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

PROCEDIMENTOS HEURÍSTICOS EM SCHEDULING DE TAREFAS INDEPENDENTES E ANÁLISE DE SEUS PIORES CASOS

REGINA CÉLIA BALBINO FIGUEIRA LISBOA

CAMPINA GRANDE - PARAÍBA
MAIO - 1980

L769p Lisboa, Regina Celia Balbino Figueira
Procedimentos heuristicos em scheduling de tarefas independentes e analise de seus piores casos / Regina Celia Balbino Figueira Lisboa. - Campina Grande, 1980.
116 f. : il.

Dissertacao (Mestrado em Ciencias) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Metodos Heuristicos 2. Scheduling - 3. Dissertacao I. Brucker, Peter Joachim Siegfried, Dr. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.023(043)

PROCEDIMENTOS HEURÍSTICOS EM SCHEDULING DE TAREFAS INDEPENDENTES E ANÁLISE DE SEUS PIORES CASOS

REGINA CÉLIA BALBINO FIGUEIRA LISBOA

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO E PESQUISA DO CENTRO DE CIÊNCIAS E TECNOLOGIA DA UNIVERSIDADE FEDERAL DA PARAÍBA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS.

A P R O V A D A

BANCA EXAMINADORA:

Prof PETER JOACHIM SIEGFRIED BRUCKER - Ph.D.
- Presidente -

Prof ULRICH SEIP - Ph.D.

Prof ULRICH SCHIEL - Ms.C.

Ao Victor, Patrícia e Renata

AGRADECIMENTOS

Agradeço sinceramente

à Universidade Federal do Mato Grosso

pela oportunidade dada para a realização deste trabalho;

ao Professor Peter Brucker

pela valiosa orientação;

ao Professor Ulrich Seip

pela sua contribuição;

a meus pais e irmãos

pelo incentivo, apesar da nossa separação.

RESUMO

Neste trabalho, serão estudados vários procedimentos heurísticos para scheduling de tarefas independentes.

Na primeira parte, identifica-se o problema de scheduling de tarefas independentes dentro da teoria de scheduling. A complexidade dos resultados é também descrita, a fim de justificar uma abordagem heurística para a maioria dos problemas.

A segunda parte descreve vários procedimentos heurísticos para o caso de único processador e processadores paralelos, que são idênticos ou não idênticos. Também são dadas implementações desses algoritmos. Finalmente, é descrito o comportamento do pior caso para processadores paralelos através de limites para a razão entre os tempos de conclusão, que correspondem as soluções heurística e ótima. São dadas demonstrações para a validade destes limites.

ABSTRACT

In this work we study various heuristic procedures for scheduling independent jobs and analize their worst-case behaviour. In the first part we identify scheduling independent jobs within the theory of scheduling. Also, complexity results described justifying heuristic approaches for most of the problems.

The second part describes various heuristics for cases with one processor, and parallel processors which are identical or nonidentical. Also, implementations of these algorithms are given. Finally, the worst-case behaviour for cases of parallel processors is described by bounds for the ratio between makespans corresponding with heuristic and optimal solutions. Proofs for the validity and the tightness of these bounds are given.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

Scheduling [1] pode ser definido como alocação de recursos sobre o tempo para executar uma coleção de tarefas. Mais informalmente, o termo scheduling pode significar um método racional para se chegar a decisões. Porém, se for olhado através de uma abordagem sistêmica, encontraremos no mesmo uma es

---

1. O termo foi usado em inglês porque não existe uma palavra correspondente em Português.

trutura formal que tem encontrado mais e mais suporte na própria prática administrativa contemporânea, visto que, na área da pesquisa operacional, é grande o interesse e, consequentemente, o aprimoramento de técnicas procurando, se não otimizar, pelo menos encontrar soluções satisfatórias para problemas onde scheduling seja fundamental.

Scheduling só se torna relevante em situações onde a natureza das tarefas a serem programadas já tenha sido determinada. Isto só é possível depois que se tenham respostas para as perguntas: Que produtos ou serviços deverão ser fornecidos? Em que escala? Que recursos estarão disponíveis? Respostas a estas perguntas devem ser oferecidas pelo planejador, as quais, logicamente, precedem a fase de scheduling.

Quando na fase de scheduling, isto é, para o problema de scheduling propriamente dito, a solução é basicamente encontrar respostas para as questões:

1. Para a execução de cada tarefa, quais recursos serão alocados?

2. Quando será executada cada tarefa?

Através dessas duas respostas pode-se perceber que a essência desses problemas dá lugar a decisões de alocação e sequenciação. Pode-se perceber, ainda, que os elementos vitais em problemas de scheduling são recursos e tarefas. Enquanto re

cursos são tipicamente caracterizados em termos de suas capacidades quantitativas e de seus tipos, as tarefas são individualmente descritas em termos de informações relativas à quantidade de recursos requerida por elas, sua duração, o tempo em que elas podem ser inicializadas e o tempo previsto para a finalização das mesmas. Além do mais, um conjunto de tarefas é descrito em termos das restrições tecnológicas existentes, ou não, entre seus elementos.

Neste trabalho, os recursos denotam-se processadores que estão disponíveis em uma ou várias unidades. As tarefas não têm restrições tecnológicas ou de precedência, o que vale dizer que os problemas apresentados são de scheduling de tarefas independentes.

A identificação desses problemas é feita no Capítulo II onde aparecem explicitamente as funções obtjetivas mais usuais ao lado de definições importantes relativas às tarefas. Nesta parte do trabalho são caracterizados, quanto à disponibilidade de recursos, os problemas de scheduling abordados em todo o trabalho: em um único-processador e em mais de um processador idênticos ou não idênticos, disponíveis em paralelo.

No Capítulo III procura-se mostrar que os problemas de scheduling, afora os de tarefas independentes em um único processador, são na grande maioria bastante complexos. Abordando sobre complexidade dos problemas e descrevendo sobre as so

luções para os mesmos, procura-se justificar a importância dos procedimentos heurísticos, visto que estes problemas são quase sempre intratáveis por procedimentos que venham produzir soluções exatas. Procura-se justificar, ainda nesta parte, o porquê da análise do pior caso, ou seja, ao se determinar soluções heurísticas ou aproximadas para um problema, é importante observar até que ponto estas soluções são "satisfatórias".

O probleme da scheduling em um único processador é tratado no Capítulo IV, onde o mesmo toma inclusive a conotação de problema de sequenciação pelas características do recurso disponível. Para o algoritmo de Wilkerson - Irwin, apresentado nesta parte, bem como para outros algoritmos heurísticos com o objetivo de minimizar Atraso, não é feita análise do pior caso. Isto, porque os estudos relativos à análise do pior caso para avaliar o desempenho de um algoritmo, aplicado a problemas de scheduling envolvendo objetivos que não sejam minimização do tempo de conclusão, não mostram resultados satisfatórios.

O Capítulo V aborda sobre procedimentos heurísticos, aplicados a problemas de scheduling em processadores paralelos. Para todos os algoritmos é feita análise do pior caso, visto que os problemas ali apresentados têm como objetivo minimizar Tempo de conclusão.

Finalmente, no Capítulo VI, são apresentadas algumas

considerações finais.

CAPÍTULO II

IDENTIFICAÇÃO DO PROBLEMA

Em problemas de scheduling normalmente existem restrições de precedência [1] entre as tarefas, consideradas estruturas especiais

Problemas de scheduling onde as tarefas aparecem independentes tecnologicamente uma da outra, também são considera

---

1. Ver (1, 2) onde são estudadas estruturas especiais como flow-shop, job-shop, árvore.

dos como uma estrutura especial. Em tais problemas é necessário caracterizar a configuração dos recursos: Pode-se ter somente um tipo de recursos disponível em uma quantidade unitária ou em m quantidades simultaneamente. Neste último caso diz-se que os recursos estão disponíveis em paralelo. Pode-se ter ainda m tipos de recursos diferentes, todos disponíveis em uma quantidade unitária simultaneamente, chamados recursos não idênticos disponíveis em paralelo.

Recurso pode ser interpretado de várias maneiras, dependendo da natureza do problema. Será considerado, a partir deste ponto, recurso como processador, pois o termo é mais específico e melhor se ajusta aos modelos que serão tratados aqui:

Como foi visto na introdução, uma tarefa individual é descrita em termos de informações que a caracterizam, tais como:

a) Tempo de processamento - $\mu(J_i)$: A quantidade de tempo requerida pela tarefa J_i para o seu processamento.

b) Término previsto - $d(J_i)$: O ponto, no tempo, previsto para o término do processamento da tarefa J_i.

Estas informações são importantes para os três modelos que serão descritos a seguir:

2.1 - SCHEDULING DE TAREFAS INDEPENDENTES EM UM ÚNICO PROCESSA

DOR

O problema básico de único-processador é fundamental no estudo de scheduling. Ele é considerado o mais simples problema de scheduling, porque nele não existe distinção entre sequência e alocação de recursos. Neste caso uma ordenação de tarefas caracteriza uma schedule, por esta razão ele é chamado problema de pura sequenciação.

De um ponto de vista prático, aplicações diretas deste modelo estão cada vez mais frequentes e importantes do que se poderia inicialmente imaginar. Existem muitas situações em que um grande complexo de equipamentos comporta-se como se fosse uma única máquina. Na indústria química ou de processamento, uma total aparelhagem muitas vezes representa um processo integrado que opera sobre um único produto em um só tempo, se não, sobre muitos diferentes produtos distintamente em sequência. Como exemplo, pode-se citar a manufaturação de detergentes, em que muitos diferentes tipos de marcas são processados separadamente e em sequência no mesmo processador; a manufaturação de tintas, em que tintas de diferentes cores são processadas como grupos separados em uma mesma linha de equipamento.

Aqui, poderiam ser citadas várias outras aplicações, mas o que é mais importante ter em mente, em relação a problemas de único-processador, é que tais problemas facilitam também a formulação e compreensão de sistemas mais complexos. Pa

ra entender completamente o comportamento de um sistema complexo, é vital entender o funcionamento de seus componentes, e,quase sempre, o problema de único-processador aparece como um elementar componente em um grande problema de scheduling.

O problema de único-processador é caracterizado pelas seguintes condições:

a) O sistema é composto de somente um único processador que está continuamente disponível, nunca se mantendo ocioso, sendo que o mesmo só pode processar uma única tarefa de cada vez.

b) Um conjunto $T = \{J_1, J_2, \ldots, J_n\}$ de tarefas independentes todas disponíveis, para processamento, no tempo zero com um tempo de processamento igual a $\mu(J_i)$

c) Uma vez iniciado o processamento de uma tarefa, ela é processada totalmente sem interrupção.

Sobre estas condições existe uma correspondência uma-a-uma entre uma sequência de n tarefas e a permutação dos índices 1, 2, ..., n das tarefas. O número total de distintas soluções para o problema de único-processador é, portanto, n! que é o número de diferentes permutações de n elementos.

Uma das maneiras de representação de um problema de scheduling é a utilização do diagrama de Gantt, onde o mesmo

transmite informações a respeito de um possível resultado de uma scheduling, isto é, no diagrama tem-se uma possível schedule, que neste caso mais específico pode ser chamada de sequência.

Basicamente, o diagrama de Gantt consiste de um gráfico de eixos cartesianos, onde recursos são mostrados ao longo do eixo vertical e uma escala de tempo é mostrada ao longo do eixo horizontal. No modelo de único-processador uma representação geral seria:

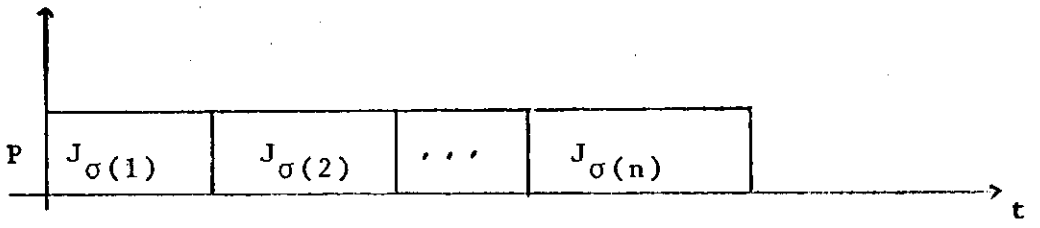


Fig. 2.1 - Schedule de um único-processador onde a permutação σ determina a sequência em que as tarefas serão executadas.

Funções Objetivas:

No plano ideal, a função objetiva consistiria de todos os custos envolvidos na scheduling, dependendo do tipo de decisão. Na prática, contudo, tais custos são difíceis para medir e também difíceis para serem completamente identificados. No entanto, três tipos de objetivos, na tomada de decisão, são os que mais prevalecem em scheduling: eficiente utilização de recursos, rápidas respostas para demandas e obediência, tanto quanto possível, aos prazos estipulados para conclusão das tare

fas.

Frequentemente, medidas de custo relativo como tempo ocioso de máquina, tempo de espera de uma tarefa, atraso de uma tarefa, podem ser usadas substituindo o custo total do sistema. A variedade dos diferentes critérios que tem sido empregada nos estudos teóricos de scheduling, em parte reflete a variedade das diferentes circunstâncias em que aparecem tais problemas e os diferentes custos e valores que são relevantes em cada caso. Contudo, não se deve deixar de salientar que a escolha do critério também tem sido influenciada pela esperança de obter uma solução.

Os dados fundamentais para serem usados na avaliação de schedules são:

Tempo de início - $s(J_i)$: O ponto, no tempo, no qual o processamento da tarefa J_i é iniciado.

Tempo de conclusão - $C(J_i)$: O ponto, no tempo, no qual o processamento da tarefa J_i é concluído. Isto é: $C(J_i) = s(J_i) + \mu(J_i)$.

Uma outra medida quantitativa importante, que é função do tempo de conclusão de uma tarefa, para avaliação de schedules é:

Desvio - $D(J_i)$: A quantidade de tempo pela qual, o tempo de conclusão da tarefa J_i difere do término previsto

$d(J_i)$ para o término da tarefa. $D(J_i) = C(J_i) - d(J_i)$.

É importante notar que a quantidade $D(J_i)$ pode assumir um valor negativo, quando uma tarefa é finalizada mais cedo. $D(J_i)$ negativo representa melhor serviço que o requisitado, enquanto $D(J_i)$ positivo representa serviço mais pobre que o requisitado. Em muitas situações, distintas penalidades ou custos podem ser associados com $D(J_i)$ positivo, enquanto que nenhum benefício será associado com $D(J_i)$ negativo. Portanto, muitas vezes torna-se útil trabalhar com uma quantidade que mede somente $D(J_i)$ positivo, que será denominada por:

Atraso - $A(J_i) = \max \{0, D(J_i)\}$

Também pode-se trabalhar com outra medida que indicará se a tarefa J_i atrasou ou não:

$$U(J_i) = \begin{cases} 0 & \text{se } C(J_i) \leq d(J_i) \\ 1 & \text{se } C(J_i) > d(J_i) \end{cases}$$

Schedules são geralmente avaliadas por quantidades agregadas que envolvem informações sobre todas as tarefas, resultando em uma medida de desempenho uni-dimensional.

Supondo que existem n tarefas para serem programadas, as principais funções objetivas que medem a qualidade de uma schedule, são:

a) Tempo de conclusão: $M = \max \{C(J_i)/1 \leqslant i \leqslant n\}$

É o tempo total gasto para o processamento de todas as tarefas, isto é, o tempo de término da schedule que é medido pelo maior $C(J_i)$. O objetivo é minimizar esse tempo de conclusão.

Podemos notar que Tempo de conclusão é um resultado trivial para modelos de único-processador, sendo seu valor uma constante para qualquer schedule das n tarefas. Portanto, só será considerado para os modelos subsequentes.

b) Fluxo de tempo ponderado: $F_\omega = \sum_{i=1}^{n} \omega(J_i)\ C(J_i)$

Aqui $\omega(J_i)$ representa um peso dado que está relacionado com a importância da tarefa J_i. O objetivo é encontrar uma schedule onde o somatório dos ponderados tempo de conclusão seja mínimo. Quando a importância de todas as tarefas é a mesma, tem-se que $\omega(J_i) = 1$ para todo i, e

$$F = \sum_{i=1}^{n} C(J_i)$$

c) Desvio máximo: $D_{max} = \max \{D(J_i)/1 \leqslant i \leqslant n\}$

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

d) Atraso ponderado: $A_{\omega} = \sum_{i=1}^{n} \omega(J_i)\ A(J_i)$

Aqui $\omega(J_i)$ representa custos de penalidades que são proporcionais aos possíveis atrasos. O objetivo é encontrar uma schedule onde a soma dos possíveis custos de penalidade seja mínima.

Quando a importância de todas as tarefas é a mesma, tem-se que $\omega(J_i) = 1$ para todo i, e

$$A = \sum_{i=1}^{n} A(J_i)$$

e) Número de atrasos das tarefas: $N_A = \sum_{i=1}^{n} U(J_i)$

Aqui, o objetivo é encontrar uma schedule onde o número de atrasos seja o mínimo possível.

2.2 - SCHEDULING DE TAREFAS INDEPENDENTES EM PROCESSADORES PARALELOS IDÊNTICOS:

Em problemas de scheduling, muitas vezes é possível aproveitar-se de paralelismo na estrutura de recursos.

Uma das aplicações de paralelismo seria em problemas de tarefas independentes processadas em um único estágio, como no problema anterior, com a diferença que aqui, ao invés de um único-processador, existiriam vários processadores idênticos disponíveis; donde conclui que este problema é uma extensão do

problema de único processador. Outro aspecto a considerar é que aqui pode ser apreciada a distinção entre alocação e sequenciação de recursos que são inerentes aos problemas de scheduling, o que não acontecia no problema de único-processador.

O problema pode ser caracterizado por:

a) O sistema é um conjunto de m processadores idênticos $\{P_1, \ldots, P_m\}$ que operam em paralelo e que estão continuamente disponíveis, nunca mantendo-se ociosos e sendo que os mesmos só podem processar uma única tarefa de cada vez.

b) O conjunto $T = \{J_1, \ldots, J_n\}$ de tarefas independentes, cada uma das quais tendo um tempo de execução $\mu(J_i)$ e requerendo somente um processador.

c) Uma vez iniciado o processamento de uma tarefa, ela é processada sem interrupção.

Uma possível schedule para T designa para cada $J_i \in T$ um processador $p(J_i)$, $1 \leq p(J_i) \leq m$ e um tempo de início de processamento $s(J_i) \geq 0$ tal que, se $p(J_i) = p(J_k)$, mas $i \neq k$, então os dois intervalos de execução $(s(J_i), c(J_i))$ e $(s(J_k), C(J_k))$ são disjuntos.

Também aqui, uma schedule pode ser representada através do diagrama de Gantt, como mostra a Figura 2.2 abaixo, so

mente que no eixo vertical existem vários processadores. Cada tarefa é representada por um retângulo cujo comprimento corresponde ao seu tempo de execução, e para cada processador existe uma linha consistindo das tarefas que ele executa. A região sombreada representa tempos durante os quais um processador está desocupado.

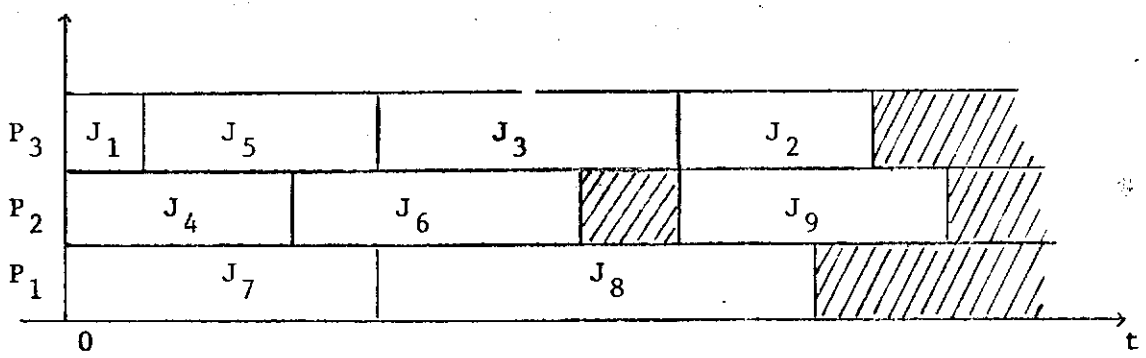


Fig. 2.2 - Schedule de vários processadores

Neste sistema, é possível proceder com as fundamentais medidas de desempenho de uma schedule que vimos no modelo de único-processador somente que, desta vez, as decisões refletirão paralelismo de recursos.

Foi visto que no modelo de único-processador Tempo de conclusão de uma schedule é um resultado trivial; ao contrário, para este modelo, é uma medida de desempenho muito importante e não trivial, apesar de ser mais simples para avaliar do que as outras medidas como F_ω, A_ω, D_{max}, etc.

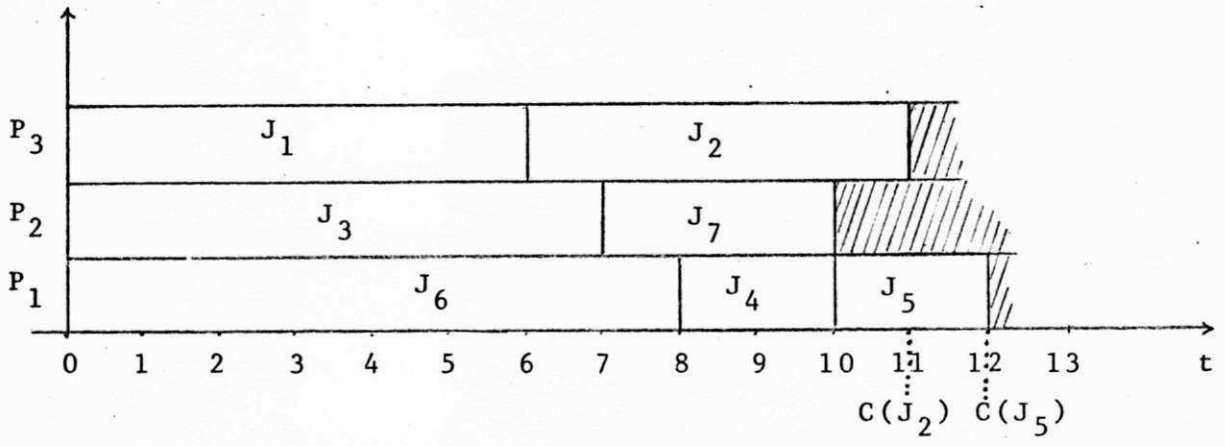
Serão descritos aqui alguns exemplos para ilustrar o uso de algumas dessas medidas para processadores paralelos idên

ticos.

Exemplo 2.1.

Considere o conjunto de tarefas abaixo, com os seus respectivos tempos de processamento, sendo que, para a execução das tarefas, 3 processadores estarão disponíveis, m = 3.

J_i	J_1	J_2	J_3	J_4	J_5	J_6	J_7
$\mu(J_i)$	6	5	7	2	2	8	3



A Figura 2.3 acima mostra uma possível schedule para o exemplo dado com:

$M = \max C(J_i) = 12$, que é o Tempo de conclusão da schedule,

$$F = \sum_{i=1}^{n} C(J_i) = 6 + 11 + 7 + 10 + 12 + 8 + 10 = 64$$

Exemplo 2.2.

