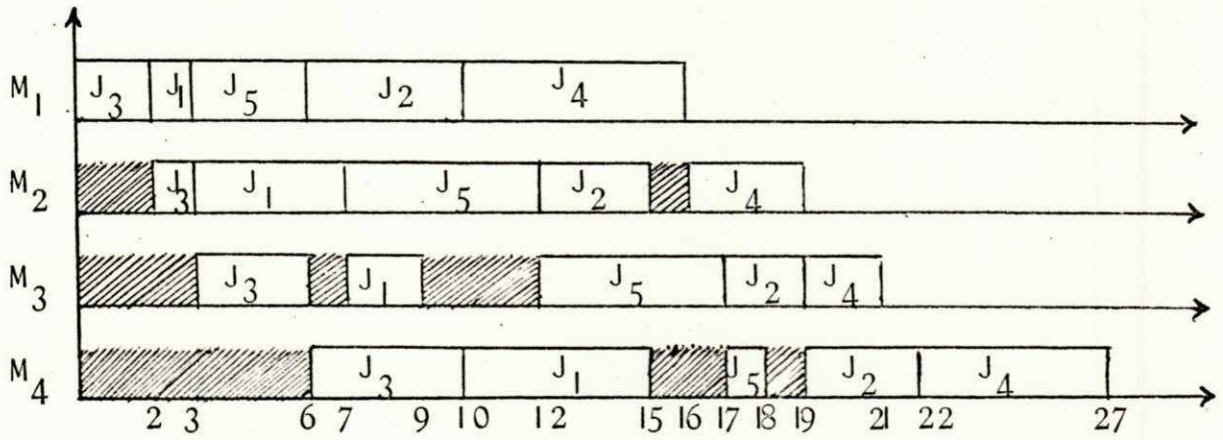


FIGURA 6.2 - "Schedule" construída a partir da sequência obtida pelo algorítmo de Gupta.

iii) O terceiro e último algorítmo heurístico para problemas "Flow-Shop" que será discutido neste trabalho, é o algorítmo CDS. Talvez este é o mais eficiente algorítmo heurístico para problemas "Flow-Shop". Isto é bem racional devido aos dois fatos abaixo:

i) O uso da regra de Johnson[1] cada vez que o algorítmo é ativado.

ii) Durante a execução do algorítmo, muitas "Schedules" são criadas podendo-se escolher a melhor.

Algorítmo CDS (Algorítmo de Campbell, Dudek e Smith)

Para clarificar, usa-se S_{ji} em vez de t_{ji} como sendo o tempo de processamento do job J_j na máquina M_i

1. For $i \leftarrow 1$ to m-1 do
   begin
2. $t_{j1} \leftarrow \sum_{k=1}^{i} S_{jk}$;
3. $t_{j2} \leftarrow \sum_{k=1}^{i} S_{j,m-k+1}$;
4. Calcular a sequência S_i e o valor da função objetiva f_i com o algorítmo 2.1 de Johnson, usando os valores t_{ji};
   end;
5. Escolher a melhor sequência S_i*.

---

1 - ver unidade 2.2

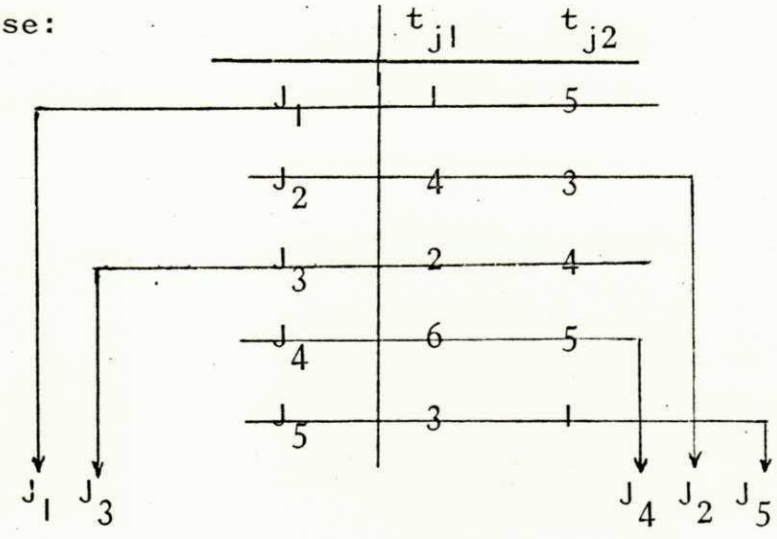
A ordem de complexidade deste algorítmo é m vezes a ordem de complexidade do algorítmo de Johnson, ou seja m.nlogn.

Campbell, Dudek e Smith aplicaram o algorítmo CDS em um número muito grande de problemas e após compararem estes resultados com os resultados obtidos pelo algorítmo de Palmer, concluiram que seu algorítmo é mais eficiente tanto para problemas grandes (muitas máquinas) como para problemas pequenos.

Este algorítmo será ilustrado no exemplo a seguir:

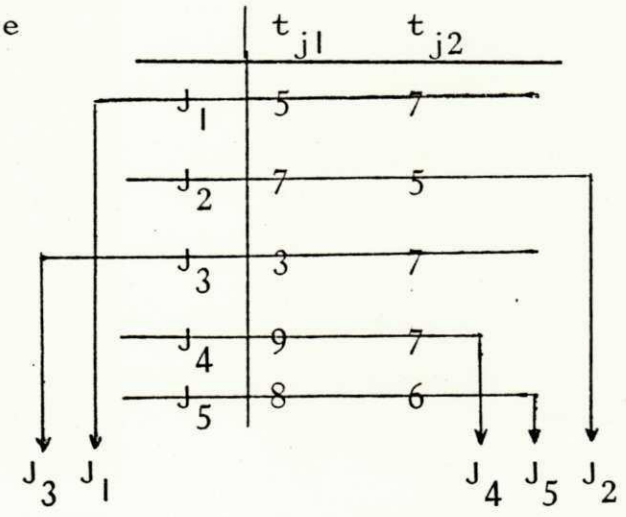
Exemplo 6.3 - Aplicar o algorítmo CDS nos dados da tabela 6.1

Para i=1, tem-se:



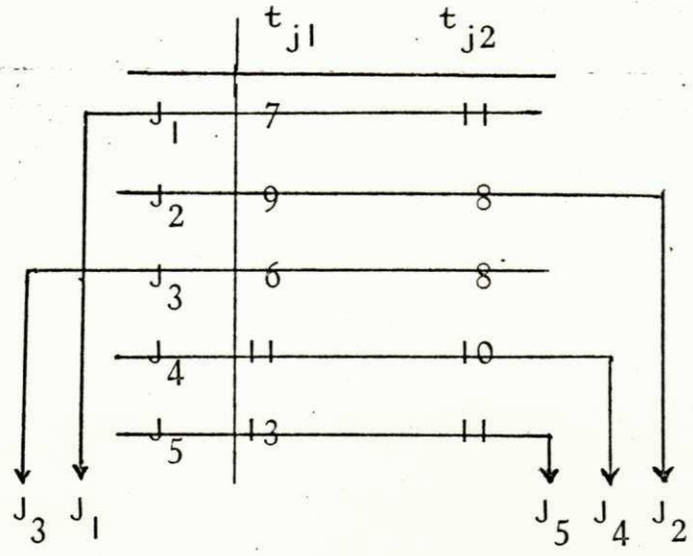
$S_1=J_1J_3J_4J_2J_5$ com $C_{max}=28$

Para i=2, tem-se



$S_2 = J_3 J_1 J_4 J_5 J_2$ com $C_{max} = 27$

Para i=3, tem-se



$S_3 = J_3 J_1 J_5 J_4 J_2$ com $C_{max} = 29$

A melhor sequência foi a S_2 cuja representação gráfica, segue-se:

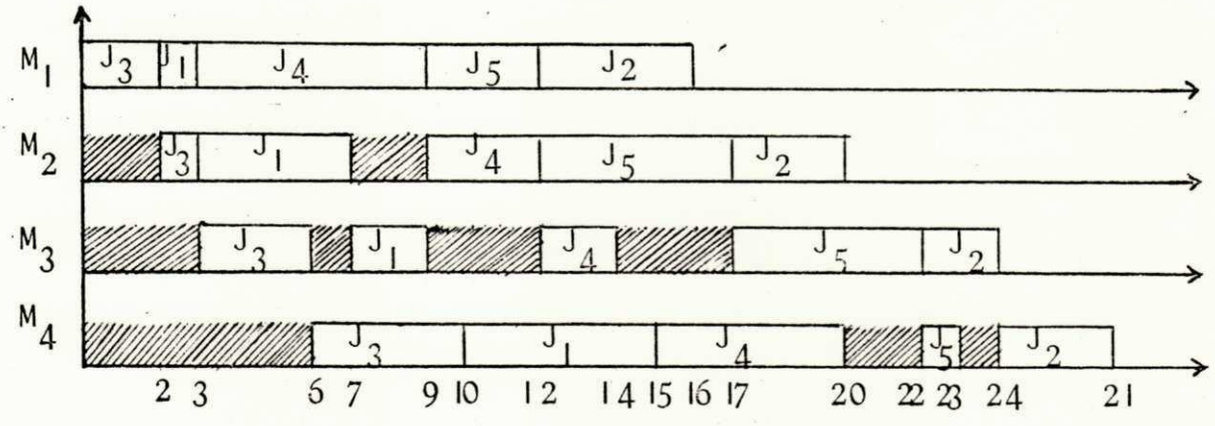


FIGURA 6.3 "Schedule" construída a partir da sequência S_2 obtida pelo algorítmo CDS.

Analisando-se as sequências determinadas pelo 3 algorítmos, constata-se que neste exemplo, todas deram o mesmo resultado.

6.2 - MÉTODOS HEURÍSTICOS PARA PROBLEMAS "JOB-SHOP".

Viu-se na unidade 4.2 que os métodos enumerativos para problemas "Job-Shop" não são aconselháveis, dado o seu crescimento exponencial com os dados[2] do problema, tornando-se às vezes, impossíveis de serem solucionados. Nesta unidade, serão tratados os métodos heurísticos que evitam tais problemas.

Enquanto no método enumerativo "Branch and Bound" em cada passo são criadas várias "schedules" parciais para atingir-se àquela com N elementos e C_{max} mínimo, no procedimento heurístico aqui desenvolvido, é criada apenas uma, ou seja, aquela que conduz à solução viável atentando-se para uma das regras de prioridade dadas abaixo:

SPT (Shortest Processing Time) - selecionar a operação de menor tempo de processamento.

MWK (Most Work Remaining) - selecionar a operação do job que tiver o maior trabalho restante para ser processado.

MOPNR (Most Operation Remaining) - selecionar a operação que tiver o maior número de sucessores.

LWKR (Lest Work Remaining) - selecionar a operação do job que tiver o menor tempo de trabalho restante para ser processado.

RANDOM (randon) - escolher a operação aleatoriamente.

---

2 - número de jobs e máquinas.

NOTA: existem outras regras de prioridade mas são de menor interesse e por isso serão omitidas deste trabalho.

Jeremiah Lalchandani e Schrage fizeram um estudo de todas estas regras de prioridade para a escolha da mais eficiente. Eles observaram que com o objetivo C_{max}, é impossível determinar precisamente a melhor, muito embora tenham observado que na maioria dos problemas, a melhor "schedule" é obtida através da regra de prioridade' MWKR ou uma de suas variações.

Estes cientistas também analisaram outros tipos de funções objetivas as quais não serão tratadas aqui.

O algorítmo heurístico que será aqui discutido, é aplicado para o mesmo tipo de problema descrito na unidade 4.2. Todas as variáveis, que serão usadas neste algorítmo, já foram definidas no ' algorítmo 4.2 com excessão de n(S) que significa o número de operações processadas em S.

ALGORÍTMO 6.2

$S \leftarrow \lambda$

While $n(S) \leq N$ do

begin

Forme o conjunto O(S) com todas as operações ' O_{jki} processáveis em S;

$\emptyset^*_{j'k'i'} \leftarrow \min_{O_{jki} \in O(S)} \{ \emptyset_{jki} \}$;

While $O(S) \neq \emptyset$ do

begin

Escolher O_{jki}, $\in O(S)$

if $\sigma_{jki}, < \emptyset^*_{jki}$, then

```
                Utilizar uma regra de prioridade especifica
                da, para a escolha da operação que deverá '
                ser acrescentada à "schedule" S;

            end;

        end.
```

As restrições para efeito da programação, se comportam' exatamente como no algorítmo 4.2 .

Exemplo 6.4 - Aplicar este algorítmo heurístico nos dados das tabelas 4.2(a) e (b) e utilizar a regra de prioridade MWKR ' para a escolha da operação que deverá ser processada em cada passo. Em caso de operações com a mesma prioridade, utilizar (somente nestes casos) a regra de prioridade SPT.

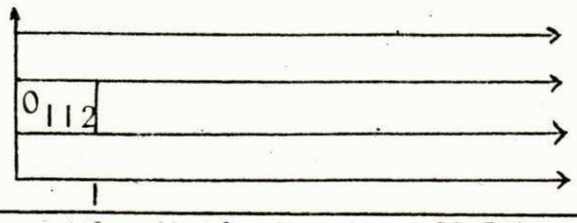
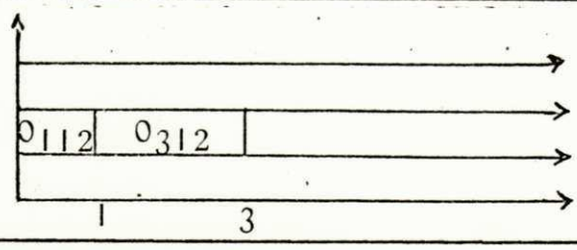
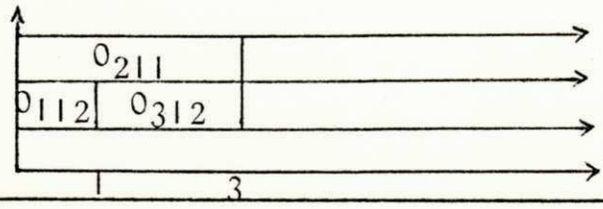
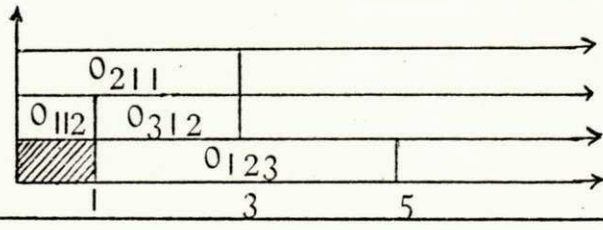
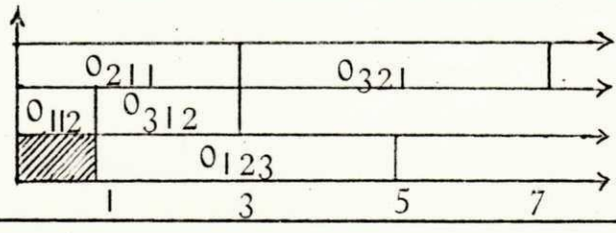
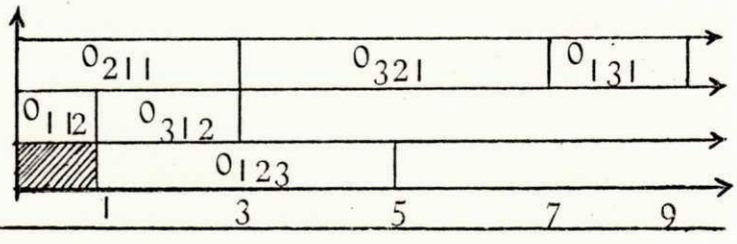
Passos		MWKR (J_j)	"Schedule parcial" S
0	O_{112}	7*	
	O_{211}	-	
	O_{312}	6	
1	O_{123}	-	
	O_{211}	3	
	O_{312}	6*	
2	O_{123}	-	
	O_{211}	3*	
	O_{321}	-	
3	O_{123}	6*	
	O_{321}	-	
4	O_{131}	2	
	O_{321}	4*	
5	O_{131}	2*	

FIGURA 6.4

Como a "schedule" S de L no passo 5 tem N elementos, então de acordo com este algorítmo heurístico esta deverá ser a escolhida.