Será introduzido aqui um exemplo com $d(J_i)$, onde $m = 2$

J_i	J_1	J_2	J_3	J_4	J_5	J_6
$\mu(J_i)$	2	6	5	3	4	5
$d(J_i)$	5	7	6	9	10	12

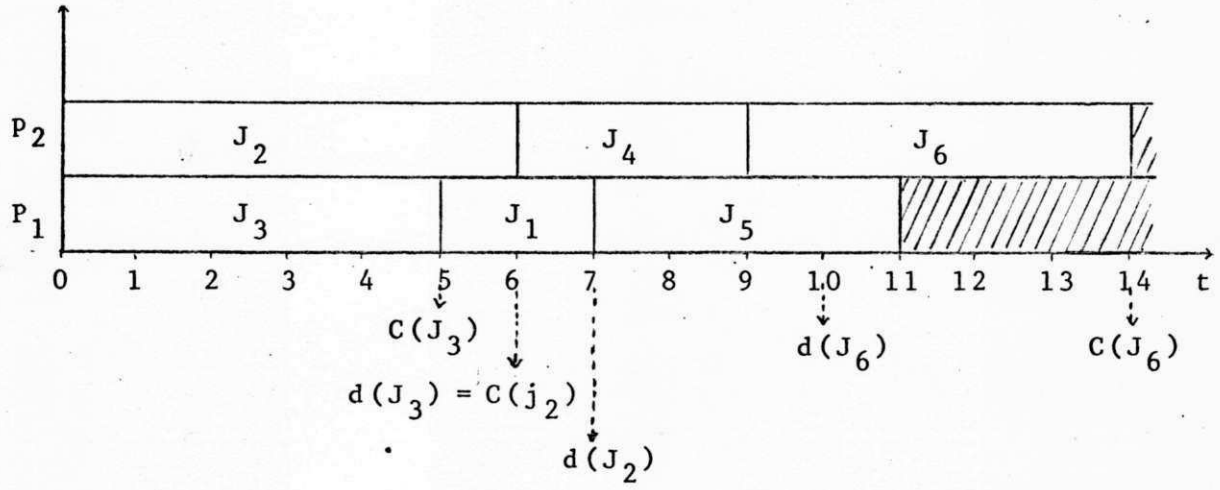


Figura 2.4.

Na Figura 2.4 tem-se a representação gráfica de uma possível schedule para o exemplo dado:

$D(J_1) = C(J_i) - d(J_i) = 7 - 5 = 2$

$D(J_2) = 6 - 7 = -1$

$D(J_3) = 5 - 6 = -1$

$D(J_4) = 9 - 9 = 0$

$$D(J_5) = 11 - 10 = 1$$

$$D(J_6) = 14 - 12 = 2$$

$$D_{max} = \max \{D(J_i)/1 \leqslant i \leqslant 6\} \rightarrow D_{max} = 2$$

$$N_A = U(J_1) + \ldots + u(J_6) = 1 + 0 + 0 + 0 + 1 + 1 = 3$$

Com os dados adicionais seguintes, calcula-se A_ω

$\omega(J_1)$	$\omega(J_2)$	$\omega(J_3)$	$\omega(J_4)$	$\omega(J_5)$	$\omega(J_6)$
2	3	5	4	2	5

$$A_\omega = \sum_{i=1}^{6} \omega(J_i)\, A(J_i) = 2 \times 2 + 0 \times 3 + 0 \times 5 + 0 \times 4 + 1 \times 2 + 2 \times 5 = 16$$

2.3 - SCHEDULING DE TAREFAS INDEPENDENTES EM PROCESSADORES PARALELOS NÃO IDÊNTICOS

O problema de scheduling de tarefas independentes em processadores paralelos não idênticos pode ser considerado como uma extensão do problema anterior onde os processadores disponíveis em paralelo são idênticos. A diferença está no fato de que aqui o desempenho dos processadores difere em relação à execução das tarefas. Para melhor clarificar este problema suponha um centro de computação com tipos diferentes de compu

tadores, onde os mesmos estão disponíveis simultaneamente para a operação de jobs que, por sua vez, independem um do outro.

O problema pode ser caracterizado por:

a) O sistema é um conjunto de m processadores não idênticos $\{P_1, \ldots, P_m\}$ que operam em paralelo e que estão continuamente disponíveis, nunca mantendo-se ociosos e sendo que os mesmos só podem processar uma única tarefa de cada vez.

b) O conjunto $T = \{J_1, \ldots, J_n\}$ de tarefas independentes, e $\mu_1, \ldots, \mu_m$ as funções de tempo de processamento definidas em T, de maneira que a tarefa J_i requer a quantidade $\mu_j(J_i)$ de tempo para ser processada em P_j, $1 \leqslant j \leqslant m$. Temos também que cada tarefa requer somente um único processador.

c) Uma vez iniciado o processamento de uma tarefa, ela é processada, totalmente, sem interrupção.

A construção de uma possível schedule segue os mesmos critérios do modelo anterior, como também a representação da mesma através do diagrama de Gantt.

Pode-se também aqui proceder com as fundamentais medidas de desempenho de uma schedule vistas nos problemas anteriores, apenas com a diferença que as decisões refletirão, além

de paralelismo de recursos, diferentes desempenhos para cada processador em relação a cada tarefa. Será mostrada esta diferença através de um exemplo.

Exemplo 2.3.

Sejam 2 processadores não idênticos disponíveis (m = 2) e abaixo a relação das tarefas com seus respectivos tempos de processamento:

(J_i)	J_1	J_2	J_3	J_4	J_5	J_6
$\mu_1(J_i)$	2	6	5	2	6	7
$\mu_2(J_i)$	4	3	3	3	5	4

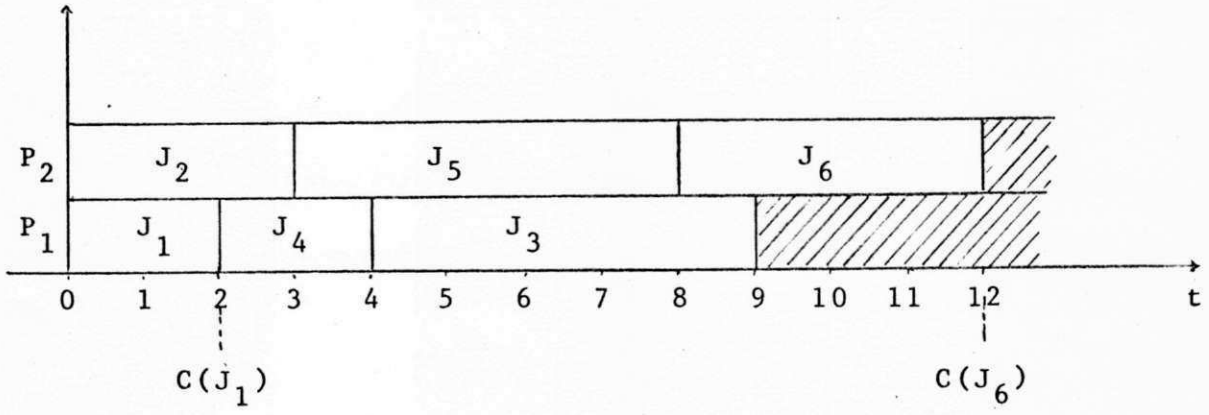


Figura 2.5.

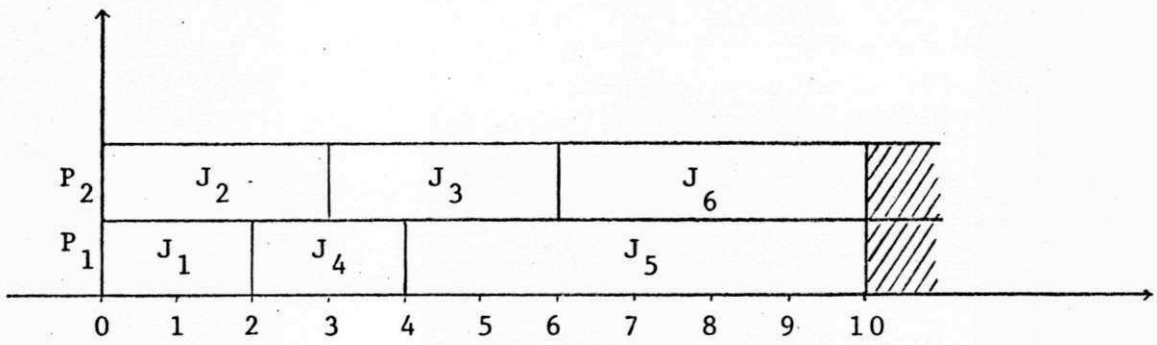


Figura 2.6.

As Figuras 2.5 e 2.6 representam duas possíveis schedules para o exemplo acima.

CAPÍTULO III

JUSTIFICATIVA: PARA OS PROCEDIMENTOS HEURÍSTICOS E ANÁLISE DO PIOR CASO

3.1 - COMPLEXIDADE DOS PROBLEMAS

Quando com um nº de passos limitado por polinômio pode-se encontrar a solução ótima de um problema, isto é, quando o esforço computacional requerido para resolução de um problema é limitado por um polinômio, o algoritmo usado para encontrar tal solução é dito limitado polinomialmente e o problema correspondente é denominado solucionável polinomialmente. Pro

blemas com essas características como fluxo em rede, caminhamento em árvore, problema do menor caminho e alguns especiais problemas de scheduling podem ser resolvidos eficientemente em um computador.

Contudo, existe uma grande classe de importantes problemas combinatoriais com a propriedade de que todos os algoritmos encontrados até o momento para resolvê-los crescem exponencialmente com o tamanho do problema. Até o momento, os mesmos são considerados intratáveis e um ponto crucial é se continuarão intratáveis para sempre. Para um grande número destes problemas, chamados na literatura de NP-Completos, existe uma importante propriedade que os torna equivalentes. Pode-se enunciar esta propriedade da seguinte forma: se for possível encontrar um algoritmo polinomial para resolver um desses problemas, então é possível encontrar um algoritmo polinomial para qualquer um outro problema NP-Completo, de acordo com um procedimento que varia de um problema para outro. Assim, ou existem algoritmos polinomiais para todos os problemas NP-Completos, ou nenhum deles tem tal algoritmo. O que se pode afirmar, no momento, é que existe um grande pessimismo dos estudiosos no assunto, pois estes não acreditam que venham encontrar solução polinomial para tais problemas.

Ao se estudar a complexidade de um problema é muito importante identificá-lo como um problema NP-Completo, o que normalmente não é simples, porque já exclui, de imediato, a via

bilidade de uma busca por algoritmos de otimização que possam ser utilizados na resolução do problema, e, segundo, como conseqüência, porque sugere uma alternativa de solução como, por exemplo, uma abordagem através de procedimentos heurísticos.

Deve ficar claro, neste ponto, que a discussão acima teve um caráter bastante informal e procurau apenas mostrar a existência de problemas para os quais não se conseguem soluções ótimas de tempo polinomial; um estudo com bastante formalismo pode ser encontrado em (3). Na Pesquisa Operacional existem problemas clássicos, como o problema do caixeiro viajante, o de encontrar o número cromático de um grafo, etc. que são considerados NP-Completos. No caso específico de scheduling, apesar da complexidade de alguns problemas ser ainda uma questão em aberto, já se pode afirmar que a maioria é NP-Completo.

Como pode ser visto no quadro abaixo, apenas os problemas de scheduling de tarefas independentes em um único processador e cuja função objetiva envolve minimização de tempo (nº de atrasos, tempo máximo, etc) são, na maioria das vezes , solucionáveis polinomialmente (PS). No caso de processadores paralelos idênticos pode-se notar no quadro que todos os problemas são NP-Completos.

Função Objetiva	Scheduling de tarefas independentes	
	único-processador	Processadores idênticos paralelos
Max $C(J_i)$	trivial	NP-Karp, R. M. (12)
$\Sigma\, \omega(J_i)\, C(J_i)$	PS - Maxwell; Conway; Miller (1)	NP-Bruno
$\Sigma\, C(J_i)$	PS - Maxwell; Conway; Miller (1)	NP-Lenstra, J; Brucker, P (9)
$\Sigma\, \omega(J_i)\, T(J_i)$	NP - Lenstra, J; Brucker; P (9)	NP-(ver (2))
$\Sigma\, T(J_i)$	em aberto	NP-Lenstra, J; Brucker, P (9)
$\Sigma\, U(J_i)$	PS - Moore (8)	NP-Lenstra, J; Brucker, P (9)
max $L(J_i)$	PS - Lawler (13)	NP-Karp, R. M. (12)

OBS.: O modelo de tarefas independentes em processadores não idênticos paralelos pode ser considerado como uma extensão do modelo de processadores idênticos paralelos, por analogia também são NP - Completos.

3.2 - SOLUÇÕES PARA PROBLEMAS COMPLEXOS

Problemas de scheduling aparecem de várias formas na vida real. Porém, basicamente, todos tomam a conotação de problemas de otimização sujeitos a restrições. Nestes problemas tem-se sempre uma coleção de tarefas para serem programadas em um sistema com restrições tecnológicas e de recursos. Esta programação deverá ser feita de forma que minimize (ou em alguns casos maximize) uma dada função objetiva. Portanto para scheduling deve-se incluir uma variedade de técnicas para a resolução dos problemas.

Muitos estudos desenvolvidos neste assunto voltam-se para o refinamento, aplicação e avaliação de procedimentos combinatoriais, de técnicas de simulação, de métodos em redes de planejamento e de resoluções heurísticas. Isto, porque, além da complexidade com que normalmente estes problemas se apresentam, não é simples selecionar uma técnica adequada ou mesmo um procedimento que seja efetivamente coerente com um determinado tipo de problema.

Embora não existam dificuldades para se planejar algoritmos de otimização (isto é, algoritmos que produzem schedules ótimas) os mesmos se tornam ineficientes diante de problemas de scheduling com dimensões realísticas. Na realidade, estes problemas continuam intratáveis por procedimentos exatos, exceto quando se apresentam especialmente estruturados e com re

duzida quantidade de tarefas.

Os resultados pessimistas encontrados até agora, mostrando que a maioria dos problemas de scheduling pertencem à classe NP-Completos, vêm reforçar ainda mais a importância dos procedimentos heurísticos. No caso específico deste trabalho, onde se trata de scheduling em tarefas independentes, pode-se observar através da classificação dos problemas em 3.1 que a grande maioria pertence à classe NP-Completo , justificando assim a aplicação de procedimentos heurísticos na resolução dos mesmos.

Para melhor ilustrar o quanto é importante o procedimento utilizado na resolução de um determinado problema, suponha-se a situação onde a codificação de um computador para um algoritmo de programação dinâmica permite que 1000 subconjuntos sejam avaliados em um segundo. Então, a solução de um problema de 25 tarefas consumiria várias horas em tempo de computação. Esta quantidade de tempo de computação pode não ser acessível e, mesmo se fosse, o custo de tão grande tempo pode bem sobrepujar os benefícios que sejam derivados do uso de programação dinâmica. É importante considerar demandas computacionais de uma técnica de otimização sempre que se pretende usá-la. Quando a capacidade de computação é cara ou escassa, ou quando decisões scheduling, necessitam de resolução em minutos, problemas de 5 a 10 tarefas podem usualmente ser resolvidos rapidamente. Mas, se o número de tarefas aumentasse para pelo menos 20, já não se pode dizer a mesma coisa quanto à

eficiência do algoritmo de otimização. Com esse número razoavelmente grande de tarefas, o problema talvez seja mais adequadamente abordado por procedimento heurísticos.

Quando se fala em algoritmo heurístico ou de aproximação, é importante fazer uma avaliação dos mesmos. Se soluções aproximadas serão encontradas, é necessário saber até que ponto estas soluções satisfazem o exemplo particular. Muitos estudiosos têm-se empenhado em desenvolver estes algoritmos com diferentes abordagens e os mesmos têm conseguido, até certo ponto, resultados satisfatórios dependendo da complexidade do algoritmo, visto que, quanto maior a complexidade destes, maior será a dificuldade em analisar e avaliar o seu desempenho. Será colocado aqui um maior formalismo nesta discussão, para com isto abordar posteriormente um outro aspecto importante relacionado a procedimentos heurísticos. O valor ótimo de uma schedule para um exemplo partícular I, que se denota por OPT(I), é o mínimo (ou em alguns casos o máximo) dos valores de todas as possíveis schedules. Um Algoritmo A de scheduling, para um problema particular, é um procedimento que, dado qualquer exemplo I daquele problema, produz para I uma possível schedule encontrada por A quando aplicado em I. Se A(I) é sempre igual a OPT(I), então, A é chamado de algoritmo de otimização. Caso contrário, A é chamado um algoritmo de aproximação, com a esperança que A encontrará schedules perto de ótimo.

Um método tradicional para avaliação dos algoritmos

de aproximação, segundo Johnson (5), é executá-los em uma amostra selecionada de exemplos do problema. Sem dúvida, para alguns algoritmos este é ainda o único método de avaliação possível, tendo, conduto, algumas desvantagens. Primeiro, é a dificuldade de escolha de um convincente conjunto de amostras dos exemplos de um problema, que reflita os exemplos que são encontrados na prática, existindo sempre o perigo de omissão, através da escolha destes exemplos, para os quais o algoritmo trabalha pobremente. Segundo, é a dificuldade de obter através deste método uma absoluta medida de desempenho para tal algoritmo. Visto que é muito difícil encontrar a schedule ótima para comparar com a solução heurística, esta abordagem de avaliação parece melhor indicada para comparação de soluções heurísticas, que para determinação quão perto do ótimo está o algoritmo.

Uma outra alternativa teórica é a avaliação dos algoritmos de aproximação usando técnicas probabilisticas. Por exemplo, poder-se-ia derivar o valor esperado de A(I) e OPT(I) e compará-los. Resultados deste tipo, não só exclusivamente para problemas de scheduling, têm aparecido recentemente. Embora esse método possa produzir informações interessantes e úteis, ele tem também suas desvantagens. A primeira delas seria proveniente da dificuldade, muitas vezes encontrada, para determinar uma distribuição de probabilidade que espelhe os exemplos dos problemas que aparecem na prática.

Embora os métodos descritos acima possam dar algumas indicações do caso médio de desempenho de um algorítmo, nenhum deles diz alguma coisa sobre como A(I) estará perto do OPT(I) em um exemplo particular. É neste ponto que a garantia de desempenho, que seria um outro método, pode dar úteis resultados.

Basicamente, uma garantia de desempenho consiste de um teorema que limita o comportamento do pior-caso de um particular algorítmo de aproximação. Para um problema de minimização ele poderia tomar a forma: Para todos os exemplos I, A(I) < r . OPT(I) + d", onde r > 1 e d são constantes especificadas. Em geral, a constante aditiva d será negligenciada (para muitos resultados deste tipo ela é zero). O fator dominante será a razão r. Na análise de um algorítmo o que se pretende é encontrar o menor r para o qual tal teorema pode ser provado, isto é, a melhor garantia possível. Através da determinação da razão de desempenho r para um algoritmo de aproximação, obtêm-se informações que suplementam aquelas obtidas por outros métodos, e que fornecem um útil e rigoroso valor, com o qual diferentes algorítmos de aproximação podem ser comparados.

Através da construção de exemplos do problema, pode-se mostrar que nenhum melhor valor que o do limite pode ser encontrado. Nestes exemplos o algorítmo tem um desempenho tão pobre quanto permite o limite. Certamente, existem também desvantagens para a análise do pior caso. Na prática, um algorít

mo pode apresentar um desempenho muito melhor do que o faz no pior-caso. O exemplo do problema causando o comportamento do pior-caso, quando criado, pode ser não realístico. Contudo, um bom limite do pior-caso pode ser tranquilizante, o que não aconteceria com um resultado do tipo médio dos casos, dando um limite que é assegurado sempre, não importando qual o exemplo do problema.

Serão abordados alguns algoritmos heurísticos para problemas de único-processador, processadores idênticos paralelos e não idênticos paralelos.

A avaliação destes algoritmos, através de uma garantia de desempenho usando o pior-caso, é feita em problemas de scheduling cuja função objetiva envolve tempo de conclusão das tarefas. Isto é, para as funções envolvendo T_{ω}, D_{max} os algoritmos heurísticos tornam-se bem mais complexos e consequentemente não se tem conseguido resultados satisfatórios neste tipo de avaliação. Como minimizar Tempo de conclusão em um único-processador é trivial, serão feitas avaliações de algoritmos heurísticos usando o pior-caso apenas para problemas de scheduling de tarefas em mais de um processador.

CAPÍTULO IV

PROCEDIMENTO HEURÍSTICO EM UM ÚNICO PROCESSADOR

4.1 - ALGUMAS CONSIDERAÇÕES SOBRE PROCEDIMENTOS NA RESOLUÇÃO DE PROBLEMAS DE ÚNICO PROCESSADOR.

Problemas de scheduling em que tarefas tem um término previsto para a sua conclusão, são os problemas que mais se encontram na prática. Quando o objetivo de um problema é fazer com que as tarefas satisfaçam, o máximo possível, o término previsto para a sua conclusão, uma natural quantificação desse objetivo envolve a medida de desempenho de uma schedule de

nominada "Atraso" (A), definida no Capítulo II. E, no caso de único processador, um problema fundamental é a minimização da média dos atrasos em uma schedule ($\bar{A}$).

Existem técnicas combinatoriais de otimização como branch and bound, programação dinâmica (vide Baker, cap 3. (2)) que têm sido aplicadas nas resoluções de problemas de sequenciação e, com maior ênfase, na minimização de $\bar{A}$. Mas, essas técnicas além de serem mais complexas se comparadas com técnicas heurísticas são, na maioria das vezes, impraticáveis porque o tempo de computação cresce exponencialmente com o número de tarefas. Por isso, será abordado aqui um importante procedimento heurístico para minimizar $\bar{A}$ no caso de único processador e que foi desenvolvido por Wilkerson - Irwin (11).

4.2 - ALGORITMO DE WILKERSON - IRWIN

O procedimento heurístico desenvolvido por Wilkerson e Irwin se baseia fundamentalmente no uso de propriedades oriundas das permutações de pares de tarefas adjacentes em uma schedule.

Considere uma schedule S em que as tarefas J_i e J_k são adjacentes na sequência, e uma schedule S' que é idêntica a S, exceto que as tarefas J_i e J_k são permutadas, conforme Figura 4.1. Nas schedules, B representa o conjunto das tarefas que precede as tarefas J_i e J_k, t_B representa o ponto, no tempo, em que inicia o processamento da tarefa J_i em S e que inicia a tarefa J_k em S', e C representa o conjunto das tarefas

que segue J_i e J_k.

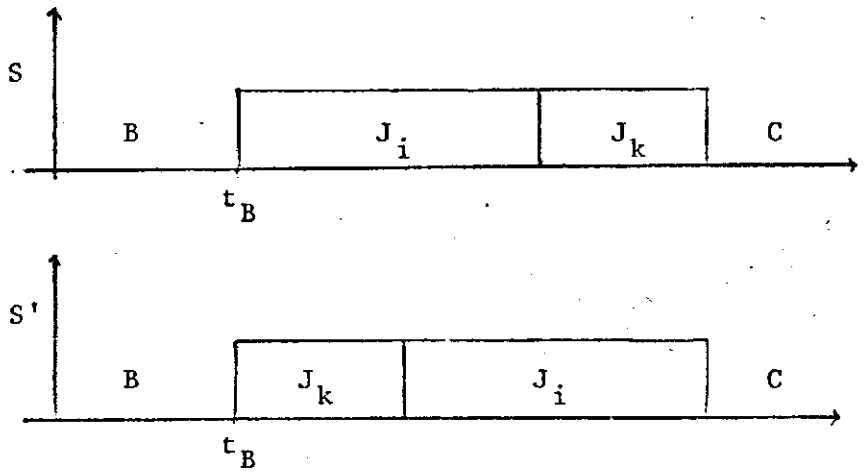


Figura 4.1.

Assumindo que $\mu(J_i) \geq \mu(J_k)$, se investigará qual das duas sequências S ou S' produzirá um resultado melhor, isto é, o Atraso total será menor ou maior com a troca de processamento das tarefas J_i e J_k. Ao invés de comparar o Atraso total (A) para as 2 sequências, será suficiente comparar as contribuições para êle oriundas das tarefas J_i e J_k, visto que a contribuição das tarefas restantes será a mesma nas 2 sequências.

Assim, considerando que $A_s(J_i)$ é o atraso de J_i na schedule S, tem-se que:

$$A_{ik} = A_s(J_i) + A_s(J_k) = \max\{t_b + \mu(J_i) - d(J_i),\ 0\} + \max\{t_b + \mu(J_i) + \mu(J_k) - d(J_k),\ 0\}$$

$$A_{ki} = A_{s'}(J_k) + A_{s'}(J_i) = \max\{t_b + \mu(J_k) - d(J_k),\ 0\} + \max\{t_b + \mu(J_k)$$

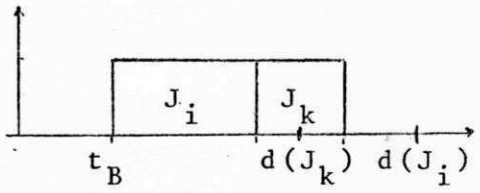
$$+ \mu(J_i) - d(J_i),\ 0$$

Na análise das alterações produzidas no atraso da schedule pela permutação das tarefas J_i e J_k no tempo t_B, os seguintes casos devem ser considerados:

1º Caso: $d(J_i) \geqslant d(J_k)$

1.1. $t_B + \mu(J_i) \leqslant d(J_i)$

Uma possível schedule:



$$A_{ik} = \max \{t_B + \mu(J_i) + \mu(J_k) - d(J_k),\ 0\}$$

$$A_{ki} = \max \{t_B + \mu(J_k) - d(J_k),\ 0\} + \max\{t_B + \mu(J_k) + \mu(J_i) - d(J_i),\ 0\}$$

Pode-se notar que A_{ik} é pelo menos tão grande quanto o primeiro máximo em A_{ki} (visto que $\mu(J_i) > 0$) e pelo menos tão grande quanto o segundo (visto que $d(J_i) \geqslant d(J_k)$). Portanto, se um, ou os 2 máximos em T_{ki} são zeros, tem-se que $A_{ik} \geqslant A_{ki}$.

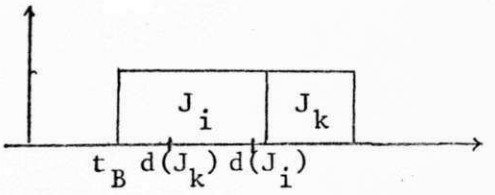
Suponha agora que nenhum termo em A_{ki} e A_{ik} seja zero. Então:

$$A_{ik} - A_{ki} = (t_B + \mu(J_i) + \mu(J_k) - d(J_k)) - (t_B + \mu(J_k) - d(J_k)) - (t_B + \mu(J_k) + \mu(J_i) - d(J_i)) = -t_B - \mu(J_k) + d(J_i) \geqslant -t_B - \mu(J_i) + d(J_i) \geqslant 0$$

Logo, 1.1. produz $A_{ik} \geqslant A_{ki}$. Portanto, é preferível que a tarefa J_k, que tem o término previsto mais cedo, preceda a tarefa J_i.

1.2. $t_B + \mu(J_i) > d(J_i)$

Uma possível schedule:



$$A_{ik} = t_B + \mu(J_i) - d(J_i) + t_B + \mu(J_i) + \mu(J_k) - d(J_k)$$

$$A_{ki} = \max \{t_B + \mu(J_k) - d(J_k), 0\} + t_B + \mu(J_k) + \mu(J_i) - d(J_i)$$

$$A_{ik} - A_{ki} = t_B + \mu(J_i) - d(J_k) - \max \{t_B + \mu(J_k) - d(J_k), 0\}$$

Se $t_B + \mu(J_k) - d(J_k)$ é negativo, então:

$$A_{ik} - A_{ki} = t_B + \mu(J_i) - d(J_k) \geqslant t_B + \mu(J_i) - d(J_i) \geqslant 0$$

Por outro lado

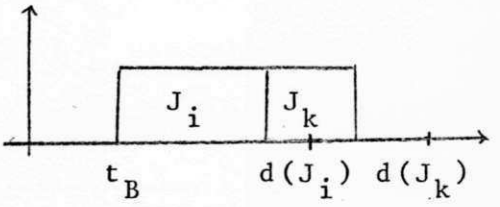
$$A_{ik} - A_{ki} = t_B + \mu(J_i) - d(J_k) - (t_B + \mu(J_k) - d(J_k)) = \mu(J_i) - \mu(J_k) \geqslant 0$$

Logo, 1.2. produz $A_{ik} \geqslant A_{ki}$. Assim, neste caso, é preferível ter também a tarefa J_k precedendo a tarefa J_i.

2º Caso: $d(J_i) < d(J_k)$

2.1. $t_B + \mu(J_i) \leqslant d(J_i)$

Uma possível schedule:



$A_{ik} = \max \{t_B + \mu(J_i) + \mu(J_k) - d(J_k),\ 0\}$

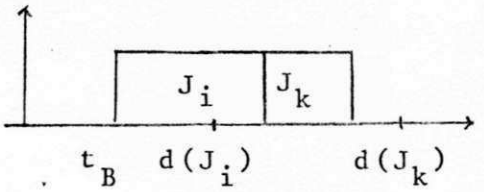
$A_{ki} = \max \{t_B + \mu(J_k) + \mu(J_i) - d(J_i),\ 0\} \geqslant T_{ik}$

Logo, 2.1. produz $A_{ki} \geqslant A_{ik}$. Portanto, é preferível ter a tarefa J_i, com o término previsto mais cedo, precedendo a tarefa J_k.

2.2. $t_B + \mu(J_i) > d(J_i)$

2.2.1. $t_B + \mu(J_i) + \mu(J_k) \leqslant d(J_k)$

Uma possível schedule:



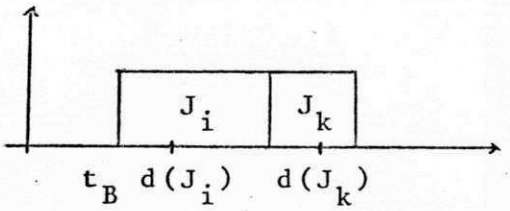
$A_{ik} = t_B + \mu(J_i) - d(J_i)$

$A_{ki} = t_B + \mu(J_k) + \mu(J_i) - d(J_i) \geqslant A_{ik}$

Logo, 2.2.1. produz $A_{ki} \geqslant A_{ik}$, sendo assim preferível ter a tarefa J_i (que tem o término previsto mais cedo) precedendo a tarefa J_k.

2.2.2. $t_B + \mu(J_i) \leqslant d(J_k) < t_B + \mu(J_i) + \mu(J_k)$

Uma possível schedule:



$$A_{ik} = t_B + \mu(J_i) - d(J_i) + t_B + \mu(J_i) + \mu(J_k) - d(J_k)$$

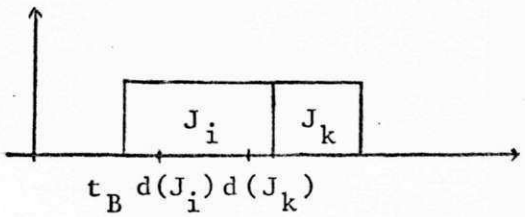
$$A_{ki} = t_B + \mu(J_k) + \mu(J_i) - d(J_i)$$

$$A_{ik} - A_{ki} = t_B + \mu(J_i) - d(J_k) \leqslant 0$$

Logo, 2.2.2. produz $A_{ki} \geqslant A_{ik}$, sendo assim preferível ter a tarefa J_i (que tem o término previsto mais cedo) precedendo a tarefa J_k.

2.2.3. $t_B + \mu(J_i) > d(J_k)$

Uma possível schedule:



$$A_{ik} = t_B + \mu(J_i) - d(J_i) + t_B + \mu(J_i) + \mu(J_k) - d(J_k)$$

$$A_{ki} = \max \{t_B + \mu(J_k) - d(J_k), 0\} + t_B + \mu(J_k)$$

$$+ \mu(J_i) - d(J_i)$$

$$A_{ik} - A_{ki} = t_B + \mu(J_i) - d(J_k) - \max\{t_B + \mu(J_k) - d(J_k), 0\}$$

$$= \begin{cases} t_B + \mu(J_i) - d(J_k) \geqslant 0 \\ \mu(J_i) - \mu(J_k) \geqslant 0 \end{cases}$$

Portanto, 2.2.3. produz $A_{ik} \geqslant A_{ki}$, sendo assim preferível ter a tarefa J_k (com o menor tempo de processamento) precedendo a tarefa J_i.

Para $\mu(J_i) \geqslant \mu(J_k)$ foram presquisadas todas as possibilidades, o que deverá ser feito também para $\mu(J_i) \leqslant \mu(J_k)$. Para ambos os casos pode-se concluir que, entre J_i e J_k, é preferível ter a tarefa com o término previsto mais cedo vindo primeiro na sequência, exceto quando: para $\mu(J_i) \geqslant \mu(J_k)$ ocorrer $d(J_i) < d(J_k)$, $t_B + \mu(J_i) > d(J_i)$ e $t_B + \mu(J_i) > d(J_k)$, e para $\mu(J_i) \leqslant \mu(J_k)$ ocorrer $d(J_i) > d(J_k)$ e $t_B + \mu(J_k) > d(J_i)$ onde, em tais circunstâncias, a tarefa com o menor tempo de processamento deverá vir primeiro na sequência.

Nestas condições, ou seja, exceto quando $\left[\mu(J_i) \geqslant \mu(J_k)\right.$ e $t_B + \mu(J_i) > d(J_k) > d(J_i)\left.\right]$ ou $\left[\mu(J_i) \leqslant \mu(J_k)\right.$ e $t_B + \mu(J_k) > d(J_i) > d(J_k)\left.\right]$ é preferível ter a tarefa com término previsto mais cedo vindo primeiro na sequência. Isto pode ser enuncia

do mais formalmente como uma regra de decisão:

"Ao se comparar J_i e J_k no tempo t_B, é preferível ter a tarefa com o término previsto mais cedo vindo primeiro na sequência, exceto quando:

$t_B + \max \{\mu(J_i), \mu(J_k)\} > \max \{d(J_i), d(J_k)\}$, em tal caso, a tarefa com o menor tempo de processamento viria primeiro na sequência".

O algorítmo de Wilkerson-Irwin, que minimiza a média dos atrasos ($\bar{A}$) em uma schedule, usa esta regra de decisão em um procedimento heurístico, fazendo comparações em pares de tarefas na construção de uma sequência. O algorítmo trabalha com 2 listas ordenadas. Uma lista L_1 com as tarefas que não foram ainda programadas, ordenada segundo o critério EDD (Earliest Due Date), isto é, em ordem não decrescente em relação aos $d(J)$; e a outra lista L_2 com as tarefas já programadas, sujeita a possíveis revisões.

Em cada estágio do algorítmo, a primeira tarefa de L_1 é removida a fim de se aplicar a regra de decisão; esta tarefa é denominada pivô. Seja:

α o índice da última tarefa de L_2,

β o índice da tarefa pivô e

γ o índice da primeira tarefa de L_1.

Na Figura 4.2 (a) está descrito a localização das tarefas J_α, J_β e J_γ. Para a avaliação da regra de decisão será usado $t_B = t_\alpha$. Em cada estágio o algoritmo aplica a regra de decisão para as tarefas J_β e J_γ, onde $d(J_\beta) \leqslant d(J_\gamma)$, da seguinte forma: Verifica-se em quais situações a tarefa J_β deve vir na sequência antes de J_γ. Isto é, se 1) $t_\alpha + \max\{\mu(J_\beta), \mu(J_\gamma)\} \leqslant \max\{d(J_\beta), d(J_\gamma)\}$, ou 2) $t_\alpha + \max\{\mu(J_\beta), \mu(J_\gamma)\} > \max\{d(J_\beta), d(J_\gamma)\}$ e $\mu(J_\beta) \leqslant \mu(J_\gamma)$, então é desejável ter J_β precedendo J_γ para este estágio, na sequência; assim, J_β é adicionada a L_2 (Figura 4.2 (b)). Se esta condição falha, a tarefa J_γ torna-se J_β e J_β retorna para a lista L_1. A regra de decisão é então, aplicada para J_α e J_β. Isto é, se 1) $(t_\alpha - \mu(J_\alpha)) + \max\{\mu(J_\alpha), \mu(J_\beta)\} \leqslant \max\{d(J_\alpha), d(J_\beta)\}$, ou 2) $(t_\alpha - \mu(J_\alpha)) + \max\{\mu(J_\alpha), \mu(J_\beta)\} > \max\{d(J_\alpha), d(j_\beta)\}$ e $\mu(J_\alpha) \leqslant \mu(J_\beta)$, então é desejável ter J_α precedendo J_β na sequência; assim, J_β é adicionada a L_2 (Figura 4.2 (c)). Contudo, se esta regra de decisão também falha, tem-se uma condição especial para tomada de decisão. Aqui, a tarefa J_α é removida da lista L_2 e colocada na lista L_1 em ordem EDD (Fig. 4.2 (d)). A última tarefa de L_2 passa a ser J_α, e a regra de decisão é novamente aplicada para as tarefas J_α e J_β. Se, ao remover a tarefa de L_2 a mesma ficar vazia, J_β torna-se J_α, a primeira tarefa de L_1 torna-se pivô e aplica-se a regra de decisão para J_β e J_γ.

Segue-se uma descrição forma do procedimento, lembrando que uma sequência das tarefas determina a solução do

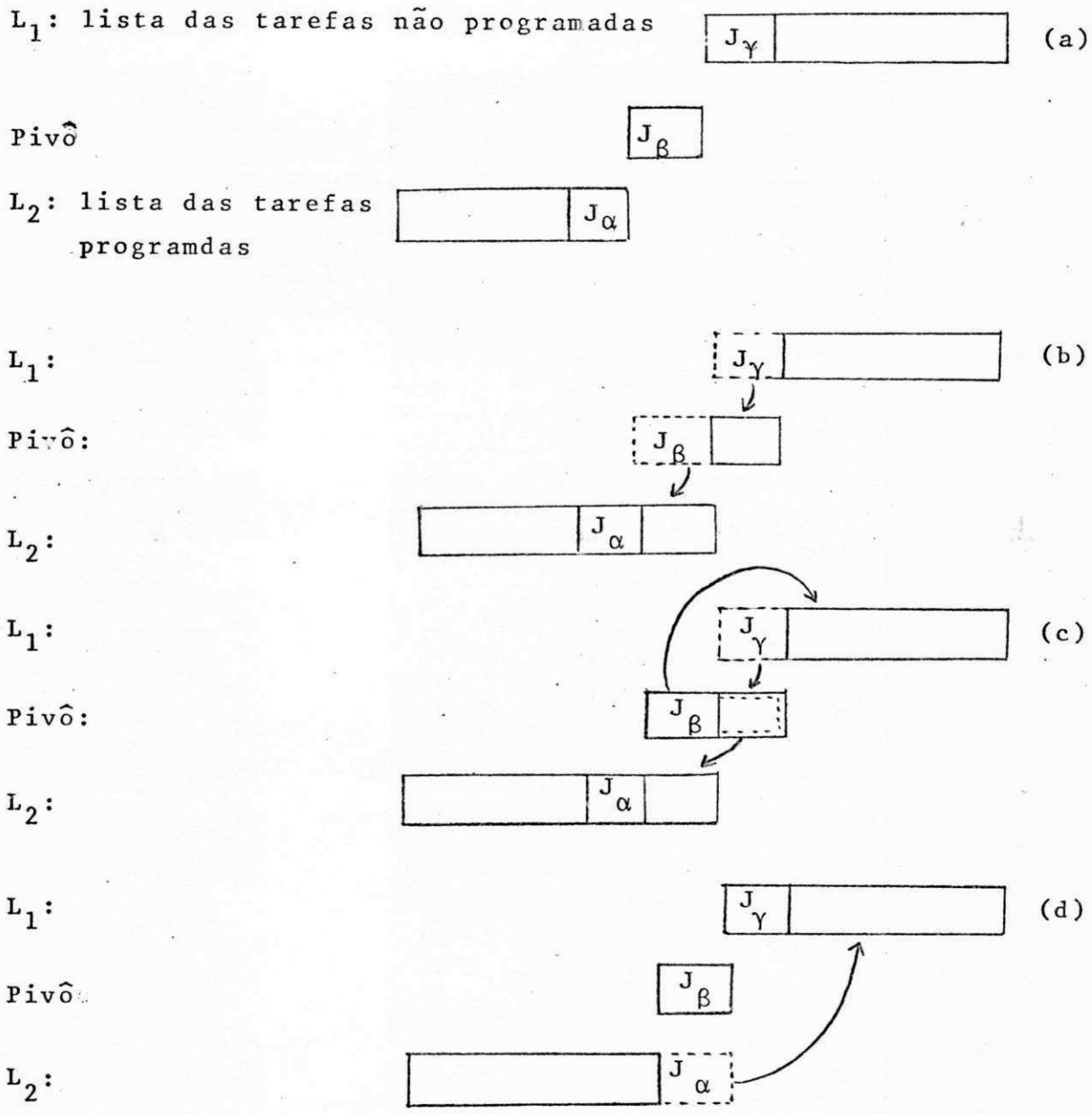


Fig. 4.2 - Manipulação das tarefas pelo algoritmo de Wilkerson Irwin

A entrada e saída de dados para o algorítmo são:

Entrada: A lista L_1 composta pela tarefas independentes J_i, ordenadas segundo o critério EDD, com seus respectivos tempos de processamento $\mu(J_i)$ e término previsto para a conclusão da tarefa $d(J_i)$, $1 \leqslant i \leqslant n$.

Saída: A lista L_2, que dá a sequência em que as tarefas devem ser programadas.

Assume-se aqui, que ao se deletar uma tarefa da lista L_1, esta será sempre a primeira tarefa, passando a próxima a ser automaticamente a tarefa J_γ. E, ao se inserir uma tarefa na lista L , esta ocupará a última posição e passará automaticamente a ser a tarefa J_α. Por equivalência, a condição 1) ou 2) acima será substituída por:

1) $t_\alpha + \max \{\mu(J_i), \mu(J_k)\} \leqslant \max \{d(J_i), d(J_k)\}$ ou 2') $\mu(J_i) \leqslant \mu(J_k)$, tornando o algorítmo mais simples.

Algoritmo de Wilkerson - Irwin

(Um procedimento heurístico para minimizar $\bar{A}$)

Passo 1: (Inicialização)

$L_2 \leftarrow \phi$;

$t_\alpha \leftarrow 0$;

$J_\beta \leftarrow 1^{\underline{a}}$ tarefa de L_1; $J_\gamma \leftarrow 2^{\underline{a}}$ tarefa de L_1;

<u>if</u> max $\{\mu(J_\beta), \mu(J_\gamma)\} \leqslant$ max $\{d(J_\beta), d(J_\gamma)\}$ <u>or</u> $\mu(J_\beta) \leqslant \mu(J_\gamma)$
<u>then</u>

<u>begin</u>

$L_2 \leftarrow L_2 \cup \{J_\beta\}$;

$J_\beta \leftarrow J_\gamma$;

<u>end</u>

<u>else</u>

$L_2 \leftarrow L_2 \cup \{J_\gamma\}$;

$L_1 \leftarrow L_1 - \{J_\beta, J_\gamma\}$;

$t_\alpha \leftarrow t_\alpha + \mu(J_\alpha)$;

Passo 2: (Programação das tarefas restantes de L_1)

<u>while</u> $L_1 \neq \phi$ <u>do</u>

<u>begin</u>

<u>if</u> t_α + max $\{\mu(J_\beta), \mu(J_\gamma)\} \leqslant$ max $\{d(J_\beta), d(J_\gamma)\}$ or $\mu(J_\beta) \leqslant$
$\mu(J_\gamma)$ <u>then</u>

```
        begin

          L2 ← L2 U {Jβ};

          Jβ ← Jγ;

          L1 ← L1 - {Jγ};

          tα ← tα + μ(Jα);

        end

      else

        begin

          trocar Jγ por Jβ;

K:        if tα - μ(Jα) + max {μ(Jα), μ(Jβ)} ≤ max{d(Jα), d(Jβ)}
                                  or μ(Jα) ≤ μ(Jβ) then

            begin

              L2 ← L2 U {Jβ};

              Jβ ← Jγ;

              L1 ← L1 - {Jγ};

              tα ← tα + μ(Jα);

            end

          else
```

```
begin
    L1 ← L1 U {Jα} em ordem EDD;
    L2 ← L2 - {Jα};
    tα ← tα - μ(Jα);
    if L2 = φ then
      begin
        L2 ← {Jβ};
        tα ← μ(Jβ);
        Jβ ← Jγ;
        L1 ← L1 - {Jγ};
      end
    else
      go to K;
  end;
end;
end.
```

Abaixo está uma ilustração do funcionamento do algoritmo através de um exemplo.

Exemplo 4.1

	$\mu(J_i)$	$d(J_i)$
J_1	4	5
J_2	3	6
J_3	7	8
J_4	2	8
J_5	2	17

$L_1 = \{J_1, J_2, J_3, J_4, J_5\}$, as tarefas ordenadas segundo o critério EDD.

$t_\alpha = 0$

$L_2 = \phi$

Comparando as tarefas J_1 e J_2: max {4, 3} < max {5, 6}. Portanto, J_1 será programada primeiro e a mesma torna-se J_γ, e J_2 torna-se J_β.

$L_1 = \{J_3, J_4, J_5\}$

$J_\beta = J_2$

$L_2 = \{J_1\}$

$t_\alpha = 4.$

A regra de decisão é, então, aplicada para J_2 e J_3, isto é, 4 + max {3, 7} > max {6, 8}, mas $\mu(J_2) < \mu(J_3)$, assim J_2 é programada antes de J_3.

$L_1 = \{J_4, J_5\}$

$J_\beta = J_3$

$L_2 = \{J_1, J_2\}$

$t_\alpha = 7.$

Neste ponto, 7 + max {7, 2} > max {8, 8} e $\mu(J_3) > \mu(J_4)$, consequentemente J_3 retorna a L_1, e J_4 torna-se pivô.

$L_1 = \{J_3, J_5\}$

$J_\beta = J_4$

$L_2 = \{J_1, J_2\}$

A regra de decisão será aplicada para J_2 e J_4, isto é, (7 - 3) + max {3, 2} < max {6, 8}. Portanto, J_2 deve ser programada antes de J_4, confirmando a situação anterior. Então, J_4 e J_3 são comparados: 7 + max {2, 7} > max {8, 8}, mas $\mu(J_4) < \mu(J_3)$, assim J_4 deve ser programada antes de J_3.