C A P Í T U L O V I I

ANÁLISE DO PIOR CASO PARA PROBLEMAS "FLOW-SHOP" COM OBJETIVOS C_{max} E $\sum C_j$.

Em capítulos anteriores estudou-se os problemas "Flow-Shop", sua complexidade, algorítmos exatos para casos especiais, algorítmos enumerativos e suas respectivas desvantagens, soluções aproximadas de um problema "Flow-Shop" mediante métodos heurísticos e agora analisar-se-ão as piores "schedules" obtidas mediante simples algorítmos de aproximação (algorítmos heurísticos). Para isto, serão estabelecidos limites superiores para a razão entre o valor da função objetiva de uma "schedule" obtida por um processo heurístico qualquer e o valor da função objetiva da "schedule" ótima do problema.

Por simplicidade, na análise da pior " schedule " serão consideradas apenas as " busy schedules ", isto é, " schedules " onde não ocorre ociosidade em pelo menos uma máquina do problema. Também nesta análise só serão tratadas as funções objetivas C_{max} e $\sum C_j$ definidas no capítulo I. Uma " schedule" qualquer é notada por S e a " schedule " ótima por S^* . Tambem por conveniência de notação FT (S) (finish time)representa o tempo

de conclusão do último job do problema na "schedule" S ($FT=C_{max}$) e MFT(S) (mean flow time) representa a média entre os tempos de conclusão de todos os jobs na "schedule" S ($MFT = \frac{1}{n} \sum_{j=1}^{n} C_j$).

Durante este capítulo certas heurísticas são utilizadas e por isso devem ser definidas aqui. Para isto precisa-se definir tempo de processamento de um job. Define-se tempo de processamento de um job J_j como sendo a soma dos tempos requeridos por cada operação deste job, isto é, $\sum_{k=1}^{m} t_{jk}$.

É importante lembrar que neste capítulo só se estar interessado nas "schedules" permutação.

Define-se "LPT - "Schedule" (Largest Processing Time' First) como sendo a heurística que executa primeiro o job com maior' tempo de processamento e SPT - "Schedule" (Shortest Processing time First) aquela que processa os jobs na ordem não decrescente de seus' tempos de processamento.

Nos teoremas seguintes, serão discutidos os limites superiores para as razões $FT(S)/FT(S^*)$ e $MFT(S)/MFT(S^*)$.

7.1 - ANÁLISE DO PIOR CASO ATRAVÉS DE HEURÍSTICAS PARA PROBLEMAS COM OBJETIVO C_{max}.

TEOREMA 7.1 - Sejam S^* uma "Schedule" ótima para um ' problema do tipo $n/m/F/r_j=0/C_{max}$, $(m > 2)$ e S uma "busy schedule" ' qualquer para este problema. Então $FT(S)/FT(S^*) \leq m$.

Prova: Sejam R_j o tempo requerido por todas as operações do job J_j $j=1,\ldots,n$, isto é, $R_j = \sum_{i=1}^{m} t_{ji}$ e $R = \sum_{j=1}^{n} R_j$.

Para qualquer "busy-schedule" S tem-se que $FT(S) \leq R$. Logicamente a "schedule" ótima S^* é tal que $FT(S^*) \geq \frac{R}{m}$ e assim $FT(S)/FT(S^*) \leq m$.

Um procedimento particular de "busy schedule" é o LPT - "schedule". Portanto este teorema também é válido para o processamento heurístico LPT - "Schedule".

Para ilustrar este algorítmo e para mostrar que este limite é o menor possível, usa-se o exemplo particular abaixo:

Exemplo 7.1 - Considere o problema "Flow-Shop" de m máquinas, n jobs e tempos de processamento dados por:

$$t_{ij} = k \qquad 1 \leq i \leq n$$

$$t_{ji} = \varepsilon_j \qquad 1 \leq j \leq n \qquad 1 \leq i \leq m \qquad i \neq j.$$

$$\varepsilon_j > \varepsilon_{j+1} \quad \text{para} \quad 1 \leq j \leq m-1 \quad \text{e} \quad \varepsilon_1 << k.$$

A "schedule" S obtida pelo LPT- "schedule" é dada na figura 7.1 (a)

A "schedule" ótima S^* é dada na figura 7.1 (b):

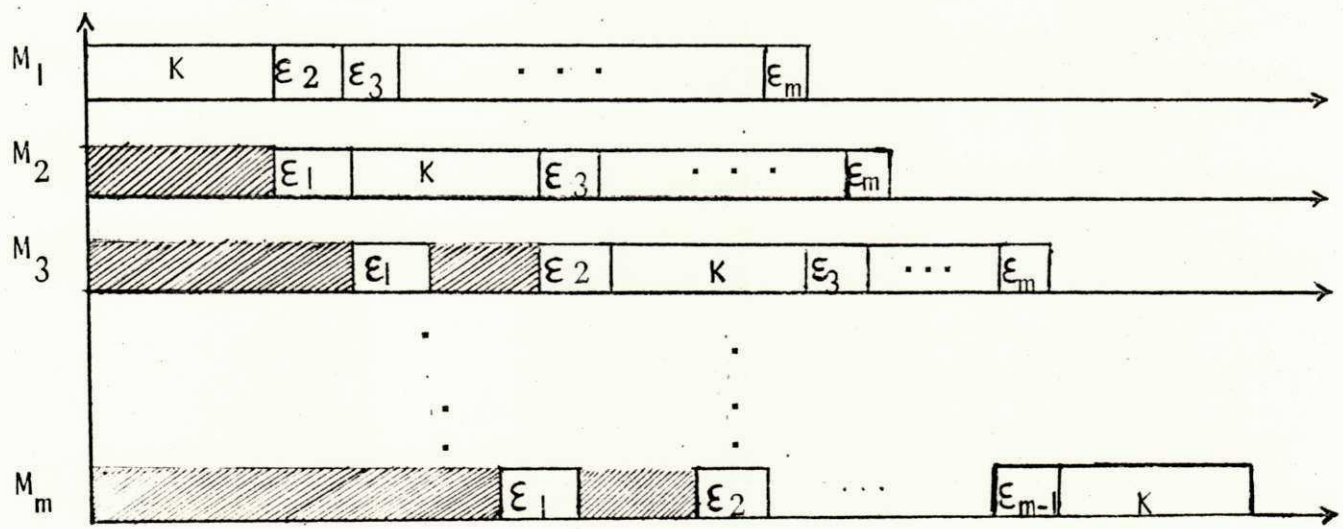


FIGURA 7.1 (a) "Schedule" (permutação) obtida através da heurística LPT-"Schedule".