$L_1 = \{J_5\}$

$J_\beta = \{J_3\}$

$L_2 = \{J_1, J_2, J_4\}$

$t_\alpha = 9.$

Prossegue-se com a aplicação do algoritmo, chegando finalmente ao seguinte resultado:

tarefa	$\mu(J_i)$	$d(J_i)$	$C(J_i)$	$A(J_i)$
J_1	4	5	4	0
J_2	3	6	7	1
J_3	7	8	9	1
J_4	2	8	16	8
J_5	2	17	18	1
Sequência final: $J_1 - J_2 - J_4 - J_3 - J_5$ e $\overline{A} = 11/5$				

No Apêndice, encontra-se uma listagem do programa para o algoritmo de Wilkerson - Irwin, e o mesmo foi testado com os seguintes dados:

	$\mu(J_i)$	$d(J_i)$		$\mu(J_i)$	$d(J_i)$
J_1	5	5	J_{11}	7	40
J_2	2	6	J_{12}	8	42
J_3	3	7	J_{13}	2	43
J_4	6	15	J_{14}	6	44
J_5	1	16	J_{15}	1	50
J_6	4	17	J_{16}	3	60
J_7	2	20	J_{17}	2	68
J_8	10	22	J_{18}	4	70
J_9	5	30	J_{19}	6	72
J_{10}	4	35	J_{20}	6	87

A sequência, apresentada como solução, foi a seguinte: 1 - 2 - 3 - 4 - 5 - 7 - 6 - 9 - 10 - 11 - 13 - 14 - 15 - 12 - 16 - 17 - 18 - 19 - 8 - 20.

CAPÍTULO V

PROCEDIMENTOS HEURÍSTICOS E UMA ANÁLISE DO PIOR CASO EM PROCESSADORES PARALELOS

A determinação de uma schedule ótima é muito dificultada pela necessidade de se tomar dois tipos de decisão: alocação e sequenciação de recursos. Logo, existe um impulso natural, através de resultados analíticos, em trabalhar primeiramente com problemas cujo objetivo seja a minimização do Tempo de conclusão de uma schedule, pois, como já foi dito anteriormente, são problemas menos complexos, se comparados com a minimização do Número de atrasos, Desvio máximo etc. A avaliação de um procedimento heurístico através da análise do pior caso só tem obtido, até o momento, resultados satisfatórios em pro

blemas que envolvam Tempo de conclusão, justamente pela sua menor complexidade.

No caso de processadores paralelos, minimizar Tempo de conclusão de uma schedule é um problema do tipo NP-Completo para $m \geqslant 2$, visto no terceito Capítulo, e, a não ser para problemas muito pequenos, não existe algoritmo de otimização para resolvê-los, principalmente quando $m \geqslant 3$. Certos algoritmos de otimização podem, na prática, trabalhar bem para $m = 2$, quando os processadores são idênticos, apesar de que, para alguns, o tempo de computação cresce exponencialmente.

5.1 - PROCESSADORES IDÊNTICOS

Serão abordados aqui 2 algoritmos heurísticos para minimizar Tempo de conclusão em processadores paralelos idênticos com suas respectivas análises dos piores casos. Para o segundo algoritmo acrescentou-se apenas mais um passo inicial ao primeiro, melhorando sensivelmente o seu desempenho, como pode ser visto através dos exemplos comparativos no final desta unidade. Os resultados foram encontrados com o auxílio do computador através da implementação dos algoritmos, encontrando-se a listagem dos programas no Apêndice.

a) Algoritmo A para minimizar Tempo de conclusão

Basicamente, o algoritmo consiste de: dada uma lista L com n tarefas independentes, construída numa dada ordem, de

signar cada uma delas, uma de cada vez para o processador com o mais cedo tempo de conclusão, a fim de tentar minimizar o crescimento do tempo de conclusão total da schedule.

Neste algoritmo as variáveis têm os seguintes significados. A lista L_j ($1 \leq j \leq m$, onde m representa o número de processadores), consiste das tarefas que deverão ser processadas no j-ésimo processador. Durante o algoritmo, as tarefas já processadas em j serão eliminadas de L e passarão a fazer parte de Lj, onde serão introduzidas, uma a uma, na última posição da lista; t_j representa o tempo corrente de conclusão do processador j na schedule parcial construída até este ponto. Coloca-se, inicialmente, $t_j = 0$ para todo j, bem como as L_j's iniciam vazias, sem nenhum elemento.

A entrada e saída de dados serão como segue:

Entrada: L é uma lista de n tarefas independentes J_i ($1 \leq i \leq n$) numa ordem qualquer, com seus respectivos tempos de processamento $\mu(J_i)$.

Saída: m listas L_j de tarefas, uma para cada processador e A(I) que é o Tempo de conclusão da schedule.

Algoritmo A

Passo 1. (Inicialização)

```
for j ← 1 to m do
    begin
        Lj ← φ;
```

$t_j \leftarrow 0$

end;

Passo 2. (Programação de cada tarefa, uma de cada vez, no processador com o menor tempo corrente de conclusão)

while L não é vazia do

begin

Encontre j com $t_j = \min \{t_j / 1 \leqslant j \leqslant m\}$ e o 1º elemento J_i da lista L;

Seja J o primeiro elemento da lista L;

$t_j \leftarrow t_j + \mu(J_i)$;

$L_j \leftarrow L_j \cup \{J_i\}$;

$L \leftarrow L - \{J_i\}$

end;

Passo 3. (Cálculo do Tempo de conclusão da schedule construída)

$A(I) \leftarrow \max \{t_j / 1 \leqslant j \leqslant m\}$

Este é, claramente, um rápido método heurístico para a construção de possíveis schedules, podendo ser implementado num tempo proporcional a n log m pois, se for olhado o tempo necessário para cada passo do algoritmo, verifica-se que os passos 1 e 3 tomam uma quantidade constante de tempo, enquanto o passo 2 toma 0(n) quantidade de tempo em relação à programação de todas as tarefas. Mas cada tarefa será programada em um

dos n processadores, fazendo aí uma busca do menor t_j numa lista que, caso esteja ordenada, toma $O(\log m)$ quantidade de tempo e, em seguida, possivelmente insere o novo valor em outro lugar da lista, tomando também $O(\log m)$ quantidade de tempo. Pode-se então concluir que a ordem de complexidade de todo o algoritmo é $O(n \log m)$. Ver em Aho, hopcroft, Ulmman, Cap. 3 (3).

Exemplo 5.1

J_i	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
$\mu(j_i)$	6	5	3	7	2	8	4	1

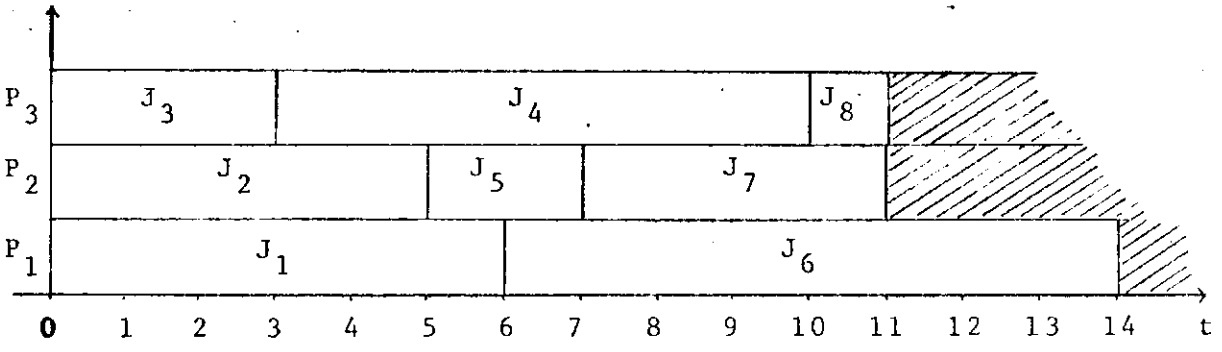


Fig. 5.1 - Schedule resultante da aplicação do Algoritmo A, onde o número de processadores é igual a 3.

O algoritmo começa designando para o processador P_1 a primeira tarefa da lista dada, que é J_1, em seguida designando J_2 para o processador P_2 e J_3 para o processador P_3, pois até aqui todos os t_j são iguais a zero. Neste ponto, conforme mostra a Figura 5.1, o menor tempo de conclusão corrente, isto é, o processador que fica disponível primeiro, é P_3, no ponto em que $t = 3$. A partir deste ponto, P_3 começará a processar J_4,

que é a próxima tarefa da lista, e assim sucessivamente.

Pode-se verificar que o Tempo de conclusão da schedule resultante da aplicação do Algorítmo A é A(I) = 14. Abaixo, na Figura 5.2 será mostrado que OPT(I) = 12.

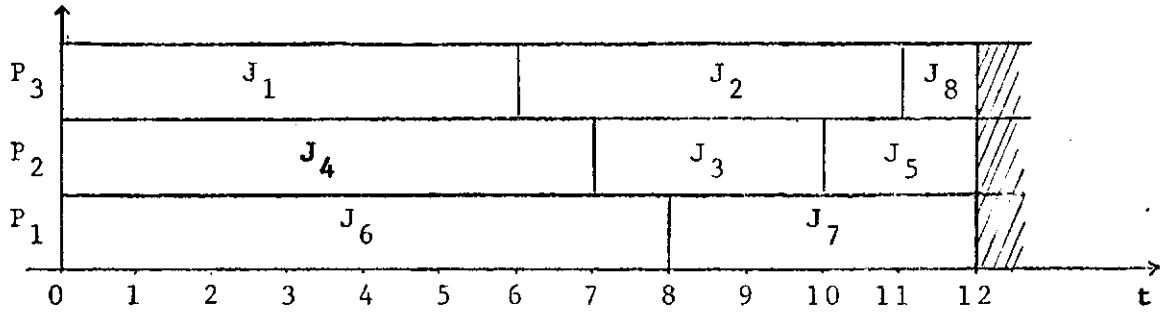


Fig. 5.2 - Schedule Ótima

b) Análise do pior caso para o Algorítmo A

Como visto em 3.2, OPT(I) é o valor ótimo de uma schedule para um exemplo partícular I, de um problema qualquer de scheduling, que tanto pode ser o mínimo, ou em alguns casos o máximo, dos valores de todas as possíveis schedules, sendo que A(I) é um algorítmo para um problema partícular e, quando aplicado em I, produz para I uma possível schedule encontrada por A. Se A(I) = OPT(I), então A(I) é um algoritmo de otimização; caso contrário, é um algorítmo heurístico ou de aproximação, que é o caso aqui considerado. Também já foi dito anteriormente da importância em se avaliar o desempenho dos algoritmos heurísticos, e que este trabalho estaria voltado para a abordagem denominada garantia de desempenho, que é um teorema que limita o comportamento do pior-caso.

Teorema 1. (Graham e Johnson (5)).

Sejam I um exemplo de um problema de scheduling de n tarefas independentes e $L = (J_1, \ldots, J_n)$ uma lista para as tarefas de I. Além disso, sejam dados n processadores iguais que operam em paralelo. Se o algoritmo em questão é o Algoritmo A, descrito acima, então:

$$\frac{A(I)}{OPT(I)} \leqslant (2 - \frac{1}{m})$$

Como encontrar uma maneira para provar tal resultado, quando, como foi visto anteriormente, que a determinação precisa do valor OPT(I) é bastante difícil? Não é necessário, neste caso, conhecer o valor exato de OPT(I), é suficiente conhecer o valor da alguns limites inferiores em OPT(I).

Claramente, devem ser satisfeitos os dois limites inferiores seguintes:

$$OPT(I) \geqslant \max \{\mu(J_i)/1 \leqslant i \leqslant n\}$$

$$OPT(I) \geqslant \frac{1}{m} \sum_{i=1}^{n} \mu(J_i)$$

Considere-se agora a schedule gerada por A e seja J_k uma das tarefas cujo término coincide com A(I). Quando foi de

signado o seu tempo de início para o processamento, já devia mos ter tido $t_j \geqslant A(I) - \mu(J_k)$ para todo $1 \leqslant j \leqslant m$, pela maneira como algoritmo trabalha. Portanto:

$$\sum_{i=1}^{n} \mu(J_i) = \mu(J_k) + \sum_{i=1}^{m} t_j \geqslant \mu(J_k) + m\ (A(I) - \mu(J_k)).$$

Isto implica que: $A(I) \leqslant \frac{m-1}{m}\ \mu(J_k) + \frac{1}{m} \sum_{i=1}^{n} \mu(J_i)$.

Através dos dois limites inferiores pode-se concluir que:

$$A(I) \leqslant \frac{1}{m} \sum_{i=1}^{n} \mu(J_i) + \frac{m-1}{m} \cdot \mu(J_k) \leqslant OPT(I) + \frac{m-1}{m}\ OPT(I) =$$

$$= OPT(I) \quad (2 - \frac{1}{m})$$

ficando assim demonstrado o teorema.

Não se sabe ainda se o limite encontrado é o melhor possível. Pode-se fazer isto, através da construção de exemplos do problema nos quais o algoritmo tem um desempenho tão pobre quanto permite o limite, ficando assim provado que não é possível melhorar tal limite.

Das figuras abaixo, pode-se garantir que o Teorema 1 dá a melhor garantia de desempenho possível para o Algoritmo A, onde foi mostrada uma schedule ótima e a schedule resultan

te da aplicação do algoritmo para um exemplo do problema, especificado por:

Exemplo 5.2

$T = \{J_1, \ldots, J_{2m-1}\}$;

$\mu(J_i) = m - 1, \quad 1 \leqslant i \leqslant m - 1$;

$\mu(J_i) = 1, \quad m \leqslant i \leqslant 2m - 2$;

$\mu(J_{2m-1}) = m$.

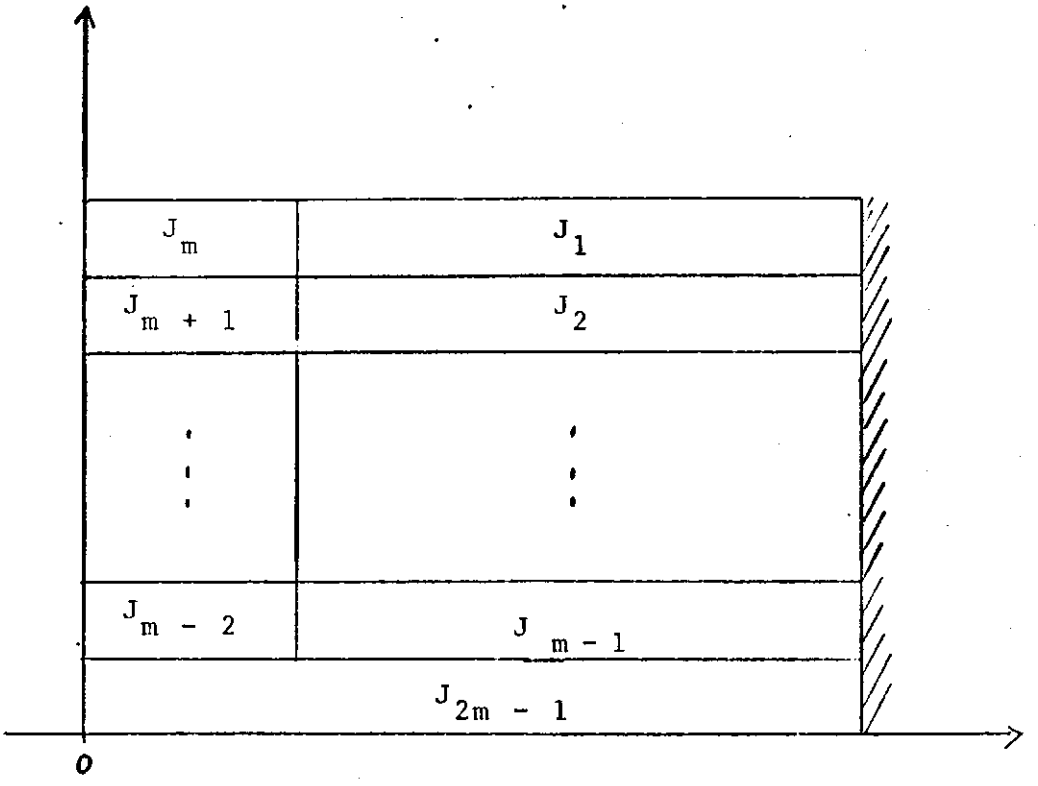


Fig. 5.3 - Schedule Ótima: OPT(I) = m

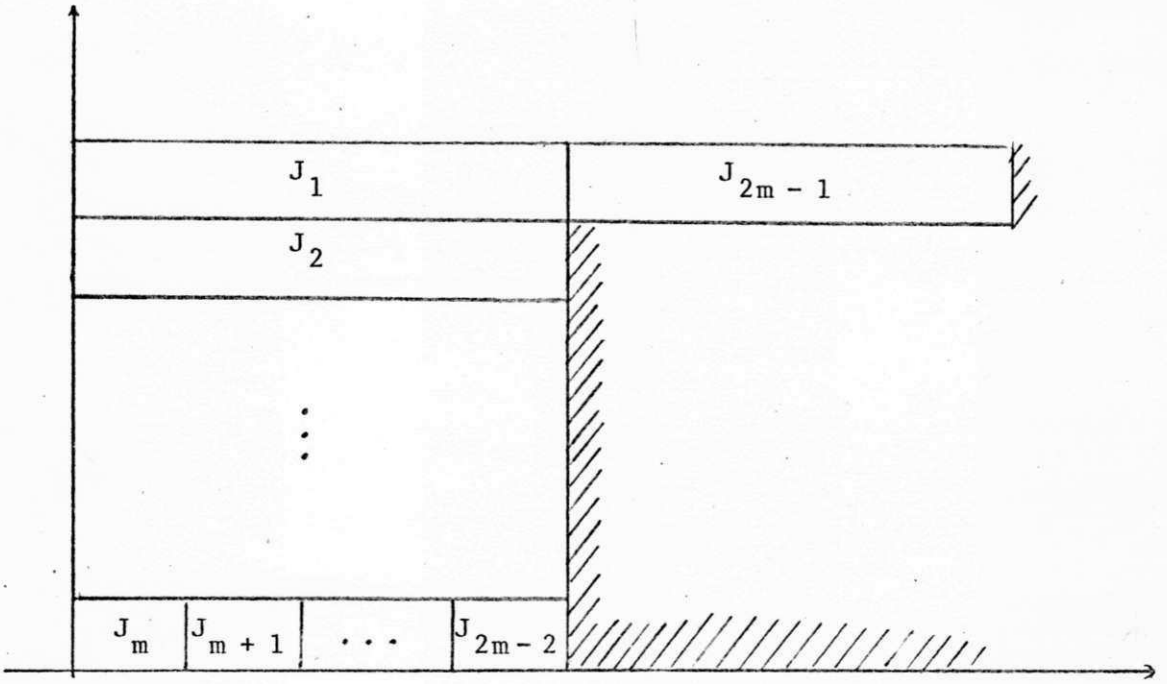


Fig. 5.4 - Schedule resultante da aplicação do Algoritmo A:

$A(I) = 2m - 1$

O exemplo acima do problema produz $\frac{A(I)}{OPT(I)} = 2 - \frac{1}{m}$ para o Algoritmo A.

Tem-se assim determinado o comportamento do pior-caso do algoritmo. Se for considerado o problema geral em que m é fixado, mas especificado como parte do exemplo do problema, a razão de desempenho do pior caso é 2.

Olhando o comportamento do algoritmo, pode-se levantar uma importante questão: Um algoritmo ainda melhor pode ser encontrado? Uma maneira para tentar encontrar é examinando os exemplos de pior-caso para o prévio algoritmo, vendo se existem princípios adicionais de senso comum que poderiam ser usa

dos para evitar tal comportamento.

Olhando para as Figuras 5.3 e 5.4, observa-se que o desempenho pobre do Algoritmo A poderia ser atribuído ao atraso com que a tarefa que requer maior tempo de processamento, foi programada.

Uma maneira natural de evitar isto, seria ordenar, primeiramente, as tarefas em ordem não crescente em relação ao seu tempo de processamento, isto é:

$$\mu(J_1) \geqslant \mu(J_2) \geqslant \mu(J_3) \geqslant \ldots \geqslant \mu(J_n)$$

Assim, será proposto o algoritmo seguinte, que é muitas vezes chamado Algoritmo LPT.

Nota: Quando as tarefas são sequenciadas em ordem não crescente em relação ao tempo de processamento, esta seqüência é chamada de LPT (largest processing time); existindo também outros tipos de sequência, com SPT (shortest processing time) que é contrária a LPT. Dependendo do algoritmo e do objetivo a alcançar pode-se usar critérios diferentes na ordenação das tarefas.

c) Algoritmo B para minimizar Tempo de conclusão.

Algoritmo B (LPT)

Passo 1. Construa uma ordenação LPT para as tarefas.

Passo 2. Aplique-se o Algoritmo A para o conjunto de tarefas reordenadas.

Exemplo 5.3.

Será aplicado o algoritmo para o mesmo exemplo do algoritmo A e será conseguido um resultado melhor.

J_i	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
$\mu(J_i)$	6	5	3	7	2	8	4	1

Passo 1. Reindexação dos índices das tarefas usando o critério LPT.

J_i	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
$\mu(J_i)$	8	7	6	5	4	3	2	1

Passo 2. Aplicação do Algoritmo A para a lista acima

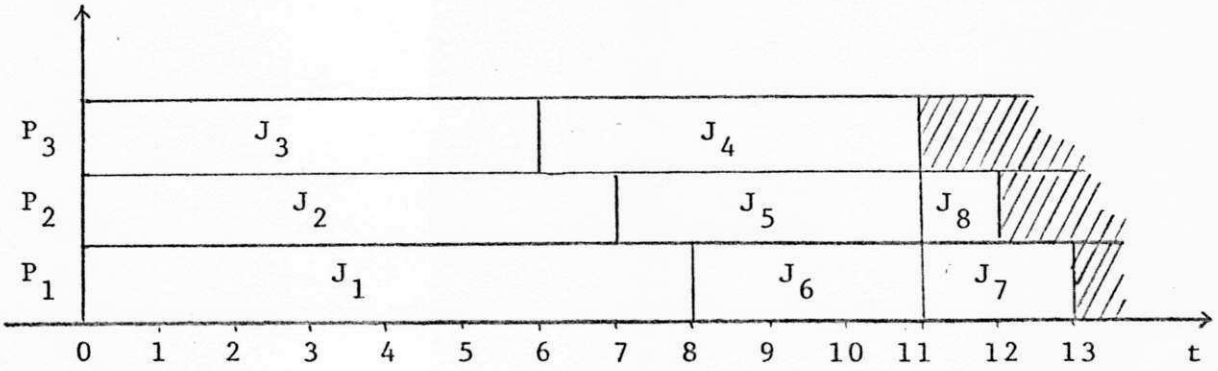


Fig. 5.5 - Schedule resultante da aplicação do Algoritmo B, onde o nº de processadores é igual a 3.

Note-se que o Tempo de conclusão da schedule resultante da aplicação do Algoritmo B é um resultado mais perto do

ótimo, OPT(I) = 12 (Figura 5.2), se comparado com o resultado da aplicação do Algoritmo A, A(I) = 14 (Figura 5.1).

d) Análise do pior-caso para o Algoritmo B (LPT)

Teorema 2. (R. L. Graham (7)).

Se I é um exemplo de um problema de scheduling de n tarefas independentes com m processadores, que operam em paralelo, e se o algoritmo em questão é o Algoritmo B, descrito acima, então:

$$(1) \quad \frac{B(I)}{OPT(I)} \leqslant \frac{4}{3} - \frac{1}{3m}$$

Demonstração:

O teorema é claramente assegurado para m = 1, portanto, pode-se assumir que m $\geqslant$ 2.

Suponha-se que exista um exemplo I de tarefas $T = \{J_1, \ldots, J_n\}$ com tempos de processamento iguais a $\mu(J_i)$, $1 \leqslant i \leqslant n$, que contradizem (1) de tal modo que todos os exemplos com um nº de tarefas menor que n satisfaçam o teorema. Denomina-se por D_b a representação gráfica de uma schedule resultante da aplicação do Algoritmo B a I. Podem acontecer aqui 2 possíveis casos, que serão considerados a seguir:

1º Caso: Suponha-se que exista r < n, com J_r sendo uma

tarefa com o mais tarde tempo de conclusão na schedule (que deve ser B(I)). Se for considerado I' um outro exemplo do problema, com a diferença que $T' = \{J_1, \ldots, J_r\}$ é um subconjunto de T, será visto que o Tempo de conclusão B(I') para T' é exatamente igual a B(I).

Por outro lado, para o valor ótimo OPT(I') para T', é verdade que $OPT(I') \leq OPT(I)$, onde OPT(I) é considerado o Tempo de conclusão para o exemplo I.

$$\frac{B(I')}{OPT(I')} \geq \frac{B(I)}{OPT(I)} > \frac{4}{3} - \frac{1}{3m} ,$$

e o conjunto T' forma um menor contra-exemplo para o teorema. Isto contradiz a suposição de minimalidade em n.

2º Caso: Considere-se agora o caso contrário ao primeiro, isto é, J_n é a única tarefa que termina no tempo B(I). Sabemos que:

$$(2) \qquad OPT(I) \geq \frac{1}{m} \sum_{i=1}^{n} \mu(J_i) , \qquad \text{e}$$

$$\sum_{i=1}^{n-1} \mu(J_i) \geq m(B(I) - \mu(J_n)),$$

como foi mostrado na análise do Algoritmo A, lembrando que aqui

$J_k = J_n$.

Portanto:

$$\frac{B(I)}{OPT(I)} = \frac{(B(I) - \mu(J_n)) + \mu(J_n)}{OPT(I)} \leqslant \frac{\mu(J_n)}{OPT(I)} + \frac{1}{m\,OPT(I)} \sum_{i=1}^{n-1} \mu(J_i)$$

$$= \frac{(m-1)\,\mu(J_n)}{m\,OPT(I)} + \frac{1}{m\,OPT(I)} \sum_{i=1}^{n} \mu(J_i)$$

de (2) temos:

$$\leqslant \frac{(m-1)\,\mu(J_n)}{m\,OPT(I)} + 1.$$

Visto que o conjunto de tarefas T contradiz (1),

$$1 + \frac{(m-1)\,\mu(J_n)}{m\,OPT(I)} \geqslant \frac{B(I)}{OPT(I)} > \frac{4}{3} - \frac{1}{3m},$$

$$\frac{(m-1)\,\mu(J_n)}{m\,OPT(I)} > \frac{1}{3} - \frac{1}{3m} = \frac{m-1}{3m}, \quad \text{e finalmente}$$

$$\mu(J_n) > \frac{OPT(I)}{3}$$

Portanto, como J_n é a tarefa com o menor tempo de processamento, conclui-se que para 3 tarefas arbitrárias J_i, J_j, J_k sempre vale $\mu(J_i) + \mu(J_j) + \mu(J_k) \geqslant 3\ \mu(J_n) > OPT(I)$. Isto implica que nenhum processador pode executar mais que 2 tarefas na schedule ótima para I.

Então, será mostrado que existe uma schedule ótima do seguinte tipo:

Os processadores são divididos em duas partes:

1) $P_1, \ldots, P_s$

2) $P_{s+1}, \ldots, P_m$

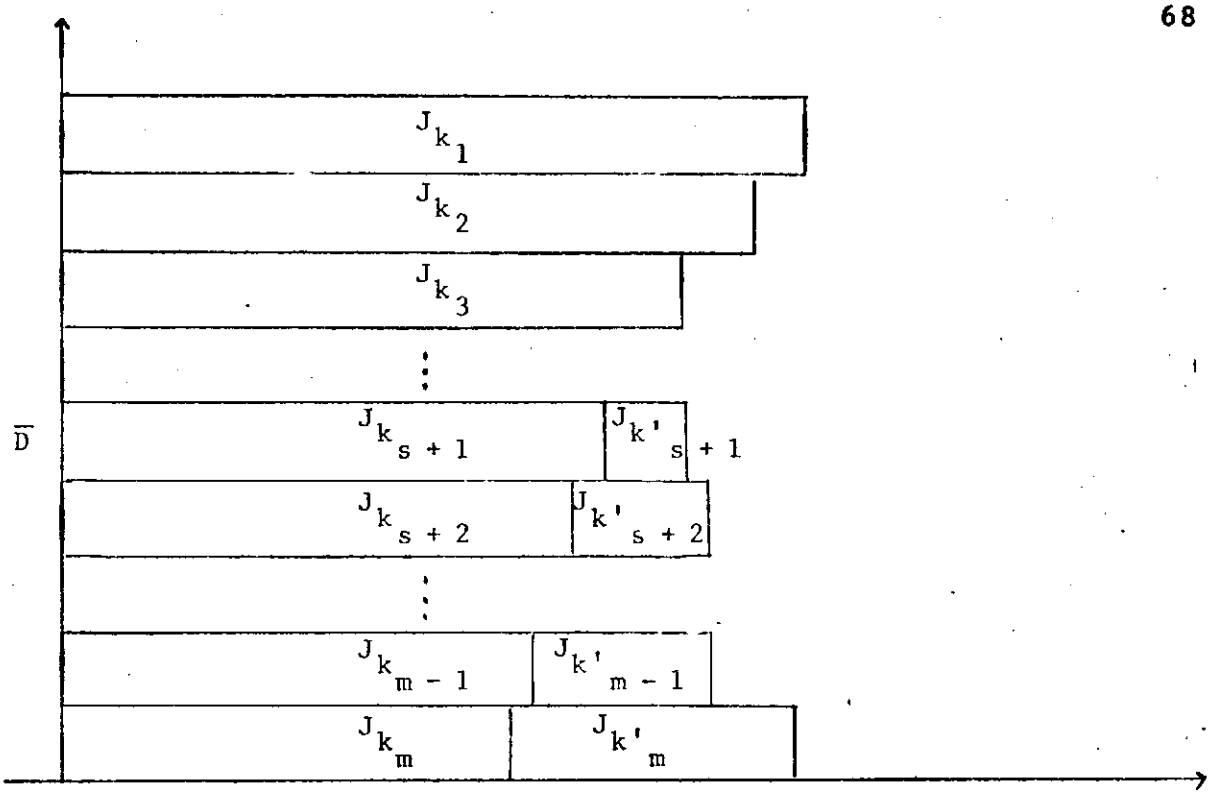
Os processadores $P_1, \ldots, P_s$ processarão somente uma tarefa e os processadores $P_{s+1}, \ldots, P_m$ processarão duas tarefas na schedule ótima sob consideração.

Além disso, se o processador P_i com $1 \leqslant i \leqslant s$ processa a tarefa J_{k_i} e o processador P_i com $s+1 \leqslant i \leqslant m$ processa primeiramente a tarefa J_{k_i} e depois a tarefa $J_{k'_i}$ temos que:

$$(3) \qquad \mu(J_{k_1}) \geqslant \ldots \geqslant \mu(J_{k_s}) \geqslant \mu(J_{k_{s+1}}) \geqslant \ldots \geqslant \mu(J_{k_m}) \geqslant$$

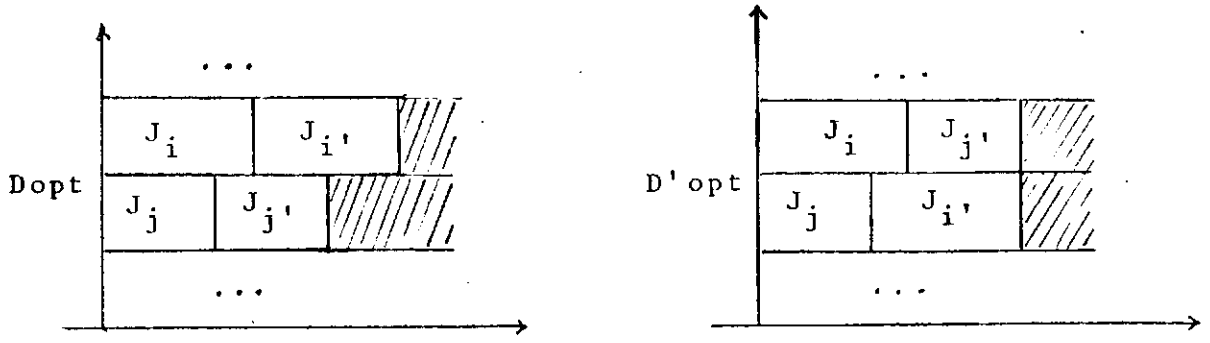
$$\mu(J_{K'_m}) \geqslant \ldots \geqslant \mu(j_{K'_{s+1}})$$

Uma possível configuração para (3), designada por $\overline{D}$, seria:

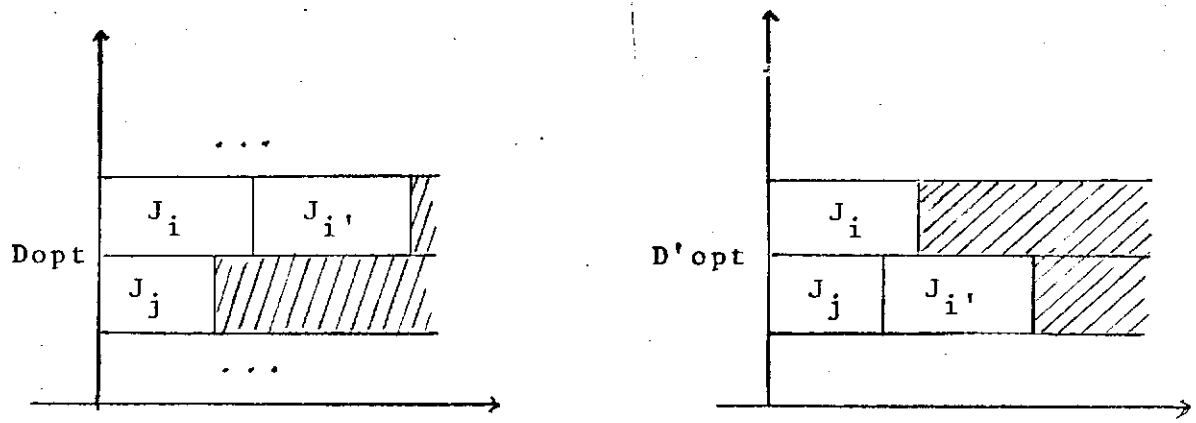


A schedule acima pode ser obtida de uma schedule arbitrária ótima, pelas seguintes mudanças (e, quando necessário, repetidas):

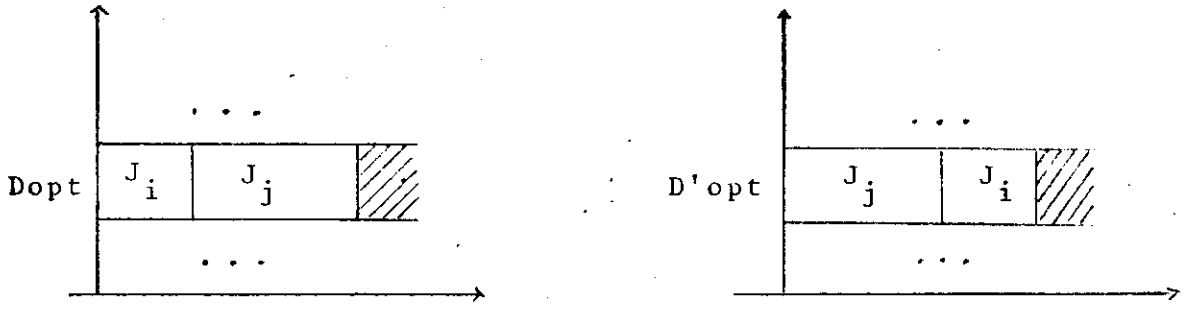
M_1: se $\mu(J_i) > \mu(J_j)$ e $\mu(J_{i'}) > \mu(J_{j'})$ na schedule arbitrária ótima, então pode-se trocar $J_{i'}$ e $J_{j'}$ de posições.



M_2: Se $\mu(J_i) > \mu(J_j)$, pode-se passar J_i, do processador P_i para o processador P_j



M_3: Se $\mu(J_i) < \mu(J_j)$, as tarefas podem ser permutadas



Em D'opt o novo Tempo de conclusão OPT'(I) satisfaz $OPT'(I) \leqslant OPT(I)$, isto é, as mudanças M_1, M_2 e M_3, se aplicadas em uma schedule arbitrária ótima, não afetam a sua optimalidade.

É necessário mostrar agora, fazendo estas mudanças, como foi possível obter (3):

a) Nos processadores de uma única tarefa, as mesmas podem ser processadas obedecendo a seguinte relação:

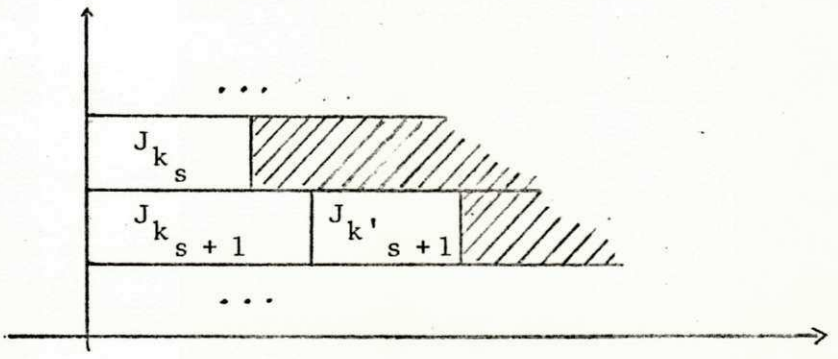
$$\mu(J_{k_1}) \geqslant \mu(J_{k_2}) \geqslant \ldots \geqslant \mu(J_{k_s})$$

b) No caso de processadores com 2 tarefas, as proces

sadas em primeiro lugar podem obedecer também a seguinte relação:

$$\mu(J_{k_{s+1}}) \geqslant \mu(J_{k_{s+2}}) \geqslant \ldots \geqslant \mu(J_{k_m})$$

c) É verdade que $\mu(J_{k_s}) \geqslant \mu(J_{k_{s+1}})$ pois em caso contrário ocorreria a seguinte configuração:

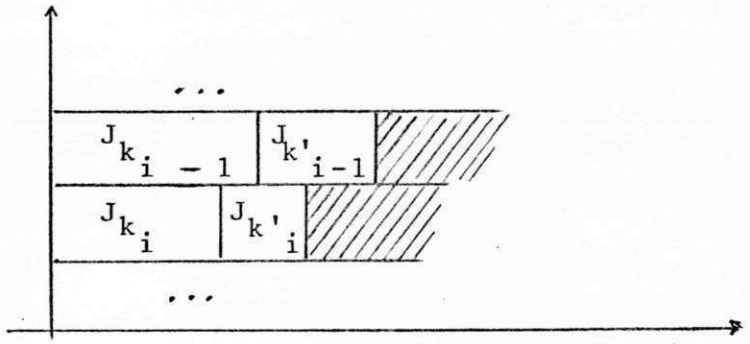


Esta configuração não é possível. Se a mesma ocorresse, ainda seria necessário fazer uma mudança M_2, isto é, passar a tarefa $J_{k'_{s+1}}$ para o processador P_s.

d) Também é verdade que $\mu(J_{k_m}) \geqslant \mu(J_{k'_m})$ pois, se ocorresse uma situação contrária, isto é, $\mu(J_{k_m}) < \mu(J_{k'_m})$, ainda seria possível uma mudança do tipo M_3.

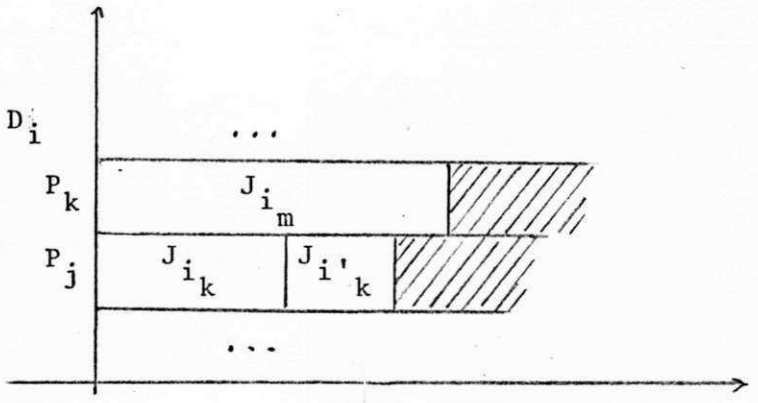
e) E, finalmente, pode-se afirmar que $\mu(J_{k'_m}) \geqslant \mu(J_{k'_{m-1}}) \geqslant \mu(J_{k'_{s+1}})$ pois, se esta dessigualdade fosse fal

sa, haveria a seguinte configuração:

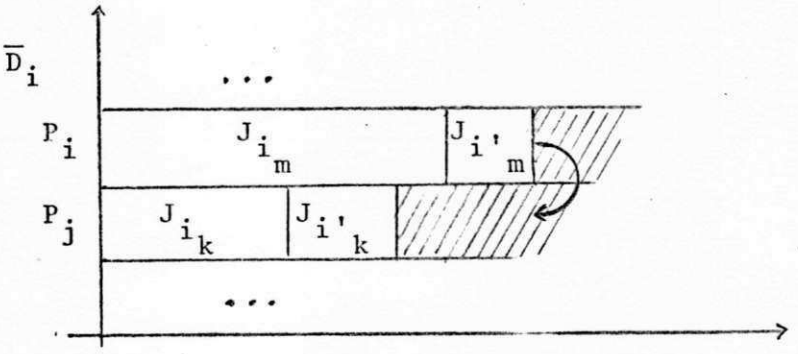


Esta situação não ocorreria, porque ainda é possível fazer uma mudança do tipo M_1, isto é, trocar as tarefas $J_{k'_{i-1}}$ e $J_{k'_i}$.

Visto que nenhuma das mudanças aplicadas em Dopt causou crescimento em OPT(I), pela optimalidade de OPT(I) o Tempo de conclusão de $\overline{D}$ deve ser também OPT(I). Mas note-se que $\overline{D}$ é similar ao diagrama D_b obtido pelo uso da lista $(J_1, \ldots, J_n)$, onde as tarefas foram organizadas em ordem não crescente em relação ao seu tempo de processamento. De fato, a única maneira em que D_b poderia diferir de $\overline{D}$, é na designação das tarefas $J_{i'_c}$ (segunda-camada). Especificamente, poderia ocorrer uma diferença, se para algum par J_{i_k}, $J_{i'_k}$ tivéssemos $\mu(J_{i_k}) + (J_{i'_k}) \leq \mu(J_{i_m})$ para algum $m < k$,



Neste caso, em D_B, $J_{i'_m}$ com comprimento $\mu(J_{i'_m})$, poderia ser designada para P_j ao invés de P_i. Contudo, se esta situação foi possível, então, em $\overline{D}$, poder-se-ia mover $J_{i'_m}$ de P_i para P_j.



Mas, obviamente, esta mudança não afeta OPT(I), causando uma contradição, visto que esta, agora, seria uma solução ótima com três tarefas designadas para um processador.

Portanto, conclui-se que B(I) = OPT(I). Mas isto contradiz a hipótese que B(I)/OPT(I) > 4/3 - 1/(3m), e a validade do limite dado pelo teorema é estabelecida.

Será mostrado através da construção de um exemplo do problema que este limite encontrado é o melhor possível, podendo com isto provar que o Teorema 2 dá a melhor garantia de desempenho possível para o Algoritmo B(LPT).

Exemplo 5.4.

$T = \{J_1, J_2, \ldots, J_{2m+1}\}$

$\mu(J_{2i-1}) = \mu(J_{2i}) = 2m - i, \quad 1 \leqslant i \leqslant m$

$\mu(J_{2m+1}) = m$

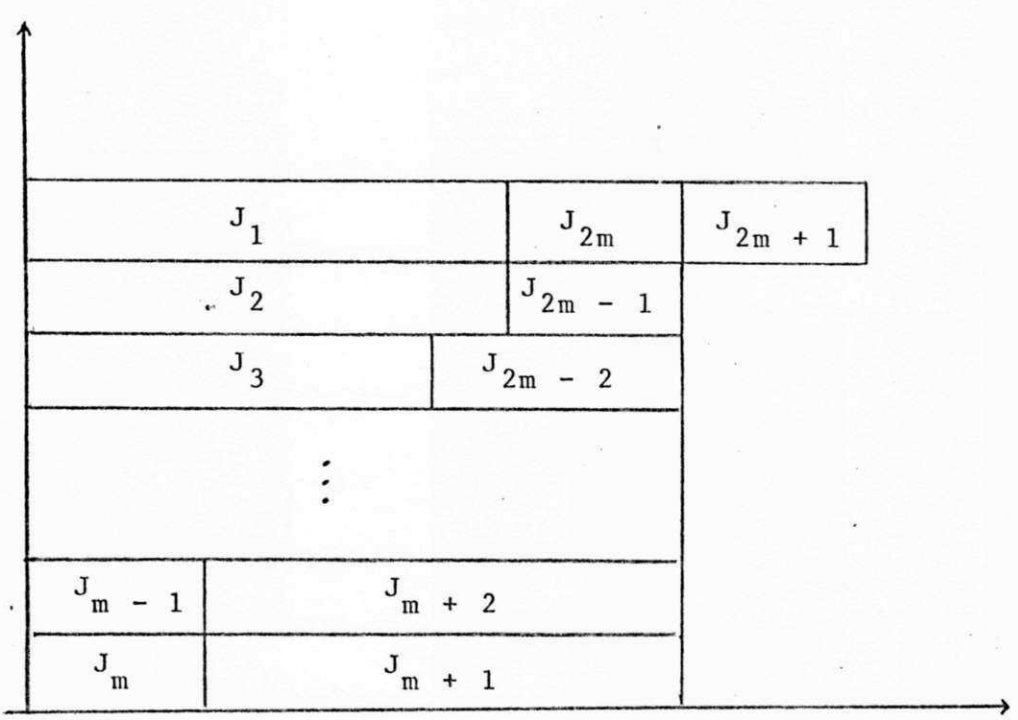


Fig. 5.6 - Schedule resultante da aplicação do Algoritmo B: $B(I) = 4m - 1$.