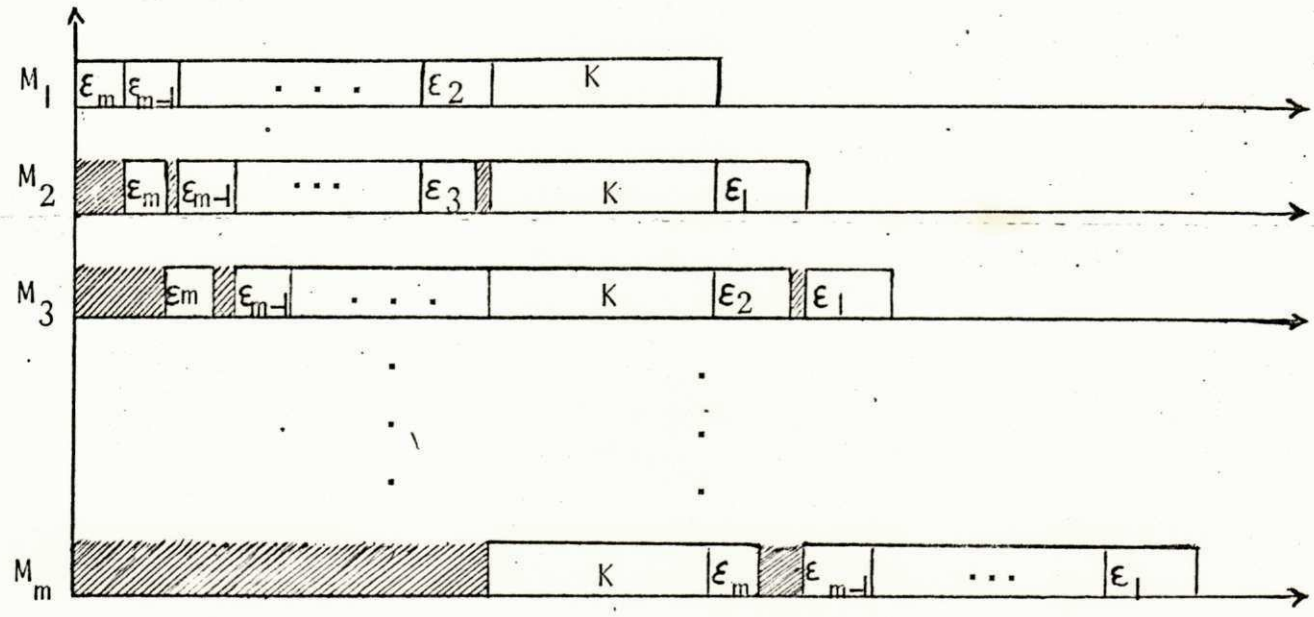


FIGURA 7.1 (b) - "Schedule" ótima para o exemplo 7.1

Nota: Esta "schedule" é ótima porque os jobs dispostos desta maneira dão o menor espaço ocioso possível.

O tempo de conclusão da "schedule" S é $FT(S)=mk+\sum_{j=1}^{m-1} \varepsilon_j$

da "schedule" ótima S^* é $FT(S^*) = \sum_{j=2}^{m} \varepsilon_j + k + (m-1)\,\varepsilon_1$ e

portanto $FT(S)/FT(S^*) \leq \lim \dfrac{mk + \sum_{j=1}^{m-1} \varepsilon_j}{\sum_{j=2}^{m} \varepsilon_j + k + (m-1)\,\varepsilon_1} = m$

$\varepsilon_1 \to 0$

Como $\varepsilon_1 \to 0$ e $\varepsilon_1 < \varepsilon_j \ \forall\, j>1$ então $\varepsilon_j \to 0$ para qualquer j.

Em se tratando da análise do pior caso para problemas "Flow-Shop" que objetivam C_{max} é possível utilizar um outro procedi-

mento heurístico de modo que a razão entre a "schedule" determinada por esta heurística e a "schedule" ótima do problema seja menor ou igual a $\lceil \frac{m}{2} \rceil$[1].

Este procedimento heurístico é denotado por Particionamento-Heurística e consiste em dividir as máquinas em grupos tal que $2k+1$ e $2k+2$, $k \in \{0,\ldots, \lceil m/2 \rceil\}$ pertençam ao mesmo grupo; no caso de um número ímpar de máquinas a última máquina será analisada sozinha. Depois aplicar o algorítmo de Johnson[2] para cada grupo e finalmente concatenar as "schedules". Obviamente esta "schedule" final S geralmente não é uma "schedule" permutação. Com base nestas definições, pode-se agora demonstrar o teorema seguinte:

TEOREMA 7.2 - Sejam S^* uma "schedule" ótima para um problema do tipo $n/m/F/r_j=0/C_{max}$ $(m > 2)$ e S a "schedule" gerada pelo algorítmo particionamento-heurística para este problema. Então $FT(S)FT(S^*)$ $|^m/2|$:

Prova: Seja $FT(S_i)$ $(i=1,\ldots, \lceil m/2 \rceil)$ o comprimento de cada "schedule" S_i obtida pelo algorítmo particionamento-heurística. Então $FT(S^*) \geq \max \{ FT(S_i)\}$ pois cada "schedule" S_i é obtida para o sub-problema. Por outro lado,

$FT(S) \leq \sum_{i=1}^{\lceil m/2 \rceil} FT(S_i) \leq \lceil m/2 \rceil \max \{FT(S_i)\}$. Assim $FT(S)/FT(S^*) \leq$

$\leq \lceil m/2 \rceil$.

---

1 - $\lceil \frac{m}{2} \rceil$ representa o menor inteiro maior ou igual a $\frac{m}{2}$

2 - Algorítmo desenvolvido na unidade 2.2

Para ilustrar este teorema e para mostrar que este limite é o menor possível, o seguinte exemplo será discutido:

Exemplo 7.2 - Considerando-se o problema "Flow-Shop" de m máquinas (m ímpar), n jobs e tempos de processamento dados por:

$$t_{j,2r-1} = \varepsilon_{2r-1}, \quad t_{j,2r} = k \quad t_{j,m}=0, \quad j=1,\ldots,n-1 \quad r=1,\ldots,\frac{m-1}{2}$$

$$t_{ni} = \varepsilon_{m-1}, \quad t_{nm}=(n-1)k \quad i=1,\ldots,m-1 \quad \text{onde}$$

$$\varepsilon_{2j-1} < \varepsilon_{2j+1} < \varepsilon_{m-1} << k \qquad i \leq j \leq \frac{m-3}{2}$$

Após aplicar-se o algorítmo particionamento-heurística para este exemplo, tem-se como resultado a "schedule" dada na figura 7.2(a). A "schedule" ótima para este problema é dada na figura 7.2(b)

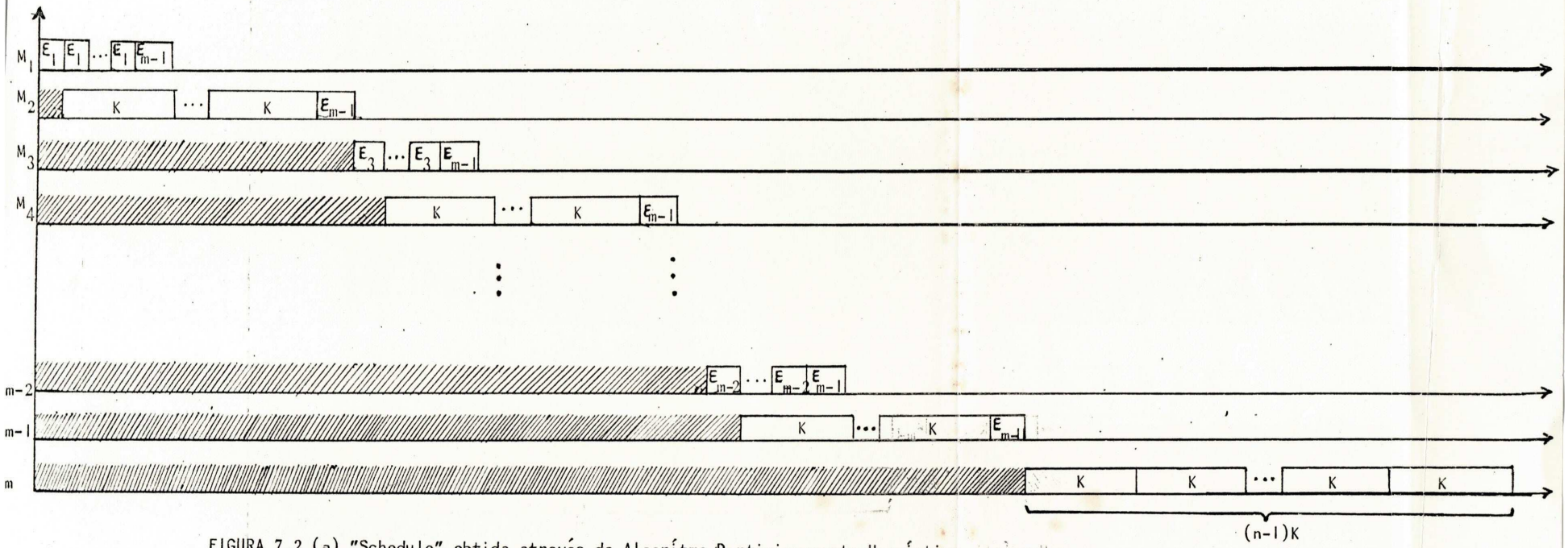


FIGURA 7.2 (a) "Schedule" obtida através do Algorítmo Particionamento-Heurística.