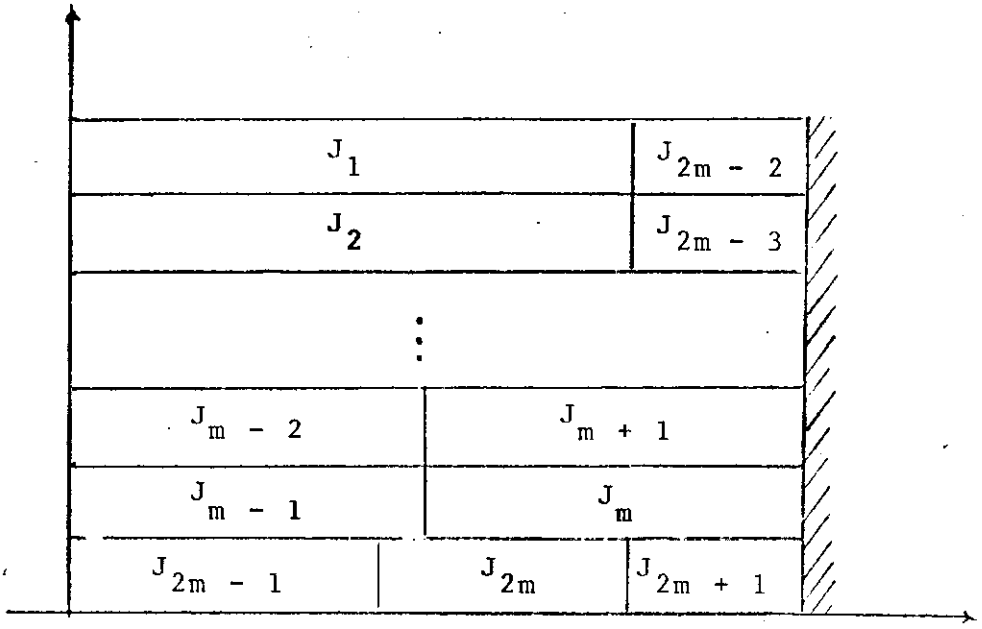


Fig. 5.7 - Schedule ótima: OPT(I) = 3m

O exemplo acima do problema produz $\frac{B(I)}{OPT(I)} = (\frac{4}{3} - \frac{1}{3m})$ para o Algoritmo B. Tem-se assim determinado o comportamento do pior caso para o algoritmo.

Aqui, se o exemplo acima for analisado, a fim de verificar se o algoritmo poderia ser melhorado, nenhuma idéia surge para tal fim. Isto acontece muitas vezes nas pesquisas para encontrar, cada vez mais, melhores algoritmos e, quando se depara com uma situação desta natureza, o melhor plano para atacar isto, é partir para investigações inteiramente novas.

Serão apresentados agora 2 exemplos comparativos aplicando os Algoritmos A e B. Os resultados foram encontrados com o uso do computador, sendo que as listagens dos programas para os referidos algoritmos encontram-se no Apêndice.

Exemplo 5.5

nº de tarefas: 25

Nº de processadores: 3

Tempos de processamento de cada tarefa (colocados entre parênteses):

1(5), 2(8), 3(2), 4(10), 5(1), 6(8), 7(7),8(15), 9(4), 10(13), 11(11), 12(1), 13(6), 14(2), 15(18), 16(5), 17(10), 18(2), 19(11), 20(1), 21(6), 22(5), 23(22), 24(3) e 25(4).

A aplicação do Algoritmo A deu como resultado o seguinte:

As tarefas que devem ser processadas no processador 1 são:

1, 5, 6, 9, 11, 15, 22 e 25.

No processador 2 são: 2, 7, 10, 13, 17, 20, 21 e 24.

No processador 3 são: 3, 4, 8, 12, 14, 16, 18, 19 e 23.

E o tempo total requerido para o processamento de todas as tarefas foi 70.

A aplicação do Algoritmo B(LPT) deu como resultado o seguinte:

Tarefas do processador 1 são: 23, 11, 6, 21, 16, 25, 24 e 12.

Tarefas do processador 2 são: 15, 19, 4, 7, 13, 9, 18 e 3.

Tarefas do processador 3 são: 8, 10, 17, 2, 22, 1, 14, 20 e 5.

O tempo total requerido foi 60.

Exemplo 5.6

nº de tarefas: 25

nº de processadores: 3

Tempos de processamento de cada tarefa (colocados entre parênteses):

1(3), 2(5), 3(4), 4(6), 5(2), 6(5), 7(5), 8(4), 9(6), 10(6), 11(2), 12(2), 13(3), 14(5), 15(4), 16(4), 17(4), 18(3), 19(3), 20(5), 21(5), 22(5), 23(2), 24(3), 25(4).

A aplicação do Algoritmo A resultou em:

Tarefas do processador 1 são: 1, 4, 8, 11, 12, 14, 18, 20 e 23.

Tarefas do processador 2 são: 2, 6, 9, 13, 16, 19, 22 e 25.

Tarefas do processador 3 são: 3, 5, 7, 10, 15, 17, 21 e 24.

O tempo total requerido foi 35.

A aplicação do Algoritmo B resultou em:

Tarefas do processador 1 são: 10, 22, 14, 2, 8, 19, 13 e 12.

Tarefas do processador 2 são: 9, 21, 7, 25, 16, 3, 1 e 11.

Tarefas do processador 3 são: 4, 20, 6, 17, 15, 24, 18, 23 e 5.

O tempo total requerido foi 34.

Pode-se notar com esses resultados que, quando a diferença dos tempos de processamento das tarefas é pequena, como ocorreu no 2º exemplo, a aplicação do Algoritmo A ou B não afetará muito o resultado. Mas, se esta diferença for grande, como ocorreu no 1º exemplo e a depender da ordem em que elas serão programadas no 1º algoritmo, o uso do 2º algoritmo é sempre mais vantajoso. Pode-se facilmente concluir isto, pelas respostas nos dois exemplos.

5.2 - PROCESSADORES NÃO IDÊNTICOS

Serão abordados inicialmente 4 algoritmos heurísticos para minimizar Tempo de conclusão com uma análise dos seus piores casos. Dentre esses procedimentos pode ser que um seja aparentemente muito simples, e até inferior, em relação aos demais, contudo existem exemplos para os quais um dá melhor resultado que o outro. Posteriormente, será apresentado um algo

ritmo para somente dois processadores a fim de mostrar que o comportamento do seu pior caso é melhor que qualquer um dos quatro apresentados inicialmente.

a) Algoritmos A, B, C e D para minimizar Tempo de conclusão

Algoritmo A:

Este algoritmo consiste basicamente de uma dada lista L com n tarefas J_i independentes, numa dada ordem, com seus respectivos tempos de processamento $\mu_j(J_i)$ $1 \leqslant j \leqslant m$, designar cada uma delas, uma de cada vez, para o processador que minimiza o seu Tempo de conclusão.

Para o Algoritmo A que será descrito agora, bem como para os algoritmos subsequentes, as variáveis, as entradas e saídas de dados têm os seguintes significados: A lista L_j ($1 \leqslant j \leqslant m$, onde m representa o número de processadores), consiste das tarefas que deverão ser processadas no j-ésimo processador; t_j representa o tempo corrente de conclusão do processador j na schedule parcial construída até esse ponto. E, finalmente, a entrada e saída dos dados como segue:

Entrada: A lista L composta pelas tarefas J_i com seus respectivos tempo de processamento $\mu_j(J_i)$, $1 \leqslant i \leqslant n$ e $1 \leqslant j \leqslant m$.

Saída: m listas L_j de tarefas, uma para cada máquina e A(I) que é o Tempo de conclusão da schedule.

Algoritmo A

Passo 1: (Inicialização)

for j ← 1 to m do

begin

$L_j \leftarrow \phi$;

$t_j \leftarrow 0$

end;

Passo 2: (Programação de cada tarefa, uma de cada vez, no processador que minimiza o seu tempo de conclusão)

for i ← 1 to n do

begin

Encontre j tal que $t_j + \mu_j(J_i) \leqslant t_\ell + \mu_\ell(J_i)$ para todo $1 \leqslant \ell \leqslant m$;

$L_j \leftarrow L_j \cup \{J_i\}$;

$t_j \leftarrow t_j + \mu_j(J_i)$;

$L \leftarrow L - \{J_i\}$

end;

Passo 3: (Cálculo do Tempo de conclusão da schedule construída)

$A(I) \leftarrow \max \{t_j / 1 \leqslant j \leqslant m\}$

Este algoritmo é um método heurístico bastante rápido para construção de possíveis schedules. O mesmo pode ser implementado em tempo proporcional a O(n), visto que os passos 1 e 3 tomam uma quantidade constante de tempo, enquanto o passo 2 toma O(n) quantidade de tempo para a programação das n tarefas, se m for fixado.

Exemplo 5.7. Neste exemplo são consideradas 7 tarefas para 3 processadores. O algoritmo começa designando a primeira tarefa para o processador P_1, porque este minimiza o seu tempo de conclusão:

Dados:

	$\mu_1(J_i)$	$\mu_2(J_i)$	$\mu_3(J_i)$
J_1	4	5	6
J_2	7	3	5
J_3	1	3	4
J_4	3	2	5
J_5	6	4	3
J_6	5	7	2
J_7	2	3	4

1ª iteração:

$$t_1 + \mu_1 (J_1) = 0 + 4 = \textcircled{4}$$

$$t_2 + \mu_2 (J_1) = 0 + 5 = 5$$

$$t_3 + \mu_3 (J_1) = 0 + 6 = 6$$

2ª iteração:

$$t_1 + \mu_1 (J_2) = 4 + 7 = 11$$

$$t_2 + \mu_2 (J_2) = 0 + 3 = \textcircled{3}$$

$$t_3 + \mu_3 (J_3) = 0 + 5 = 5$$

E assim por diante, até todas as tarefas serem processadas, chegando ao resultado final com o Tempo de conclusão igual a 9, conforme Figura 5.8.

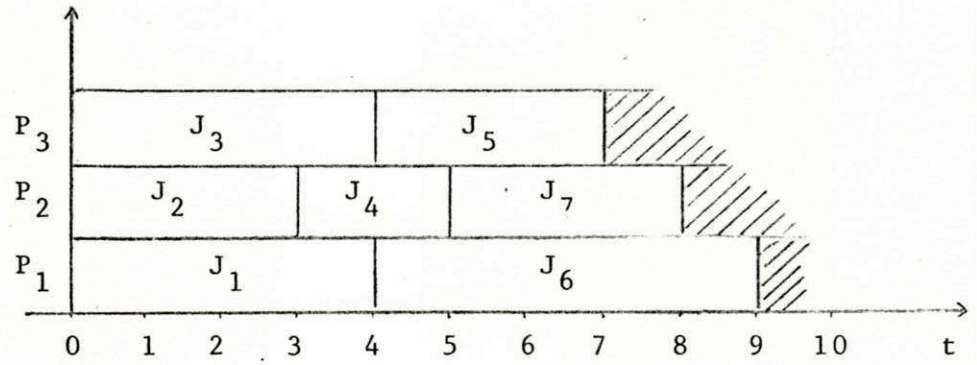


Fig. 5.8 - Schedule resultante da aplicação do Algoritmo A para o Ex. 5.7.

Algoritmo B:

Como cada tarefa J requer tempos diferentes para cada processador, esse algoritmo primeiramente lista as tarefas em ordem não crescente com relação ao menor tempo requerido em cada máquina, que é $\min_j \mu_j (J)$, e, finalmente, aplica o Algoritmo A para as tarefas reordenadas.

Algoritmo B

Passo 1: Construa uma lista das tarefas, em ordem não crescente em relação min $\{\mu_j (J) / 1 \leqslant j \leqslant m\}$

Passo 2: Aplique o Algoritmo A para o conjunto de tarefas reordenadas.

Este procedimento pode muitas vezes obter melhores resultados que o primeiro, não sendo, entretanto, regra geral. A complexidade deste algoritmo não é a mesma de A. A avaliação de $\mu_j(J)$ para todo J toma tempo $0(n)$, se m é fixo; a ordenação das tarefas toma $0(n \log n)$ e o 2º passo, que é a aplicação do Algoritmo A, toma $0(n)$, já visto anteriormente. Logo, pode-se concluir que a complexidade de B é $0(n \log n)$.

Para o Exemplo 5.7, exposto anteriormente, esse algoritmo dá um resultado melhor. Abaixo se encontram os dois quadros, o primeiro sendo o original com os mínimos marcados com asteriscos e o segundo com as tarefas reordenadas, prontas para a aplicação do Algoritmo A, com os resultados nos círculos.

	P_1	P_2	P_3
J_1	4	5	6
J_2	7	3*	5
J_3	1*	3	4
J_4	3	2*	5
J_5	6	4	3*
J_6	5	7	2*
J_7	2*	3	4

	P_1	P_2	P_3
J_1	①	3	4
J_2	3	②	5
J_3	5	7	2
J_4	②	3	4
J_5	7	③	5
J_6	6	4	③
J_7	④	5	6

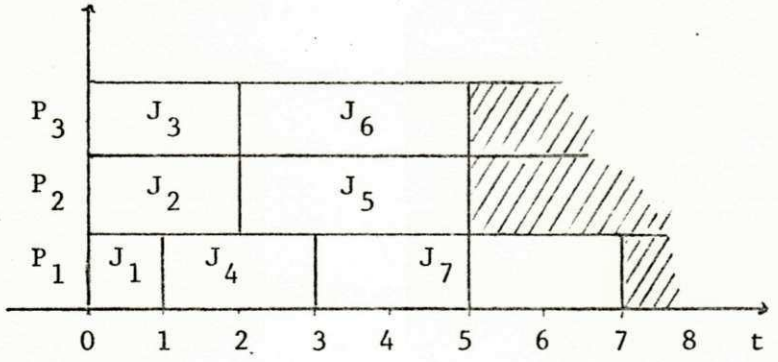


Fig. 5.9 - Schedule resultante da aplicação do Algoritmo B para o Exemplo 5.7.

Pode-se observar pela figura acima que B(I) = 7 é um resultado melhor que o anterior.

Algoritmo C:

Este algoritmo é diferente na maneira de trabalhar em relação aos anteriores. Após ter programado i tarefas, ele

programa uma tarefa (dentre as (n - i) tarefas restantes) que dá o Tempo de conclusão mínimo.

Algoritmo C

Passo 1: (Inicialização)

for j ← 1 to n do

begin

$t_j \leftarrow 0$;

$L_j \leftarrow \phi$

end.

Passo 2: (Todas as tarefas já foram programadas?)

if L = ϕ then

$C(I) \leftarrow \max \{t_j / 1 \leqslant j \leqslant m\}$; retorne;

Passo 3: (Programação das tarefas)

Encontre uma tarefa J e uma máquina j em L tal que $\min_j \{t_j + \mu_j(J)\} \leqslant \min_{j'} \{t_{j'} + \mu_{j'}(J')\}$ para todo J' em L;

$L_j \leftarrow L_j \cup \{J\}$;

$t_j \leftarrow t_j + \mu_j(J)$;

$L \leftarrow L - \{J\}$;

go to passo 2;

Para este algoritmo, a programação de uma tarefa, após i tarefas terem sido programadas, toma $O(n - i)$ quantidade de tempo. Assim, a ordem de complexidade do algoritmo é $O(n^2)$.

Algoritmo D:

A diferença deste algoritmo para C é no passo 3 que é substituído por:

Passo 3.

Encontre uma tarefa J e uma máquina j em L tal que $\max_j\{t_j + \mu_j(J)\} \leq \max_{j'}\{t_{j'} + \mu_{j'}(J)\}$ para todo J em L;

$L_j \leftarrow L_j \cup \{J\}$;

$t_j \leftarrow t_j$;

$L \leftarrow L - \{J\}$;

go to passo 2;

É obvio que a ordem de complexidade deste algoritmo é a mesma de C.

b) Análise dos piores casos para os Algoritmos A, B, C e D.

Teorema 3. (Ibarra, O. e Kim, C. (6))

Se I é um exemplo de um problema de scheduling de n tarefas independentes em processadores não idênticos que operam em paralelo, e se os algoritmos em questão são A, B, C e D acima, então:

$\frac{\alpha(I)}{OPT(I)} \leqslant m$, onde α = A, B, C ou D

Demonstração:

Seja $g(n) = \sum_{i=1}^{n} \min \{\mu_j(J_i)/1 \leqslant j \leqslant m\}$

Claramente $OPT(I) \geqslant \frac{1}{m} \cdot g(n)$

É provado por indução que $\alpha(I) \leqslant g(n)$ para α = A e C.

Trivialmente, $\alpha(1) \leqslant g(1)$. Assume-se que $\alpha(n-1) \leqslant g(n-1)$ e considere-se um conjunto T_n de n tarefas. Suponha-se que J_n seja a última tarefa programada pelo algoritmo α. Pela hipótese da indução $\alpha(n-1) \leqslant g(n-1)$ para o conjunto de $n-1$ tarefas $T_{n-1} = T_n - \{J_n\}$. Assim, para T_n o algoritmo proporciona $\alpha(n) \overset{①}{\leqslant} \alpha(n-1) + \min \{\mu_j(J_n)/1 \leqslant j \leqslant m\} \leqslant g(n-1) + \min \{\mu_j(J_n)/1 \leqslant j \leqslant m\} = g(n)$. A desigualdade ① tanto é verdadeira para o Algoritmo A como para D. Quando a última tarefa vai ser programada, o procedimento para ambos os casos é o mesmo. Isto se deve ao fato de que é necessário processar J_n no processador ℓ, tal que $t_\ell + \mu_\ell(J_n)$ seja o $\min\{t_j + \mu_j(J_n)/1 \leqslant j \leqslant m\}$.

Assim, $\alpha(I) \leqslant g(n)$. Como consequência $\frac{\alpha(I)}{OPT(I)} \leqslant m$.

Obviamente, este resultado vale para o Algoritmo B

que aplica A. Um argumento similimar à demonstração acima po de ser usado para o Algoritmo D.

Será mostrado, através da construção de exemplos do problema, que este limite encontrado é o melhor possível para todos os algoritmos, exceto C. Para este algoritmo, foi possível mostrar somente para m = 2, estando em aberto quando $m \geq 3$.

Exemplo 5.8: Considere-se o conjunto de n tarefas,

J_1 (u, u, ..., u),

J_2 (u, 2u - 1, ..., 2u - 1)

J_3 (2u, u, 3u - 2, ..., 3u - 2),

J_ℓ((ℓ - 1)u, (ℓ - 2)u, ..., 2u, u, ℓu - (ℓ - 1), ..., ℓu - (ℓ - 1)),

J_m (m - 1)u, (m - 2)u, ..., 2u, u, mu - (m - 1)).

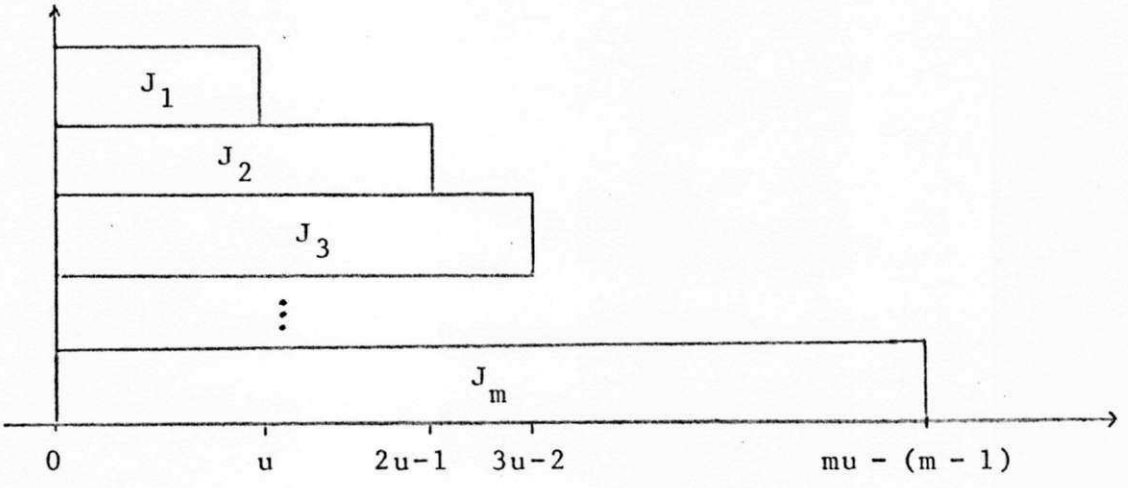


Fig. 5.10 - Schedule resultante da aplicação dos Algoritmos A e B. A(I) = B(I) = mu - (m - 1).

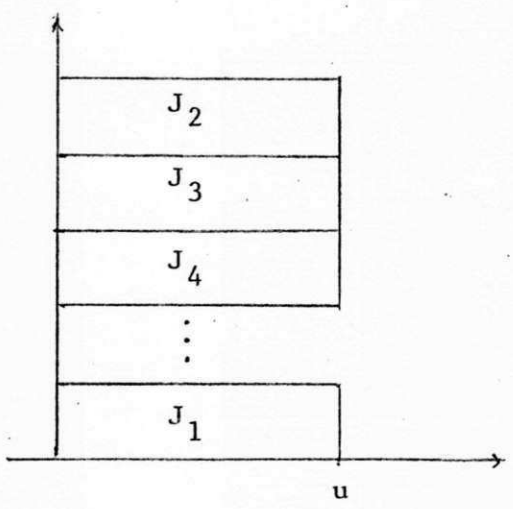


Fig. 5.11 - Schedule ótima: OPT(I) = u

Assim, $\frac{A(I)}{OPT(I)} = \frac{B(I)}{OPT(I)} = \frac{mu - (m - 1)}{u} = m - \frac{(m - 1)}{u}$

A expressão resultante m - (m - 1)/u tende a m, quando u tende a infinito, ficando assim provado que m é o melhor limite possível.

Através deste exemplo, pode-se garantir que o Teorema 3 dá a melhor garantia de desempenho possível para os Algoritmos A e B.

Exemplo 5.9: Considere-se o conjunto de (m - 1)m + 1 tarefas, onde u é bem maior que m,

$J_1(u, u, \ldots, u)$, $J_2(1,u,\ldots, u)$, $\ldots$, $J_m(1, 1, \ldots, u)$,

$J_{m+1}(u-1, u-1, \ldots, u-1)$, $J_{m+2}(1, u-1, \ldots, u-1)$, ...

$\vdots$ $J_{2m}(1, \ldots, 1, u-1)$,

$J_{(m-2)m+1}(u-(m-2), u-(m-2), \ldots, u-(m-2))$,

$J_{(m-2)m+2}(1, u-(m-2), \ldots, u-(m-2))$,

..., $J_{(m-1)m}(1, 1, \ldots, 1, u-(m-2))$.

$J_{(m-1)m+1}(u-(m-1), u(m-1), \ldots, u-(m-1))$.

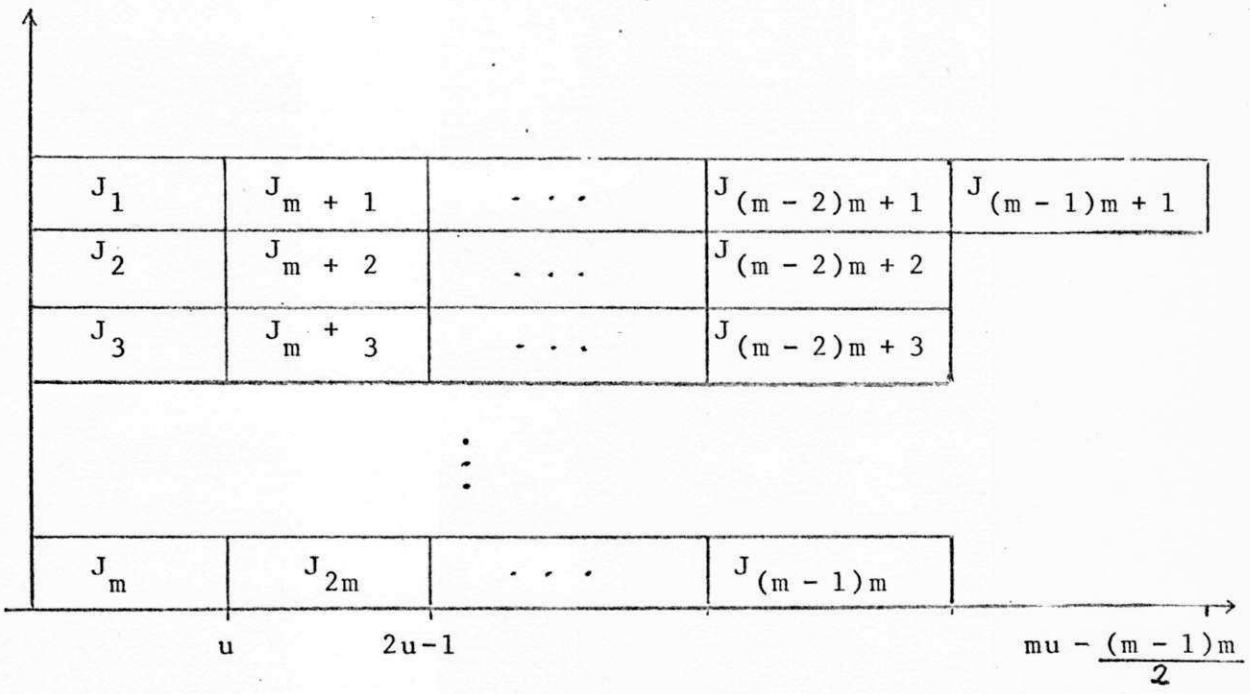


Fig. 5.12 - Schedule resultante da aplicação do Algoritmo D:
$D(I) = mu - \lceil (m-1)m \rceil / 2$.

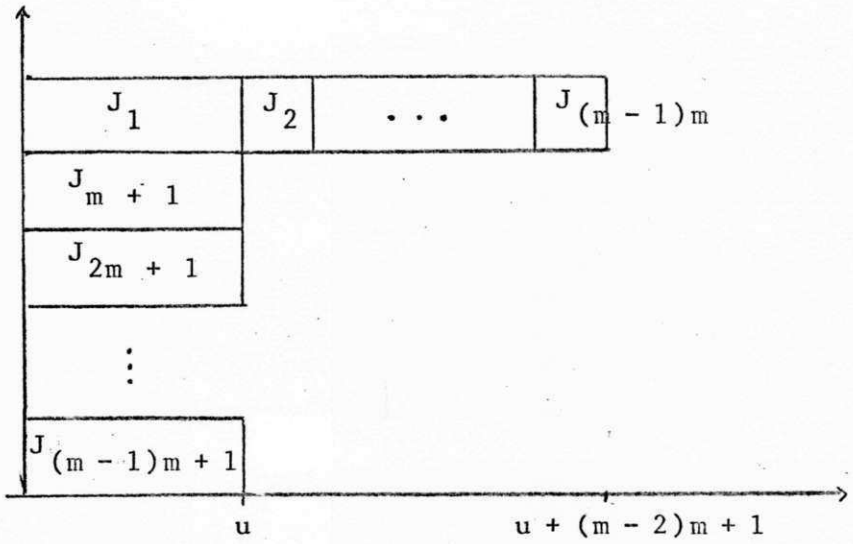


Fig. 5.13 - Schedule ótima: $OPT(I) = u + (m-2)m + 1$

Assim, $$\frac{D(I)}{OPT(I)} = [mu - m(m-1)/2]/[u + (m-2)m + 1]$$

$$= [m - m(m-1)/2u]/\{1 + [(m-2)m + 1]/u\}$$

Note-se que esta última expressão tende a m quando u tende a infinito, ficando assim provado que m é o melhor limite possível para o Algoritmo D.

Observação: Não foi possível mostrar que m é o melhor limite possível para o Algoritmo C, quando $m \geq 3$. De fato, não se conseguiu ainda nenhum exemplo que produza $C(I)/OPT(I) > 2$. Contudo, o exemplo dado por:

$I = \{J_1(u + 1, 2u), J_2(u, u)\}$ dá $OPT(I) = u + 1$ e

C(I) = 2u, mostra que o limite para o Algoritmo C é o melhor possível para m = 2.

c) Algoritmo E para minimizar Tempo de conclusão em 2 processadores.

Foi mostrado anteriormente que todos os algoritmos, exceto C, têm em relação ao pior-caso um limite m para a razão $\alpha(I)/OPT(I)$, sendo também este limite o melhor possível. Para o Algoritmo C, foi somente possível mostrar que a razão m é aproximável para m = 2. Agoram será apresentado um algoritmo heurístico para um sistema com 2 processadores, que tem um comportamento do pior caso melhor que qualquer um dos algoritmos anteriores. Somente não foi ainda possível estender a idéia usada no algoritmo para o caso de $m \geqslant 3$.

O algoritmo começa programando temperariamente as tarefas de maneira que cada uma seja designada para o processador onde a tarefa tem um tempo de processamento menor. Se os tempo de término dos processadores são iguais, obviamente a schedule é ótima. Contudo, se um processador fica ocioso enquanto o outro não está, então a schedule pode não ser ótima. Sem perda de generalidade, assume-se que P_1 tenha o maior tempo de término. Então pode ser possível reduzir este tempo através da passagem de algumas tarefas de P_1 para o processador P_2. A idéia é decrescer o tempo de conclusão de P_1 tanto quanto possível, enquanto se minimiza o crescimento do tempo de con

clusão em P_2. Assim, se tarefas J_1 e J_2 em P_1 são tais que $\mu_1(J_1)/\mu_2(J_1) \geqslant \mu_1(J_2)/\mu_2(J_2)$, então J_1 deve ser melhor candidata que J_2 para ser trocada de processador. O algoritmo então lista as tarefas em P_1 nesta ordem e posteriormente muda-as para P_2, enquanto o tempo de conclusão puder ser reduzido.

Abaixo está a descrição formal do algoritmo:

Algoritmo E:

Passo 1: (Inicialização)

$L_1 \leftarrow \phi$; $L_2 \leftarrow \phi$

$t_1 \leftarrow 0$; $t_2 \leftarrow 0$;

Passo 2: (Programação de cada tarefa no processador em que ela requer menor tempo de processamento e verificação do processador que terminou por último)

for i ← 1 to n do

if $\mu_1(J_i) \leqslant \mu_2(J_i)$ then

begin

$L_1 \leftarrow L_1 \cup \{J_i\}$;

$t_1 \leftarrow t_1 + \mu(J_i)$,

end

else

begin

$L_2 \leftarrow L_2 \cup \{J_i\}$;

$t_2 \leftarrow t_2 + \mu(J_i)$

end;

if $t_1 = t_2$ then $E(I) \leftarrow t_1$; retorne

if $t_1 > t_2$ then

begin

$\alpha \leftarrow 1$; $\beta \leftarrow 2$;

end

else

begin

$\alpha \leftarrow 2$; $\beta \leftarrow 1$;

end;

Passo 3: ($t_\alpha > t_\beta$; ordenação de L e programação de algumas tarefas de L_α em L_β).

Ordene L_α de acordo com: $\mu_\alpha(J_i)/\mu_\beta(J_i) \leqslant \mu_\alpha(J_{i+1})/\mu_\beta(J_{i+1})$ para $1 \leqslant i \leqslant k - 1$;

for i $\leftarrow$ k to 1 step - 1 do

if $t_\alpha > t_\beta + \mu_\beta(J_i)$ then

begin

$L_\alpha \leftarrow L_\alpha - \{J_i\}$; $t_\alpha \leftarrow t_\alpha - \mu_\alpha(J_i)$;

$L_\beta \leftarrow L_\beta \cup \{J_i\}$; $t_\beta \leftarrow t_\beta + \mu_\beta(J_i)$;

end;

if $t_\alpha \geqslant t_\beta$ then $E(I) \leftarrow t_\alpha$; retorne

else $E(I) \leftarrow t\beta$; retorne.

Para encontrar a ordem de complexidade deste algoritmo será analisada a necessidade de cada passo. O passo 1 toma uma quantidade constante de tempo, enquanto o passo 2 toma $0(n)$. Para o passo 3, seja $|L_\alpha| = r \leqslant n$. Então o passo 3.1 e o passo 3.2 tomam, respectivamente, $0(r \log r)$ e $0(r)$ quantidades de tempo. O passo 3.3 toma uma quantidade constante de tempo. Logo, o algoritmo trabalha em $0(n) + 0(r \log n) + 0(r) \leqslant 0(n \log n)$ que é, então, a ordem de complexidade de todo o algoritmo.

Pode ser notado que o Algoritmo E difere dos anteriores na maneira como procede, mas é também muito simples. Será apresentado aqui um exemplo, cujo resultado foi encontrado com o uso do computador. A listagem do programa para o referido algoritmo encontra-se no Apêndice .

Exemplo 5.10

Nº de tarefas = 30

Relação das tarefas com os seus respectivos tempos de processamento, nos processadores 1 e 2.

	$\mu_1(J_i)$	$\mu_2(J_i)$		$\mu_1(J_i)$	$\mu_2(J_i)$		$\mu_1(J_i)$	$\mu_2(J_i)$
J_1	2	5	J_{11}	5	2	J_{21}	1	7
J_2	3	2	J_{12}	3	3	J_{22}	9	6
J_3	4	4	J_{13}	1	5	J_{23}	3	3
J_4	5	1	J_{14}	2	8	J_{24}	3	6
J_5	5	7	J_{15}	3	9	J_{25}	7	1
J_6	6	2	J_{16}	7	3	J_{26}	6	6
J_7	3	5	J_{17}	2	4	J_{27}	4	4
J_8	1	4	J_{18}	3	6	J_{28}	2	10
J_9	2	8	J_{19}	1	4	J_{29}	9	6
J_{10}	6	7	J_{20}	3	10	J_{30}	2	8

O resultado encontrado foi:

Tarefas para o processador 1: 10, 5, 7, 17, 18, 24, 1, 15, 20, 8, 9, 14, 19, 30, 13, 28 e 21

Tarefas para o processador 2: 27, 26, 23, 12, 3, 29,

25, 22, 16, 11, 6, 4 e 2.

Tempo total requerido para o processamento de todas tarefas foi igual a 43.

d) Análise do pior caso para o Algoritmo E

Para o Algoritmo E, será apenas anunciado o teorema que garante um limite para a razão E(I)/OPT(I), sem demonstrá-lo, podendo a demonstração ser encontrada em Ibarra, O. e kim, E. (6).

Teorema 4

Se I é um exemplo de problema de scheduling de tarefas independentes em 2 processadores não idênticos que operam em paralelo, e se o algoritmo em questão é o Algoritmo E, então:

$$\frac{E(I)}{OPT(I)} \leqslant \frac{\sqrt{5}+1}{2}$$

Esta razão é um resultado melhor que os anteriores, pois para os primeiros algoritmos $\alpha(I)/OPT(I) \leqslant 2$, com $\alpha = A$, B, C ou D e neste caso $E(I)/OPT(I) \leqslant 1{,}6180 \ldots$

Resta finalmente mostrar através de um exemplo, que

esta razão de desempenho é a melhor possível para o Algoritmo E.

Exemplo 5.11.

Considere-se o conjunto T de tarefas:

$$T = \{J_1(u,\ [(\sqrt{5}+1)/2]\,u),\ J_2([(\sqrt{5}+1)/2]u,\ [(\sqrt{5}+3)/2]u-1)\}$$

O Algoritmo E produz $L_1 = \{J_1\}$ e $L_2 = \{J_2\}$. Portanto $E(I) = [(\sqrt{5}+3)/2]u - 1$. A schedule ótima teria $L_1 = \{J_2\}$ e $L_2 = \{J_1\}$, dando $OPT(I) = [(\sqrt{5}+1)/2]u$, conforme figura abaixo:

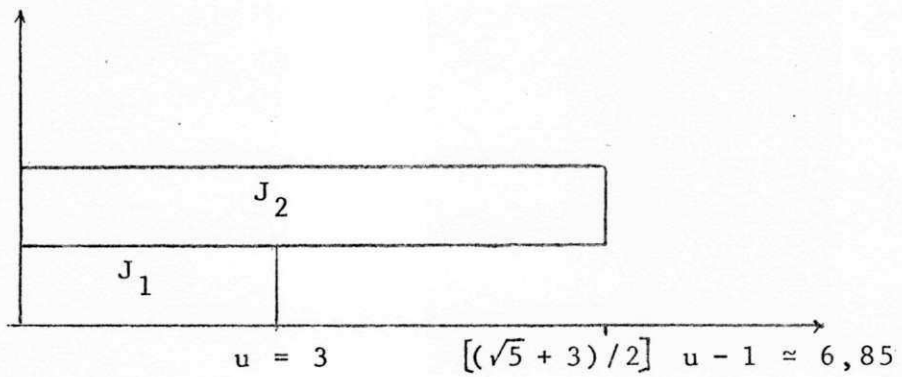


Fig. 5.14 - Schedule resultante da aplicação do Algoritmo E em T, com u = 3.

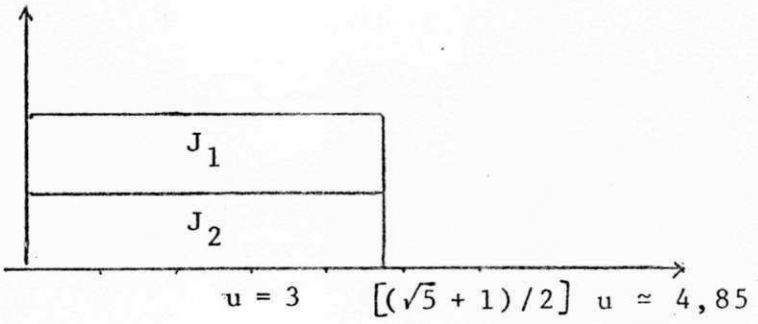


Fig. 5.15 - Schedule ótima OPT(I) ≃ 4,85.

Assim, $\frac{E(I)}{OPT(I)} = (\frac{\sqrt{5}+3}{2}.u - 1)/ (\frac{\sqrt{5}+1}{2}. u) = \frac{\sqrt{5}+1}{2} - \frac{\sqrt{5}-1}{2u}$

que tende a $(\sqrt{5} + 1)/2$, quando u tende a infinito.

Ficando assim provado que $(\sqrt{5} + 1)/2$ é o melhor limite possível.

CAPÍTULO VI

CONSIDERAÇÕES FINAIS

Nestes últimos anos houve um crescente interesse dos cientistas no estudo de scheduling, acarretando muitas publicações a respeito, publicações estas que se encontram, contudo, fragmentadas em jornais, periódicos, revistas cientificas etc. Apesar disto, atualmente existem poucos livros sobre scheduling e, desses poucos, muito se deixa a dizer em relação aos recentes resultados da área; inclusive em relação à análise dos algoritmos de aproximação através de seus piores casos.

Este trabalho procurou reunir o que há de mais importante nos estudos de scheduling relativo ao modelo básico de

único processaor e processadores paralelos, no caso de tarefas independentes. Pela complexidade com que normalmente se apresentam os problemas nesta área, os procedimentos para resolução dos mesmos foram de aproximação ou heurísticos. Deu-se maior importância aos procedimentos que pudessem ser avaliados através da análise do seu pior caso. Esses procedimentos foram uniformizados em suas linguagens, escritos de uma forma estruturada, visando facilitar a compreensão do leitor.

Foram introduzidos no Capítulo II os conceitos fundamentais relativos aos modelos apresentados em todo o trabalho ou seja, o problema foi identificado através de exemplos ilustrativos, que apresentaram diferentes situações de problemas. Para tal fim, foram definidas as variáveis e funções objetivas mais usadas no estudo de scheduling.

No Capítulo III procurou-se justificar o uso de procedimentos heurísticos em problemas de scheduling, mostrando os resultados encontrados até o momento, no que se refere à complexidade desses problemas. Quando classificados em PS (conjunto de problemas solucionáveis polinomialmente por algoritmos exatos) e NP (conjunto de problemas que não podem ser solucionáveis polinomialmente por algoritmo exatos), ficou claro a importância dos procedimentos heurísticos para a resolução desses problemas, visto que a grande maioria está na classe NP. Na segunda parte deste Capítulo procurou-se mostrar a importância da avaliação do desempenho de um algoritmo heurístico, visto que,

obtida uma solução heuristicamente, é importante detectar até que ponto ela é aceitável. Foram enfatizadas as vantagens e desvantagens ao se avaliar um algoritmo através da análise do pior caso, onde ficou constatada a sua superioridade em relação a outros métodos de avaliação.

Um importante algoritmo heurístico para minimizar a média dos atrasos em uma schedule, para o caso de único processador, foi apresentado no Capítulo IV. Este algoritmo usa as propriedades resultantes da permutação de pares de tarefas adjacentes em uma schedule.

No Capítulo V todos os algoritmos apresentados têm como objetivo a minimização do Tempo de conclusão de uma schedule. Para todos eles é feita a avaliação de desempenho através da análise do pior caso. No caso de processadores paralelos idênticos foram visto dois algoritmos. No primeiro, as tarefas para serem programadas não se encontram em uma ordenação pré-estabelecida em relação aos seus tempos de processamento, como acontece no segundo. Isto faz com que os resultados obtidos pelo segundo algoritmo sejam, na maioria das vezes, melhores que os obtidos pelo primeiro. Para os processadores paralelos não idênticos, os quatro primeiros algoritmos vistos apresentam poucas diferenças, mas, dependendo do problema, um dá melhor resultado que outro. Esta semelhança pode ser notada pela própria análise do pior caso, que dá um mesmo limite para a razão de desempenho dos algoritmos. Um limite melhor pa

ra a razão de desempenho foi encontrado para o último algoritmo, embora o mesmo seja específico a dois processadores ($m = 2$), não se conseguindo, até o momento, extender esse resultado para $m \neq 2$.