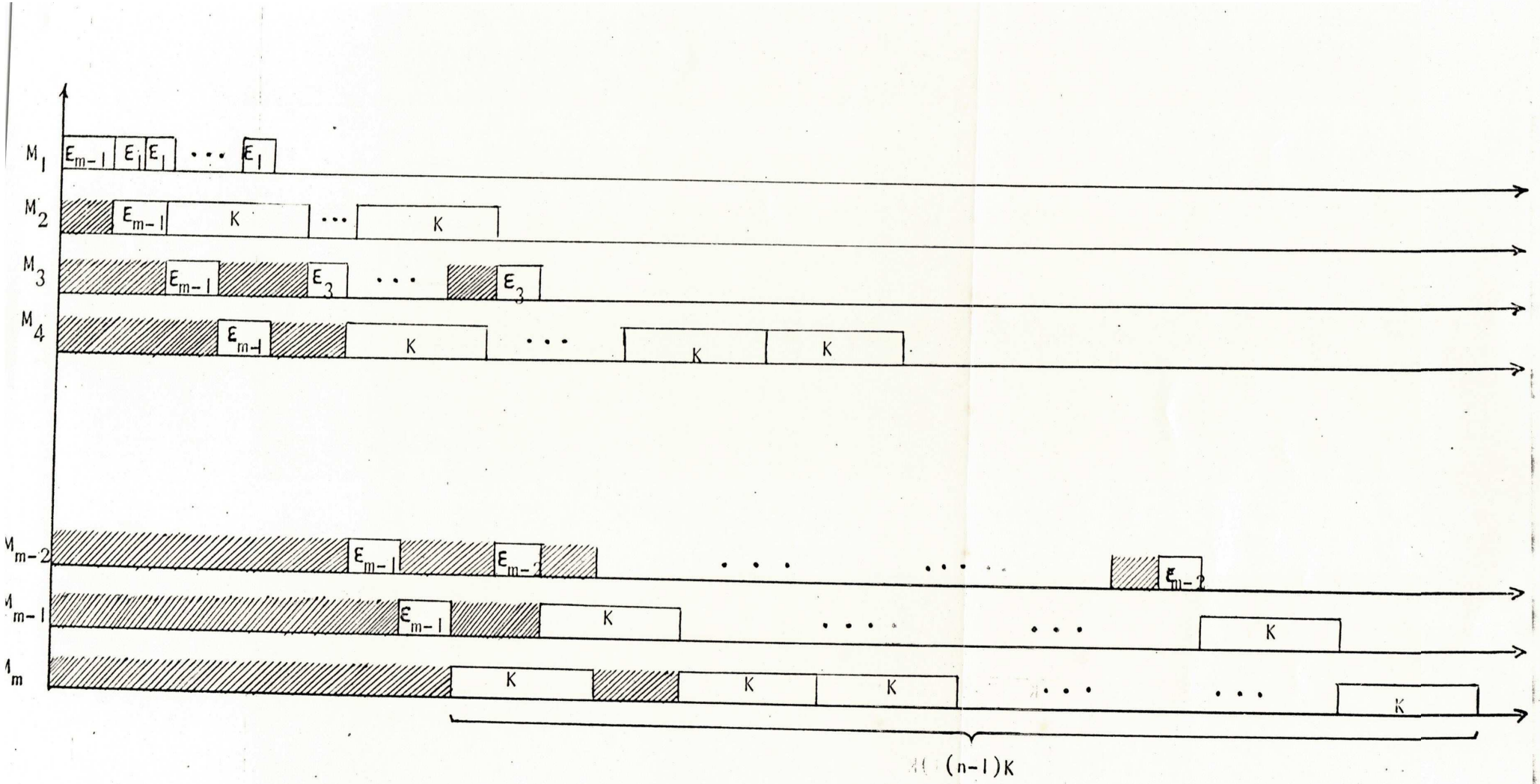


FIGURA 7.2 (b) "Schedule" ótima S^*

O tempo de conclusão da "Schedule S é:

$$FT(S) = \left\lceil \frac{m-2}{2} \right\rceil \left((n-1)\,k + \varepsilon_{m-1} \right) + \sum_{j=1}^{\frac{m-1}{2}} \varepsilon_{2j-1} + (n-1)\,k$$

e da "Schedule" S^* é:

$$FT(S^*) = (m-1)\,\varepsilon_{m-1} + (n-1)\,k + \sum_{r=2}^{\frac{m-1}{2}} \varepsilon_{2r-1}$$

$$\frac{FT(S)}{FT(S^*)} \cdot \lim_{\varepsilon_{m-1} \to 0} \frac{\left\lceil \frac{m-2}{2} \right\rceil \left((n-1)k + \varepsilon_{m-1} \right) + \sum_{j=1}^{\frac{m-1}{2}} \varepsilon_{2j-1} + (n-1)\,k}{(m-1)\,\varepsilon_{m-1} + (n-1)\,k + \sum_{r=2}^{(m-1)/2} \varepsilon_{2r-1}} =$$

$$= \frac{\left\lceil \frac{m-2}{2} \right\rceil (n-1)\,k + (n-1)\,k}{(n-1)k} = \left\lceil \frac{m-2}{2} \right\rceil + 1 = \left\lceil \frac{m}{2} \right\rceil .$$ Como $\varepsilon_{m-1} \to 0$,

então todos os valores ε_t tenderão a 0 (zero) visto que ε_{m-1} é o maior deles.

O mesmo raciocínio é utilizado quando m for um número par.

7.2 - ANÁLISE DO PIOR CASO ATRAVÉS DE HEURÍSTICAS PARA PROBLEMAS COM OBJETIVO C_j.

TEOREMA 7.3 - Sejam S^* uma "Schedule" ótima para um problema do tipo $n/m/F/r_j=0/\sum_{j=1}^{n} C_j$ e S uma "busy-schedule" qualquer para este problema.

Então $MFT(S) / MFT(S^*) \leq n$.

Prova: Para qualquer "busy schedule" S sabe-se que $C_j \leq R$ $j=1,\ldots,n$ onde $R = \sum_{j=1}^{n} R_j$ e $R_j = \sum_{i=1}^{m} t_{ji}$, e portanto $MFT(S) \leq R$.

Por outro lado, para a "schedule" ótima S^*, tem-se $R_j(S^*) \leq C_j(S^*)$ e assim

$$MFT(S^*) = \frac{1}{n} \sum C_j(S^*) \geq \frac{1}{n} \sum_{j=1}^{n} R_j = \frac{R}{n} .$$

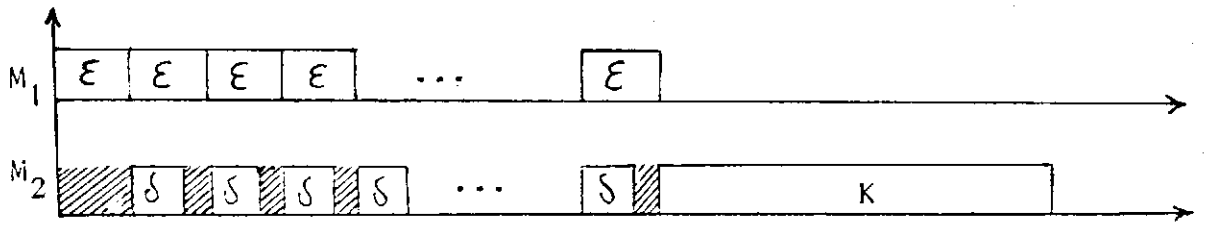
Logo $MFT(S)/MFT(S^*) \leq n$.