APÊNDICE

```
0001 1-    BEGIN  COMMENT  REGINA CELIA BAIBINO FIGUEIRA LISBOA
0001 --    ***************************************************************************
0001 --    ***************************************************************************
0001 --      ESTE PROGRAMA E UM PROCEDIMENTO HEURISTICO PARA MINIMIZAR A MEDIA DOS
0001 --    ATRASOS EM UM PROBLEMA DE SCHEDULING DE UNICO PROCESSADOR ONDE AS TAREFA
0001 --    S SAO INDEPENDENTES.
0001 --    ***************************************************************************
0001 --    ***************************************************************************
0001 --      INTEGER A. B. T. CONT. PIVOT. AUX. AUX1. N
0002 --      INTFIELDSIZE =3
0003 --      READ(N)
0004 2-      BEGIN
0005 --        RECORD LISTA (INTEGER JOB  REFERENCE (LISTA) LINK)
0007 --        REFERENCE (LISTA) APONT. LIVRE. APONT1. APONT2
0008 --        INTEGER ARRAY TPRO. DUE. AUXILIAR (1 .. N)
0009 --        APONT =LIVRE =NULL
0010 --        FOR I =1 UNTIL N DO
0010 3-          BEGIN
0011 --            APONT1 =LISTA
0012 --            READON(JOB(APONT1))
0014 --            LINK(APONT1) =APONT
0014 --            APONT =APONT1
0015 -3          END
0016 --        FOR I =1 UNTIL N DO
0016 --          READON (TPRO(I). DUE(I))
0017 --        T =CONT =0
0018 --        AUX =TPRO(JOB(LINK(APONT)))
0019 --        IF AUX)TPRO(JOB(APONT)) THEN
0019 3-          BEGIN
0020 --            AUX =TPRO(JOB(APONT))
0021 --            CONT =1
0022 -3          END
0023 --        IF (AUX)=DUE(JOB(LINK(APONT)))) OR (CONT=0) THEN
0023 3-          BEGIN
0024 --            LIVRE =APONT
0025 --            APONT =LINK(APONT)
0026 -3          END
0026 --        ELSE
0027 3-          BEGIN
0028 --            LIVRE =LINK(APONT)
0029 --            LINK(APONT) =LINK(LINK(APONT))
0030 -3          END
0031 --        LINK(LIVRE) =NULL
0032 --        T =T+TPRO(JOB(LIVRE))
0033 --        PIVOT =JOB(APONT)
0034 --        APONT =LINK(APONT)
0035 --        WHILE APONT =NULL DO
0035 3-          BEGIN
0036 --            AUX =TPRO(PIVOT)
0037 --            IF AUX)TPRO(JOB(APONT)) THEN
0037 --              AUX =TPRO(JOB(APONT))
0038 --            AUX1 =DUE(PIVOT)
0039 --            IF AUX1)DUE(JOB(APONT)) THEN
0039 --              AUX1 =DUE(JOB(APONT))
0040 --            IF (T+AUX)=AUX1) OR (TPRO(PIVOT))=TPRO(JOB(APONT))) THEN
0040 4-              BEGIN
0041 --                APONT1 =LISTA
0042 --                JOB(APONT1) =PIVOT
```

```
0043 --                    LINK(APUNT1) =LIVRE
0044 --                    LIVRE =APUNT1
0045 --                    PIVOT =JOB(APUNT)
0046 --                    APUNT =LINK(APUNT)
0047 --                    T =T+TPRO(JOB(LIVRE))
0048 -4                  END
0048 --                ELSE
0049 4-                  BEGIN
0050 --                    AUX =PIVOT
0051 --                    PIVOT =JOB(APUNT)
0052 --                    JOB(APUNT) =AUX
0053 --          K         AUX =TPRO(PIVOT)
0054 --                    IF AUX)TPRO(JOB(LIVRE)) THEN
0054 --                      AUX =TPRO(JOB(LIVRE))
0055 --                    AUX1 =DUE(PIVOT)
0056 --                    IF AUX1)DUE(JOB(LIVRE)) THEN
0056 --                      AUX1 =DUE(JOB(LIVRE))
0057 --                    IF (T-TPRO(JOB(LIVRE))+AUX)=AUX1) OR (TPRO(JOB(LIVRE)))=
0057 --                                                          TPRO(PIVOT)) THEN
0057 5-                     BEGIN
0058 --                       APUNT1 =LISTA
0059 --                       JOB(APUNT1) =PIVOT
0060 --                       LINK(APUNT1) =LIVRE
0061 --                       LIVRE =APUNT1
0062 --                       PIVOT =JOB(APUNT)
0063 --                       APUNT =LINK(APUNT)
0064 --                       T =T+TPRO(JOB(LIVRE))
0065 -5                     END
0065 --                   ELSE
0066 5-                     BEGIN
0067 --                       T =T-TPRO(JOB(LIVRE))
0068 --                       IF DUE(JOB(LIVRE)))=DUE(JOB(APUNT)) THEN
0068 6-                         BEGIN
0069 --                           APUNT1 =APUNT
0070 --                           APUNT =LIVRE
0071 --                           LIVRE =LINK(LIVRE)
0072 --                           LINK(APUNT) =APUNT1
0073 -6                         END
0073 --                       ELSE
0074 6-                         BEGIN
0075 --                           APUNT1 =APUNT
0076 --                           WHILE (APUNT1 =NULL) AND (DUE(JOB(LIVRE))
0076 --                                                     DUE(JOB(APUNT1))) DO
0076 7-                             BEGIN
0077 --                               APUNT2 =APUNT1
0078 --                               APUNT1 =LINK(APUNT1)
0079 -7                             END
0080 --                           LINK(APUNT2) =LIVRE
0081 --                           LIVRE =LINK(LIVRE)
0082 --                           LINK(LINK(APUNT2)) =APUNT1
0083 -6                         END
0084 --                       IF LIVRE=NULL THEN
0084 6-                         BEGIN
0085 --                           APUNT1 =LISTA
0086 --                           JOB(APUNT1) =PIVOT
0087 --                           LIVRE =APUNT1
0088 --                           LINK(LIVRE) =NULL
0089 --                           T =TPRO(JOB(LIVRE))
```

```
0090 --                          PIVOT =JOB[APONT]
0091 --                          APONT =LINK[APONT]
0092 -6                        END
0092 --                      ELSE
0093 --                        GO TO K
0094 -5                    END
0095 -4                  END
0096 -3                END
0097 --              FOR J =N-1 STEP -1 UNTIL 1 DO
0097 3-                BEGIN
0098 --                  AUXILIAR[J] =JOB[LIVRE]
0099 --                  LIVRE =LINK[LIVRE]
0100 -3                END
0101 --              AUXILIAR[N] =PIVOT
0102 --              WRITE( 'NUMERO DE TAREFAS= ', N)
0103 --              WRITE(    )
0104 --              WRITE( ABAIXO ESTAO AS TAREFAS COM SEUS RESPECTIVOS TEMPO DE PROCESS
0104 --         AMENTO E TEMPO PREVISTO PARA A SUA CONCLUSAO COLOCADOS ENTR PARENTESES )
0104 --              WRITE(    )
0106 --              FOR I =1 UNTIL N DO
0106 --                WRITE(I,  ( , TPRO[I], DUE[I],  ) )
0107 --              WRITE(    )    WRITE(    )
0108 --              WRITE( ABAIXO ESTA A SEQUENCIA EM QUE AS TAREFAS DEVEM SER PROCESSAD
0109 --         AS PARA MINIMIZAR A MEDIA DOS ATRASOS )
0110 --              WRITE(    )
0111 --              WRITE(AUXILIAR[1])
0112 --              FOR J =2 UNTIL N DO
0112 --                WRITEON(AUXILIAR[J])
0113 -2          END
0114 -1        END.

EXECUTION OPTIONS   DEBUG.1   TIME=5 SECONDS PAGES=5

002.30 SECONDS IN COMPILATION. (03184. 01704) BYTES OF CODE GENERATED
```

```
0001 --    ************************************************************************
0001 --    ************************************************************************
0001 --      ESTE PROGRAMA E UM PROCEDIMENTO HEURISTICO PARA MINIMIZAR O TEMPO
0001 --         TOTAL REQUERIDO PARA PROCESSAMENTO DE TAREFAS INDEPENDENTES EM
0001 --         PROCESSADORES IDENTICOS QUE OPERAM EM PARALELO.
0001 --      ESTE PROCEDIMENTO NAO TEM RESTRICOES COM RELACAO A ORDEM DAS TARE-
0001 --         FAS. ESTAS PODEM SER APRESENTADAS EM QUALQUER ORDEM DADA.
0001 --    ************************************************************************
0001 --    ************************************************************************
0001 --      INTEGER M. N. I. AUX. AUX1
0002 --      INTEGERSIZE =3
0003 --      READ (M.N)
0004 2-      BEGIN
0005 --        RECORD LISTA (INTEGER JOB  INTEGER TPRO  REFERENCE (LISTA) LINK)
0008 --        REFERENCE (LISTA) APONT. APONT1. NOVO1
0009 --        REFERENCE (LISTA) ARRAY NOVO(1  M)
0010 --        INTEGER ARRAY T(1  M)
0011 --        WRITE( NUMERO DE TAREFAS = . N)
0012 --        WRITE( NUMERO DE PROCESSADORES = . M)
0013 --        WRITE(     )
0014 --        APONT =NULL
0015 --        FOR I =1 UNTIL N DO
0015 3-          BEGIN
0016 --            APONT1 =LISTA
0017 --            READON (JOB(APONT1). TPRO(APONT1))
0018 --            IF APONT=NULL THEN
0018 4-              BEGIN
0019 --                LINK(APONT1) =NULL
0021 --                APONT =APONT1
0021 -4              END
0021 --            ELSE
0022 4-              BEGIN
0023 --                LINK(APONT1) =APONT
0024 --                APONT =APONT1
0025 -4              END
0026 -3          END
0027 --        WRITE( AS TAREFAS COM SEUS RESPECTIVOS TEMPOS DE PROCESSAMENTO COLOC
0027 --      ADOS ENTRE PARENTESES SAO  )
0028 --        FOR I =1 UNTIL M DO
0028 --          NOVO(I) =NULL
0029 --        FOR I =1 UNTIL M DO
0029 3-          BEGIN
0030 --            WRITEON(JOB(APONT).    ( . TPRO(APONT). ).       )
0031 --            T(I) =TPRO(APONT)
0032 --            NOVO(I) =APONT
0033 --            APONT =LINK(APONT)
0034 --            LINK(NOVO(I)) =NULL
0035 -3          END
0036 --        WHILE APONT = NULL DO
0036 3-          BEGIN
0037 --            AUX =1000
0038 --            FOR I =1 UNTIL M DO
0038 --              IF T(I)>AUX THEN
0038 4-                BEGIN
0039 --                  AUX =T(I)
0040 --                  AUX1 =I
0041 -4                END
```

```
0042 --                T(AUX1) =T(AUX1)+TPRO(APONT)
0043 --                WRITEON(JOB(APONT).   ( . TPRO(APONT).  ).       )
0044 --                NOVO1 =LINK(APONT)
0045 --                LINK(APONT) =NOVO(AUX1)
0046 --                NOVO(AUX1) =APONT
0047 --                APONT =NOVO1
0048 -3              END
0049 --            AUX =0
0050 --            FOR I =1 UNTIL M DO
0050 --              IF T(I)  AUX THEN
0051 --                  AUX =T(I)
0051 --                WRITE(     )
0052 --                WRITE(     )
0053 --            WRITE( OBSERVACAO  AS TAREFAS ESTAO AQUI LISTADAS EM ORDEM INVERSA
0053 --        EM RELACAO AO SEU PROCESSAMENTO )
0054 --            WRITE(     )
0055 --            FOR I =1 UNTIL M DO
0055 3-              BEGIN
0056 --              WRITE( AS TAREFAS DO PROCESSADOR .I. SAO )
0057 --              NOVO1 =NOVO(I)
0058 --              WHILE NOVO1  =NULL DO
0058 4-                BEGIN
0059 --                  WRITEON(JOB(NOVO1))
0060 --                  NOVO1 =LINK(NOVO1)
0061 -4                END
0062 --                WRITE(     )
0063 -3              END
0064 --            WRITE(     )
0065 --            WRITE ( O TEMPO TOTAL REQUERIDO PARA O PROCESSAMENTO DE TODAS AS TAR
0065 --        EFAS E IGUAL A . AUX)
0066 -2          END
0067 -1        END.

EXECUTION OPTIONS  DEBUG.1  TIME=5 SECONDS PAGES=5

001.50 SECONDS IN COMPILATION. (02196. 01498) BYTES OF CODE GENERATED
```

```
0000 1-      BEGIN COMMENT  REGINA CELIA BALBINO FIGUEIRA LISBOA
0001 --      ************************************************************************
0001 --      ************************************************************************
0001 --        ESTE PROGRAMA E UM PROCEDIMENTO HEURISTICO PARA MINIMIZAR O TEMPO
0001 --           TOTAL REQUERIDO PARA PROCESSAMENTO DE TAREFAS INDEPENDENTES EM
0001 --           PROCESSADORES IDENTICOS QUE OPERAM EM PARALELO.
0001 --        ESTE PROCEDIMENTO E CHAMADO DE ALGORITMO LPT, POIS PRIMEIRAMENTE
0001 --           RECRDENA AS TAREFAS EM ORDEM NAO DECRESCENTE EM RELACAO AO SEJ
0001 --           TEMPO DE PROCESSAMENTO PARA DEPOIS PROGRAMA-LAS.
0001 --      ************************************************************************
0001 --      ************************************************************************
0001 --        INTEGER M, N, I, AUX, AUX1
0002 --        INTFIELDSIZE =3
0003 --        READ (M,N)
0004 2-        BEGIN
0005 --          RECORD LISTA (INTEGER JOB  INTEGER TPRO  REFERENCE (LISTA) LINK)
0008 --          REFERENCE (LISTA) APONT, APONT1, NOVO1
0009 --          REFERENCE (LISTA) ARRAY NOVO(1  M)
0010 --          INTEGER ARRAY T(1  M)
0011 --          PROCEDURE INSORD (REFERENCE (LISTA) VALUE RESULT APONT
0012 --                            REFERENCE (LISTA) VALUE APONT1)
0013 3-            BEGIN
0014 --              REFERENCE (LISTA) APONT2, APONT3
0015 --              IF APONT=NULL THEN
0015 4-                BEGIN
0016 --                  APONT =APONT1
0017 --                  LINK(APONT) =NULL
0018 -4                END
0018 --              ELSE
0019 4-                BEGIN
0020 --                  IF TPRO(APONT1) =TPRO(APONT) THEN
0020 5-                    BEGIN
0021 --                      LINK(APONT1) =APONT
0022 --                      APONT =APONT1
0023 -5                    END
0023 --                  ELSE
0024 5-                    BEGIN
0025 --                      APONT2 =APONT
0026 --                      WHILE (APONT2 =NULL) AND (TPRO(APONT2) TPRO(APONT1)) DO
0026 6-                        BEGIN
0027 --                          APONT3 =APONT2
0028 --                          APONT2 =LINK(APONT2)
0029 -6                        END
0030 --                      IF APONT2=NULL THEN
0030 6-                        BEGIN
0031 --                          LINK(APONT3) =APONT1
0032 --                          LINK(LINK(APONT3)) =NULL
0033 -6                        END
0033 --                      ELSE
0034 6-                        BEGIN
0035 --                          LINK(APONT3) =APONT1
0036 --                          LINK(LINK(APONT3)) =APONT2
0037 -6                        END
0038 -5                    END
0039 -4                END
0040 -3            END
0041 --          WRITE( NUMERO DE TAREFAS = , N)
0042 --          WRITE( NUMERO DE PROCESSADORES = , M)
```

```
0043 --          WRITE( AS TAREFAS COM SEUS RESPECTIVOS TEMPOS DE PROCESSAMENTO COLOC
0043 --      ADOS ENTRE PARENTESES SAO )
0044 --          APONT =NULL
0045 --          FOR I =1 UNTIL N DO
0045 3-            BEGIN
0046 --              APONT1 =LISTA
0047 --              READON (JOB(APONT1), TPRO(APONT1))
0048 --              WRITEON(JOB(APONT1))
0049 --              WRITEON(  ( , TPRO(APONT1),  ),       )
0050 --              INSORD (APONT, APONT1)
0051 -3           END
0052 --          FOR I =1 UNTIL M DO
0052 --            NOVO(I) =NULL
0053 --          FOR I =1 UNTIL M DO
0053 3-            BEGIN
0054 --              T(I) =TPRO(APONT)
0055 --              NOVO(I) =APONT
0056 --              APONT =LINK(APONT)
0057 --              LINK(NOVO(I)) =NULL
0058 -3           END
0059 --          WHILE APONT = NULL DO
0059 3-            BEGIN
0060 --              AUX =1000
0061 --              FOR I =1 UNTIL M DO
0061 --                IF T(I))AUX THEN
0061 4-                BEGIN
0062 --                  AUX =T(I).
0063 --                  AUX1 =I
0064 -4                END
0065 --              T(AUX1) =T(AUX1)+TPRO(APONT)
0066 --              NOVO1 =LINK(APONT)
0067 --              LINK(APONT) =NOVO(AUX1)
0068 --              NOVO(AUX1) =APONT
0069 --              APONT =NOVO1
0070 -3           END
0071 --          AUX =0
0072 --          FOR I =1 UNTIL M DO
0072 --            IF T(I)  AUX THEN
0072 --              AUX =T(I)
0073 --          WRITE( OBSERVACAO  AS TAREFAS ESTAO AQUI LISTADAS EM ORDEM INVERSA
0073 --      EM RELACAO AO SEU PROCESSAMENTO )
0074 --          FOR I =1 UNTIL M DO
0074 3-            BEGIN
0075 --              WRITE( AS TAREFAS DO PROCESSADOR ,I, SAO )
0076 --              NOVO1 =NOVO(I)
0077 --              WHILE NOVO1  =NULL DO
0077 4-                BEGIN
0078 --                  WRITEON(JOB(NOVO1))
0079 --                  NOVO1 =LINK(NOVO1)
0080 -4                END
0081 -3           END
0082 --          WRITE ( O TEMPO TOTAL REQUERIDO PARA O PROCESSAMENTO DE TODAS AS TAR
0082 --      EFAS E IGUAL A , AUX)
0083 -2        END
0084 -1      END.

EXECUTION OPTIONS  DEBUG,1  TIME=5 SECONDS PAGES=5
```

```
0000 1-      BEGIN COMMENT  REGINA CELIA BALBINO FIGUEIRA LISBOA
0001 --      ************************************************************************
0001 --      ************************************************************************
0001 --        ESTE E UM PROCEDIMENTO HEURISTICO PARA MINIMIZAR O TEMPO TOTAL REQUERI
0001 --      DO PARA A EXECUCAO DE TODAS AS TAREFAS, EM DOIS PROCESSADORES NAO IDENTI
0001 --      COS QUE OPERAM EM PARALELO E, NAO EXISTINDO RESTRICOES DE PRECEDENCIA EN
0001 --      TRE AS TAREFAS.
0001 --      ************************************************************************
0001 --      ************************************************************************
0001 --        INTEGER M, N, I, A, B, AUX
0002 --        INTFIELDSIZE =3
0003 --        READ (M,N)
0004 2-        BEGIN
0005 --          INTEGER ARRAY T(1  2)
0006 --          INTEGER ARRAY TPRO (1  N,1  M)
0007 --          RECORD SEQ (INTEGER JOB  REAL QUOC  REFERENCE(SEQ) LINK)
0010 --          REFERENCE (SEQ) APONT1, APONT2, NOVO, NOVO1, NOVO2, NOVO3,
0010 --                          TEMP, TEMP1
0011 --          PROCEDURE INSORD (REFERENCE(SEQ) VALUE RESULT TEMP, NOVO)
0012 3-            BEGIN
0013 --              REFERENCE (SEQ) TEMP1, TEMP2
0014 --              IF NOVO = NULL THEN
0014 4-                BEGIN
0015 --                  NOVO =TEMP
0016 --                  TEMP =LINK(TEMP)
0017 --                  LINK(NOVO) =NULL
0018 -4                 END
0018 --               ELSE
0019 4-                 BEGIN
0020 --                   IF QUOC(TEMP) =QUOC(NOVO) THEN
0020 5-                     BEGIN
0021 --                       TEMP1 =LINK(TEMP)
0022 --                       LINK(TEMP) =NOVO
0023 --                       NOVO =TEMP
0024 --                       TEMP =TEMP1
0025 -5                     END
0025 --                   ELSE
0026 5-                     BEGIN
0027 --                       TEMP1 =NOVO
0028 --                       WHILE(TEMP1 =NULL) AND (QUOC(TEMP1) QUOC(TEMP)) DO
0028 6-                         BEGIN
0029 --                           TEMP2 =TEMP1
0030 --                           TEMP1 =LINK(TEMP1)
0031 -6                         END
0032 --                       LINK(TEMP2) =TEMP
0033 --                       TEMP =LINK(TEMP)
0034 --                       LINK(LINK(TEMP2)) =TEMP1
0035 -5                     END
0036 -4                 END
0037 -3            END
0038 --          FOR I =1 UNTIL N DO
0038 --            FOR J =1 UNTIL M DO
0038 --              READON(TPRO(I,J))
0039 --          T(1) =0
0040 --          T(2) =0
0041 --          APONT1 =APONT2 =NULL
0042 --          FOR I =1 UNTIL N DO
0042 3-            BEGIN
```

```
0043 --                NOVO =SEQ
0044 --                JOB(NOVO) =I
0045 --                IF TPRO(I,1))=TPRO(I,2) THEN
0045 4-                  BEGIN
0046 --                    T(1) =T(1)+TPRO(I,1)
0047 --                    LINK(NOVO) =APONT1
0048 --                    APONT1 =NOVO
0049 -4                  END
0049 --                ELSE
0050 4-                  BEGIN
0051 --                    T(2) =T(2)+TPRO(I,2)
0052 --                    LINK(NOVO) =APONT2
0053 --                    APONT2 =NOVO
0054 -4                  END
0055 -3              END
0056 --            IF T(1) =T(2) THEN
0056 3-              BEGIN
0057 --                IF T(1) T(2) THEN
0057 4-                  BEGIN
0058 --                    A =1
0059 --                    B =2
0060 --                    TEMP =APONT1
0061 --                    TEMP1 =APONT2
0062 -4                  END
0062 --                ELSE
0063 4-                  BEGIN
0064 --                    A =2
0065 --                    B =1
0066 --                    TEMP =APONT2
0067 --                    TEMP1 =APONT1
0068 -4                  END
0069 --                NOVO =NULL
0070 --                WHILE TEMP = NULL DO
0070 4-                  BEGIN
0071 --                    QUOC(TEMP) =TPRO(JOB(TEMP),A) / TPRO(JOB(TEMP),B)
0072 --                    INSORD (TEMP,NOVO)
0073 -4                  END
0074 --                NOVO1 =NULL
0075 --                NOVO2 =NOVO
0076 --                WHILE NOVO2 =NULL DO
0076 4-                  BEGIN
0077 --                    IF T(A) T(B)+TPRO(JOB(NOVO2),B) THEN
0077 5-                      BEGIN
0078 --                        IF NOVO2 = NOVO THEN
0078 --                          NOVO =LINK(NOVO)
0078 --                        ELSE
0079 --                          LINK(NOVO1) =LINK(NOVO2)
0080 --                        T(A) =T(A)-TPRO(JOB(NOVO2),A)
0081 --                        T(B) =T(B)+TPRO(JOB(NOVO2),B)
0082 --                        NOVO3 =TEMP1
0083 --                        TEMP1 =NOVO2
0084 --                        NOVO2 =LINK(NOVO2)
0085 --                        LINK(TEMP1) =NOVO3.
0086 -5                     END
0086 --                    ELSE
0087 5-                      BEGIN
0088 --                        NOVO1 =NOVO2
0089 --                        NOVO2 =LINK(NOVO2)
```

```
0090 -5                    END
0091 -4                  END
0092 --                IF T(A) =T(B) THEN
0092 --                  AUX =T(A)
0092 --                ELSE
0093 --                  AUX =T(B)
0094 -3              END
0094 --                ELSE
0095 --                  AUX =T(1)
0096 --            WRITE( NUMERO DE TAREFAS= , N)
0097 --            WRITE( NUMERO DE PROCESSADORES= , M)
0098 --            WRITE( TAREFAS COM SEUS RESPECTIVOS TEMPOS DE PROCESSAMENTO NOS PROC
0098 --        ESSADORES 1 E 2, COLOCADOS ENTRE PARENTESES  )
0099 --            FOR I =1 UNTIL N DO
0099 --              WRITE(I,  ( , TPRO(I,1), TPRO(I,2),  ) )
0100 --            WRITE( AS TAREFAS QUE DEVEM SER PROCESSADAS NO PROCESSADOR , A,  SAO
0100 --          )
0101 --            WHILE NOVO =NULL DO
0101 3-              BEGIN
0102 --                WRITEON (JOB(NOVO))
0103 --                NOVO =LINK(NOVO)
0104 -3              END
0105 --            WRITE( AS TAREFAS QUE DEVEM SER PROCESSADAS NO PROCESSADOR , B,  SAO
0105 --          )
0106 --            WHILE TEMP1 =NULL DO
0106 3-              BEGIN
0107 --                WRITEON(JOB(TEMP1))
0108 --                TEMP1 =LINK(TEMP1)
0109 -3              END
0110 --            WRITE( O TEMPO TOTAL REQUERIDO PARA O PROCESSAMENTO DE TODAS AS TARE
0110 --        FAS E IGUAL A , AUX)
0111 -2          END
0112 -1        END.

EXECUTION OPTIONS   DEBUG,1 _TIME=5 SECONDS PAGES=5

002.08 SECONDS IN COMPILATION, (03432, 01724) BYTES OF CODE GENERATED
```

REFERÊNCIAS BIBLIOGRÁFICAS

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

02. BAKER, K. R. - "Introduction to Sequencing and Scheduling". Wiley, New York (1974).

03. AHO, A., HOPCROFT, J. e ULLMAN, J. - " The design and Analysis of Computer Algorithms". Addison Wesley, Reading, Mass., (1974).

04. LUCCHESI, C., SIMON, J. e KOWALTOWSKI, J. - "Aspectos Teóricos da computação", IMPA, Rio de Janeiro (1977).

05. GAREY, M., GRAHAM, R. e JOHNSON, D. - "Performance Guarantees for Scheduling Algorithms". Opns. Res. 26, 3 - 21 (1978)

06. IBARRA, O. e KIM, C. - "Heuristic Algorithms for Scheduling Independent Task on Nonidentical Processors". J.Assoc. Comput. Mach. 24, 280 - 289 (1977).

07. GRAHAN, R. - "Bounds on Multiprocessing Timing Anomalies" SIAM J. Appl. Math. 17, 416 - 429 (1969).

08. MOORE J. M. - "An n job, one Machine Sequencing Algorithm for Minimizing the Number of late jobs". Management Sci. 15, 102 - 109 (1968).

09. LEUSTRA, J. K., KAN, R. e BRUCKER, P. - "Complexity of machine Scheduling Problems". A. D. Mathematics. 1, 343 - 362 (1977).

10. LAWLER, E. e MOORE, J. = "A functional equation and its application to resource allocation and sequencing Problems", Management Sci. 16, 77 - 84 (1969).

11. WILKERSON, L. J., e IRWIN, J. D. - "An Improved Algorithm for Scheduling Independente Tasks", AIIE Transactions, Vol. 3, nº 3 (1971).

12. KARP, R. M. - "Reducibility Among Combinational Problems , pp. 85 - 103 em R.E. Miller, J. W. Traıcher (eds.), Complexity of Computer Computations, Plenum Press, New York (1972)

13. LAWLER, E. L. - "Optimal sequencing of a single machine subject to Procedure Constraints", Management Sci. 544—546 (1973).

14. SAHNI, S. - "Algorithms for Scheduling Independents Tasks", J. Assoc. Comput. Mach. 23, 114 - 127 (1976).

15. HOROWITZ, E. e SAHNI, S. - "Exact and aproximate Algorithms for Scheduling Nonidentical Processores", J. Assoc. Comput. Mach. 23, 317 - 327 (1976).

16. GONZALES, T., IBARRA, O. e SAHNIS, S. - "Bounds for LPT Schedules on Uniform Processors", SIAM J. Comput. 6, 155 - 166 (1977).