Para ilustrar este teorema e mostrar que este limite é o menor possível para este algorítmo, o exemplo seguinte será discutido.

Exemplo 7.3 - Considerar o problema "Flow-Shop" de 2 máquinas, n jobs e tempos de processamentos dados por $t_{j1}=\varepsilon$, $1 \leq j \leq n$ $t_{12}=k$ e $t_{j2}=\delta$ $2 \leq j \leq n$ e $\delta < \varepsilon << \frac{k}{n^2}$.

Em se tratando da função objetiva $\sum C_j$, a "schedule" ótima S^* para este problema é a "schedule" da figura 7.3(b). Por outro lado, quando aplicada a heurística "busy schedule", encontra-se uma "schedule" S mostrada na figura 7.3(a).

(a) - "Schedule" S para o exemplo 7.3 com objetivo $\sum C_j$.



(b) - "Schedule" ótima S^* para o exemplo 7.3 com objetivo $\sum C_j$.

FIGURA 7.3

A média dos tempos de conclusão dos jobs na "schedule" S é dada por

$$MFT(S)= \frac{1}{n} [(\varepsilon+k) + (\varepsilon+k+\delta) + (\varepsilon+k+2\delta) + \ldots + (\varepsilon+k + (n-1)\delta]=$$

$$= (k+\varepsilon) + (n-1)\,\delta/2 \quad \text{e para a "schedule" ótima } S^* \text{ é}$$

$$MFT(S^*) = \frac{1}{n} [(\varepsilon+\delta) + (2\,\varepsilon+\delta) + \ldots + ((n-1)\,\varepsilon +\delta) + (n\,\varepsilon+k)] =$$

$$= (n+1)\varepsilon/2+ \delta - \delta/n + k/n$$

Assim $\dfrac{MFT(S)}{MFT(S^*)} \leq \lim_{\varepsilon \to 0} = \dfrac{k + \varepsilon + (n-1)\,\delta /2}{(n+1)\,\varepsilon/2 + \delta - \delta/n + k/n} = n$

Como $\varepsilon \to 0$ então $\delta \to 0$ pois $\varepsilon > \delta$.

Vale observar que a heurística utilizada na determinação da "schedule" S^* do exemplo 7.3 foi processar os jobs em ordem não crescente de seus tempos de processamento $\sum_{i=1}^{2} t_{ji}$. Esta heurística é conhecida por SPT-"schedule".

TEOREMA 7.4 - Sejam S^* uma "schedule" ótima para um problema do tipo $n/m/F/r_j=0/ \sum C_j$ e S um SPT-"Schedule" para este problema. Então

$$\frac{MFT(S)}{MFT(S^*)} \leq \min \{m,n\} .$$

PROVA:

i) De acordo com o teorema 7.3,

$MFT(S)/MFT(S^*) \leq n.$

ii) Sem perda de generalidade, supor que $R_1 \leq R_2 \leq \ldots \leq R_n$

então

$$C_i(S) \leq \sum_{j=1}^{i} R_j \text{ e portanto } MFT(S) = \frac{1}{n} \sum_{i=1}^{n} C_i(S) \leq \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{i} R_j .$$

Por outro lado, seja $(J_{j_i}, \ldots, J_{j_n})$ a ordem dos jobs de acordo com seus tempos de conclusão da "schedule" ótima S^*. Assim,

$$C_{i_k}(S^*) \geq \frac{1}{m} \sum_{j=1}^{k} R_j \geq \frac{1}{m} \sum_{j=1}^{k} R_j \quad \text{por causa da ordem dos } R_j$$

$$\text{Logo } MFT(S^*) \geq \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{i} R_j \text{ e consequentemente}$$

$$MFT(S)/MFT(S^*) \leq m.$$

Para ilustrar este teorema e mostrar que este limite é o menor possível para este algorítmo, o exemplo seguinte será discutido:

Exemplo 7.4 - Considerar o problema "Flow-Shop" de m máquinas mn jobs e tempos de processamento dados por:

$t_{j1} = k \quad t_{j2} = t_{j3} = .. = t_{jm} = \varepsilon_1 , \quad j=1,...,n$

$t_{j1} = \varepsilon_2 , \quad t_{j2} = k \quad t_{j3} = \cdots = t_{jm} = \varepsilon_2 . \quad j=n+1,...,2n$

$t_{j1} = t_{j2} = \varepsilon_3 \quad t_{j3} = k \quad t_{j4} = ... = t_{jm} = \varepsilon_3 \quad j=2n+1,...3n$

$t_{j1} = t_{j2} = \quad . \; . \; . \quad = t_{jm-1} = \varepsilon_m , \; t_{jm} = k , \quad j=(m-1)n+1,...,mn$

onde $\varepsilon_j < \varepsilon_{j+1} \qquad 1 \leq j \leq (m-1) \quad \varepsilon_m << \frac{k}{n^2}$.

De acordo com o algorítmo SPT, a "schedule" S será a "schedule" da figura 7.4(a). Por outro lado, a "schedule" ótima S^* para este problema é a "schedule" da figura 7.4 (b).

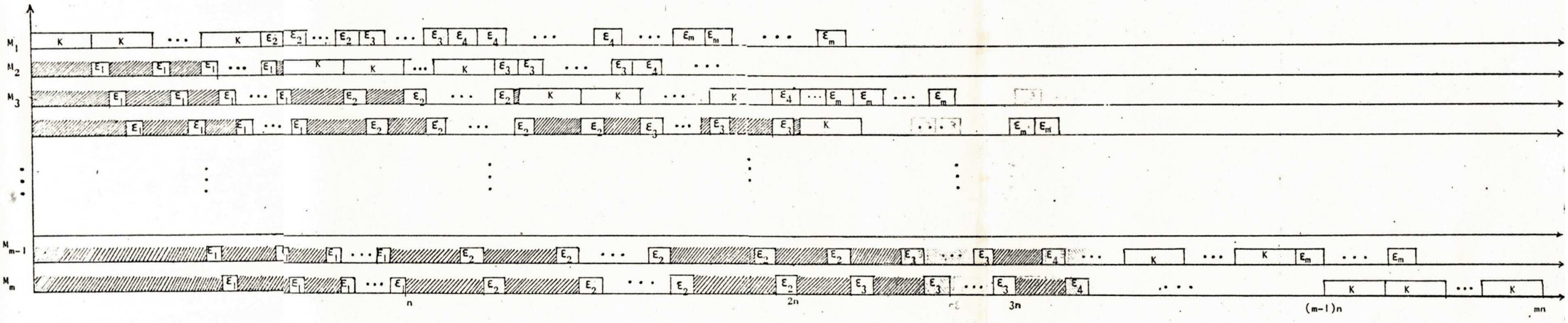


FIGURA 7.4 (a) "Schedule S para o exemplo 7.4 com objetivo ΣC_j

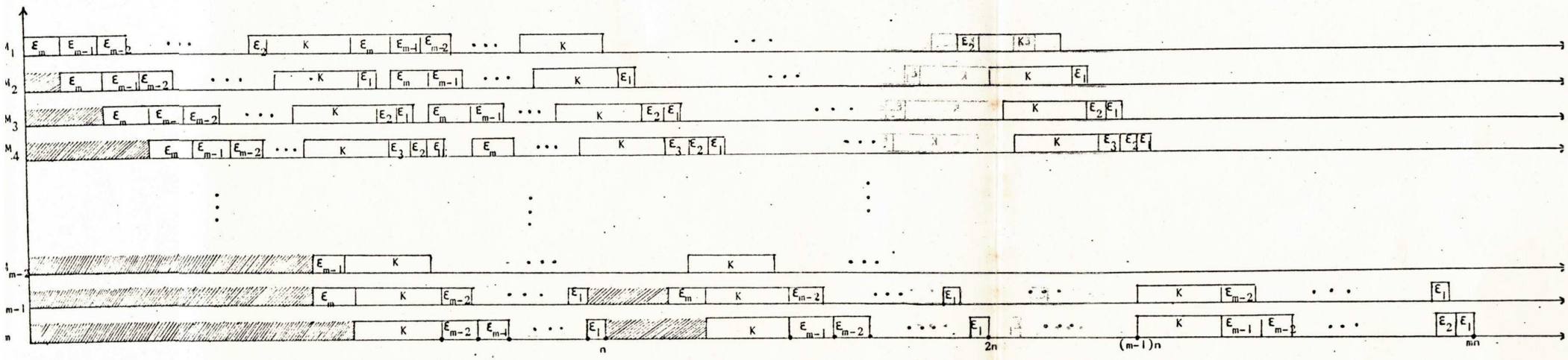


FIGURA 7.4 (b) "Schedule" ótima S^* para o exemplo 7.4 com objetivo $\sum C_j$

A média entre os tempos de conclusão dos jobs da "schedule" S (Figura 7.4(a)) é:

$$MFT(S) = \frac{1}{mn}\left[((m-1)\varepsilon_1+k) + ((m-1)\varepsilon_1+2k+\ldots+((m-1)\varepsilon_1+nk)+\right.$$

$$+ ((m-2)\varepsilon_2+\varepsilon_2 + (n+1)k) + (m-2)\varepsilon_2+\varepsilon_2 + (n+2)k)+\ldots. +$$

$$+ ((m-2)\varepsilon_2+\varepsilon_2 + 2nk) + ((m-3)\varepsilon_3 + \varepsilon_2 + \varepsilon_3 + (2_{n+1})k+\ldots+$$

$$+ ((m-3)\varepsilon_3+\varepsilon_2+\varepsilon_3+3nk) + \ldots + ((m-4)\varepsilon_4+\varepsilon_2+\varepsilon_3+\varepsilon_4 +$$

$$+ (3n+1)k)+ \ldots+ (\varepsilon_2+ \ldots+\varepsilon_{m-1} + (m-1)nk + \ldots.+$$

$$\left.+ (\varepsilon_2 + \varepsilon_3 + \ldots + \varepsilon_m + m\ n\ k)\right] =$$

$$= \frac{1}{mn}\left[(n(m-1)\varepsilon_1 + k + 2k + \ldots+nk) + (n(m-2)\varepsilon_2 + n\varepsilon_2 +\right.$$

$$+ (n+1)k + \ldots+ 2nk) + n(m-3)\varepsilon_3 + n(\varepsilon_2 + \varepsilon_3) +$$

$$+ (2n+1)k + \ldots + 3nk) + n(m-4)\varepsilon_4 + n(\varepsilon_2+\varepsilon_3+\varepsilon_4) +$$

$$+ (3n+1)k + \ldots + 4nk + \ldots + n(\varepsilon_2+\varepsilon_3 + \ldots + \varepsilon_m) +$$

$$\left.+((m-1)n + 1)k + \ldots\ldots\ldots + mnk\right) =$$

$$= \frac{1}{mn} [\, k + 2k + .. + nk + (n+1)\, k + .. + 2nk + .. + mnk + n/m-1)\varepsilon_1$$

$$+ n(m-2)\varepsilon_2 + n\varepsilon_2 + n\,(m-3)\varepsilon_3 + n(\varepsilon_2+\varepsilon_3) + n(m-4)\,\varepsilon_4 +$$

$$+ n(\varepsilon_2+ \varepsilon_3+\varepsilon_4) + .. + n(\varepsilon_2+\varepsilon_3 + ... + \varepsilon_m)\,] =$$

$$= \frac{1}{mn} \left[\frac{(1 + .mn)\, mnk}{2} + n \left(\sum_{j=1}^{m-1} (m-j)\, \varepsilon_j + \sum_{j=1}^{m-2} (j-1)\, \varepsilon_{m-j} + \varepsilon_m \right) \right] =$$

$$= \boxed{\frac{(1 + mn)\, k}{2} + \frac{1}{m} \sum_{i=1}^{m} \times_j \varepsilon_j \;)}$$

Para os jobs da "schedule" ótima S^* (Fig. 7.4 (b)), tem-se

$$MFT(S^*) = \frac{1}{mn} [\, ((m-1)\,\varepsilon_m + k) + (m-1)\,\varepsilon_m + k + \varepsilon_{m-1}) +$$

$$+((m-1)\varepsilon_m +k+\varepsilon_{m-1}+ \varepsilon_{m-2}) + ... + (m-1)\,\varepsilon_m+k + \sum_{j=1}^{m-1} \varepsilon_j) +$$

$$+ ((m-1)\,\varepsilon_m + k + \sum_{j=2}^{m} \varepsilon_j + k) + ((m-1)\varepsilon_m+k+ \sum_{j=2}^{m} \varepsilon_j+k+\varepsilon_{m-1})+...+$$

$$+ ... + ((m-1)\,\varepsilon_m + k + \sum_{j=2}^{m} \varepsilon_j + k + \sum_{j=1}^{m-1} \varepsilon_j)\,] + ((m-1)\,\varepsilon_m +$$

$$+ k + 2 \left(\sum_{j=2}^{m} \varepsilon_j + k \right)) + \ldots + \left((n-1)\, \varepsilon_m + k + 2 \left(\sum_{j=2}^{m} \varepsilon_j + k \right) \right.$$

$$+ \sum_{j=1}^{m-1} \varepsilon_j + \left((m-1)\, \varepsilon_m + k + 3 \left(\sum_{j=2}^{m} \varepsilon_j + k \right) \right) + \left((m-1)\, \varepsilon_m + k + \right.$$

$$+ 3 \left(\sum_{j=2}^{m} \varepsilon_j + k \right) + \varepsilon_{m-1}) + \ldots + \left((m-1)\, \varepsilon_m + k + 3 \left(\sum_{j=2}^{m} \varepsilon_j + k + \right. \right.$$

$$+ \sum_{j=1}^{m} \varepsilon_j + \ldots + \left((m-1)\, \varepsilon_m + k + (n-1) \left(\sum_{j=2}^{m} \varepsilon_j + k \right) \right) + \ldots +$$

$$+ \left((m-1)\, \varepsilon_m + k + (n-1) \left(\sum_{j=2}^{m} \varepsilon_j + k \right) + \sum_{j=1}^{m} \varepsilon_j \right] =$$

$$= \frac{1}{mn} \left[m\, n \left((m-1)\, \varepsilon_m + k \right) + m\, k + 2\, m\, k + \ldots + (n-1) m\, k + \right.$$

$$\left. + Y_j\, \varepsilon_j \right| \quad j = 1, \ldots m =$$

$$\frac{1}{mn} \left[mk + 2\, mk + \ldots + (n-1)\, mk + mnk + Z_j\, \varepsilon_j \right] =$$

$$= \boxed{\frac{(1+n)\, k}{2} + \frac{Z_j\ \ \varepsilon_j}{mn}}$$

Assim,

$$\frac{MFT(S)}{MFT(S^*)} \quad \lim_{\substack{n\to\infty \\ \varepsilon_j \to 0}} \quad \frac{\frac{k}{2}(1+mn) + \frac{1}{m} x_j \varepsilon_j}{\frac{k}{2}(1+n) + \frac{1}{mn} z_j \varepsilon_j} =$$

$$\lim_{n\to\infty} \frac{1+mn}{1+n} = m$$

Aqui, encontrou-se como limite o valor m e no exemplo 7.3, o valor n, logo a solução final é $\min\{m, n\}$.

C O N C L U S Ã O

Neste trabalho foi feito um apanhado geral dos diversos métodos de solução para problemas "Flow-Shop" e "Job-Shop". Conforme foi visto durante todo o contexto, estes problemas têm despertado grande interesse entre os pesquisadores operacionais, devido à sua aplicabilidade no processo industrial e também a grande quantidade de problemas em aberto.

Quanto ao capítulo II :

O único algorítmo exato capaz de resolver problemas "Flow-Shop" em tempo polinomial, foi escrito por Johnson e resolve todos os problemas do tipo $n/2/F/r_j=0/C_{max}$. Sua otimalidade foi demonstrada neste trabalho.

Alguns problemas "Flow-Shop" muito particulares de 3 máquinas com função objetiva C_{max}, podem ser resolvidos também em tempo polinomial, usando o algorítmo de Johnson mediante algumas modificações.

Quanto ao capítulo III:

Os problemas "Job-Shop" do tipo $n/2/J/n_j \leq 2/C_{max}$ e $n/2/J/t_{jki}=1/C_{max}$ podem ser resolvidos por algorítmos exatos em tempo polinomial. Melhorou-se de maneira considerável a demonstração da otimalidade deste algorítmos.

Existem outros alogrítmos exatos e polinomiais para problemas "job-Shop", como por exemplo o algorítmo recentemente escrito por P. Brucker[1] para o problema do tipo $n/2/J/t_{jki}=1/L_{max}$. A idéia é a mesma do algorítmo para o problema $n/2/J/t_{jki}=1/C_{max}$ onde as operações O_{jk}' com rótulos dados por $(O_{jk})=d_j-n_j+k$ devem ser processados de maneira não decrescente.

Quanto ao capítulo IV:

Conclui-se que os algorítmos "Branch and Bound" para problemas "Flow-Shop" e "job-Shop" apesar de resolverem, de maneira exata, problemas deste tipo, são muitas vezes desaconselháveis ou mesmo intratáveis, devido ao seu crescimento exponencial com os dados do problema.

Neste trabalho, o algorítmo "Branch and Bound" para problemas "Flow-Shop" além de ter sido generalizado, foi escrito utilizando-se'

---

1 - Este algorítmo ainda não foi publicado.

uma lista, tornando deste modo, fácil de entender e implementar. O algorítmo "Branch and Bound" para problemas "Job-Shop", foi escrito a partir do algorítmo de enumeração total. Aqui também usou-se uma lista para melhor entendimento.

Quanto ao capítulo V:

Este capítulo trata da teoria NP-Completa assunto de maior polêmica entre os pesquisadores operacionais da atualidade.

A partir desta teoria tem-se argumentos contra a possibilidade de resolver problemas NP-Completos usando algorítmos polinomiais em um computador comum.

Fundamentais diferenças foram dadas às classes P e NP. Também foram dados vários conceitos e demonstrações tornando-se possível verificar quando um problema é NP-Completo, conforme foi feito com vários problemas "Flow-Shop" e "Job-Shop".

Este capítulo está inteiramente ligado à ordem de complexidade de algorítmos.

Quanto ao capítulo VI :

Os algorítmos heurísticos para problemas "Flow-Shop" e "Job-Shop" são algorítmos polinomiais e aplicáveis para qualquer tipo de problema . Entretanto, eles apresentam certas desvantagens, como por exemplo, a incerteza quanto à qualidade da solução. Quantos aos métodos heurísticos para problemas "flow-Shop" desenvolvidos neste trabalho, Campbell, Dudek e Smith fizeram um estudo comparativo de seus resultados constatando, na maioria dos casos, maior eficiência no algorítmo escrito por eles, também chamado algorítmos CDS.

Melhorou-se a linguagem deste algorítmo a partir do algorítmo de Johnson[2].

Quanto ao capítulo VII :

A partir de certas heurísticas, é possível determinar a qualidade da solução de um problema determinando um limite superior para a razão entre o valor da função objetiva de uma "schedule" obtida para' um processo heurístico qualquer e o valor da função objetiva da "schedule" ótima do problema.

Aqui foram discutidos somente problemas "Flow-Shop" e as funções objetivas C_{max} e $\sum C_j$ com resultados muito simples. Desenvolveu-se exemplos genéricos para mostrar que estes limites, para estes algorítmos, são os menores possíveis. Na bibliografia só tinha-se exemplos com m=2.

É provável que seja possível construir outras rotinas que tornem estes limites melhores como também trabalhar com outras funções ' objetivas e com os problemas "Job-Shop".

---

2 - Alogorítmo desenvolvido na unidade 2.2

REFERÊNCIAS BIBLIOGRÁFICAS

1) AHO, A.V., HOPCROFT, J.E., e ULLMAN, J.D., The Design and Analysis of Computer Algorithms, Addison - Wesley, California, 1974.

2) BAKER, K.R., Introduction to Sequencing and Scheduling, New York, 1974.

3) BRUCKER, P. "NP-Complete Operations Research Problems and Aproximation Algorithms", Zeitschrift fur Operation Reserch, Band 23, 1979.

4) BRUCKER, P. Scheduling Akaderische Verlagsans talt, Wiesbaden, que brevemente será publicado.

5) CONWAY, R.W., MAXWELL. W.L., e MILLER, L.W., Theory of Scheduling, Addison-Wesley, Reading, Mass 1967.

6) COOK, S.A., The Complexity of theorem - proving procedures. Proceedings of the Third Annual ACM Symposium of Theory of Computing , 1971.

7) GAREY, M., JOHNSON, D., e SETHI, R., "The Complexity of Flow-Shop and Job-Shop Scheduling" Mathematics Operations Research, V.1, 1976.

8) GIFFLER, B. e THOMPSON, G.L., "Algorithms for Solving Production'

Scheduling Problems", Operations Research, 8, 1966.

9) GONZALEZ, T. e SAHNI, S., "Flow-Shop and Job-Shop Schedules: Complexity and Aproximation", Operations Research, 1978.

10) GONZALEZ, T. e SAHNI, S., "Open Shop scheduling to minimize Finish Time", J. ACM, 23, 1976.

11) HEFETZ, N. e ADIRI, I., "An efficient Optimal Algorithm for the Two-Machines, Unit-time, Job-Shop Schedule-length, Problem," Discussion paper, mimeograph Series nº 237, Operations Research, Statistics and Economica, TECHNION, HAIFA, 1979.

12) JACKSON, J.R., "Notes on Some Scheduling Problems", Management Sciences Research Project Research Report nº 35, University of California, Los Angeles, 1954 (mimeographed).

13) JACKSON, J.R., An Extension of Johnson's Results on Job Lot Scheduling", Naval Research Logistics Quarterly, V.3., 1956.

14) JOHNSON, M.S., "Optimal Two-and Three-Stage Production Schedules' With Setup Times Included", Naval Research Logistics Quartily, V.1 1954.

15) KARP, R.M., Reducibility Among Combinatorial Problems. Complexity of Computer Computations, New York, 1972.

16) LAGEWEG, S.J. LENSTRA, J.K. e RINNOOY KAN, A.H.G., Job-Shop Scheduling by Implicit Enumeration, 1976.

17) LAGEWEG, S.J. LENSTRA, J.K. e RINNOOY KAN, A.H.G., "A General Bounding Scheme for the Permutation Flow-Shop Problem", Operation Research, Vol 26, 1978.

18) LENSTRA, J.K.e RINNOOY KAN, A.H.G., e BRUCKER, P., "Complexity of Machine Scheduling Problems", Annals of Discrete Mathematics, V.1, 1977.

19) LENSTRA, J.K., e RINNOOY KAN, A.H.G., "Computational Complexity of Discrete Otimization Problems". Econometric Institute Report, 1978.

20) SZWARC, W., Solution of the Akers-Friedman Scheduling Problem, Operations Research, 1960.