ALGORITMOS POLINOMIAIS EXATOS OU APROXIMATIVOS PARA CERTOS PROBLEMAS EM SCHEDULING

Antonio de Almeida Pinho

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIENCIAS (D.Sc.)

Aprovada por:

JAIME LUIZ SZWARCFITER
(Presidente)

PAULO ROBERTO OLIVEIRA

JACK SCHECHTMAN

PAULO BOAVENTURA NETO

ANTONIO A. F. DE OLIVEIRA

SUELI MENDES DOS SANTOS

Rio de Janeiro - BRASIL

DEZEMBRO de 1983

Pinho, Antonio de Almeida

Algoritmos polinomiais exatos ou aproximativos para certos problemas em scheduling/Antonio de Almeida Pinho. - Rio de Janeiro: COPPE/UFRJ, 1983.

ix + 120p: il; 29,7 cm. - (Tese - Universidade Federal do Rio de Janeiro. Programa de Engenharia de Sistemas)

1. Teoria de Scheduling.2. Algoritmos aproximativos I. Título II. Série.

A minha esposa, Sulyr,

a meus filhos,

a meus pais.

AGRADECIMENTOS

Ao Professor Jaime Luiz Szwarcfiter, pela dedicação e segurança na orientação desta Tese.

Aos demais professores que constituiram a banca examinadora:

Paulo Roberto Oliveira

Jack Schechtman

Paulo Boaventura Neto

Antonio Alberto Fernandes de Oliveira

Sueli Mendes dos Santos

À secretária Célia Regina de Oliveira pelo trabalho de datilografia.

A todos que, de alguma forma, contribuiram para que este trabalho fosse levado a bom termo.

RESUMO

O presente trabalho examina alguns problemas da Teoria de Scheduling, propondo algoritmos polinomiais para minimização do tempo máximo de processamento e da soma dos tempos de processamento, tanto para um problema de máquinas paralelas, com máquinas uniformes e tempos unitários, como para uma particularização do problema de Flow-shop Scheduling, abordando tanto o caso de armazenamento ilimitado, como o caso de armazenamento zero (no wait).

Propomos, ainda, um algoritmo aproximativo para a minimização do tempo máximo de processamento, em um problema especial de duas máquinas não relacionadas, com os tempos de processamento sujeitos à restrição $q_i = \alpha p_i + \beta$, com qualidade melhor que a existente.

ABSTRACT

The present work examines some problems in Scheduling Theory. It contains proposals of polinomial time algorithms for mimimizing the maximum completion time and the sum of all completion times, both for Scheduling uniform machines with unit processing times and for a certain restriction of the flow shop problem. In the latter case, both unlimited and no buffering (no wait) flow shops are considered. In addition, an approximation algorithm is described for minimizing the maximum completion time in a special unrelated two machine problem, with processing time satisfying the restriction $q_j = \alpha p_j + \beta$, which improves the quality of a solution so far available.

INDICE

INTRODUÇÃO

Este trabalho pode ser dividido em tres partes; na primeira, constituida pelos Capítulos I e II, são apresentados os conceitos de complexidade de algoritmos, as classes P e NP e os conceitos básicos referentes a algoritmos aproximativos. Ainda nesta primeira parte, é descrita, de maneira informal, a Teoria de Scheduling, apresentando-se a notação utilizada no trabalho, bem como o relacionamento de complexidade entre os vários problemas em Scheduling.

A segunda parte, constituida dos tres Capítulos seguintes, constitui a tese propriamente dita. No Capítulo III, é descrito um problema de máquinas uniformes e tempos unitários e apresentado um algoritmo polinomial que obtem, de forma otima, o tempo mínimo total de processamento e a soma mínima dos tempos de processamento, para n jobs e m máquinas.

No Capítulo IV é abordado um problema de duas máquinas paralelas, onde os tempos de processamento obedecem a uma determinada relação, e para o qual é descrito um algoritmo aproximativo polinomial, para o tempo mínimo total de processamento. É calculada a complexidade e a qualidade do algoritmo.

No Capítulo V, é colocado o problema de Flow-shop Scheduling, impondo-se a condição de que, para cada job, o tempo de processamento seja o mesmo em qualquer máquina. Esta restrição torna polinomial o problema de obter as permutações de jobs que minimizam o tempo total de processamento e a soma dos tempos de processamento. São obtidas tais permutações, tanto

para o problema de "buffer" ilimitado como para "buffer" zero, este último conhecido também como atendendo a restrição "no wait".

Encerrando o trabalho, são apresentadas, no Capítulo VI, que constitui a terceira parte, sugestões de linhas de pesquisa, que visam generalizar os resultados obtidos.

CAPÍTULO I

COMPLEXIDADE

1.1. Análise de Algoritmos

Quando se cria um método de resolução de um determinado problema, e, para o qual deve ser usado um computador, isto é, deve ser escrito um algoritmo, surgem naturalmente diversas questões concernentes ao método e ao algoritmo:

- o algoritmo está correto? (isto é, ele realiza aquilo a que se propõe?)
- quanto tempo de computador o algoritmo necessita, para sua execução?
- qual a quantidade de memória utilizada?

Tais questões são colocadas principalmente a partir do instante em que se dispõe de dois ou mais algoritmos que resolvem o mesmo problema, e se deseja saber qual o melhor, ou qual o mais eficiente. Nesse caso, claramente, deve existir uma escala de critérios sob os quais os algoritmos serão julgados; o tempo necessário de execução e a memoria utilizada são parâmetros importantes, mas não são os únicos; a otimalidade (é possível construir um algoritmo melhor?), a facilidade de depuração e/ou correção, a facilidade de implementar modificações, a simplicidade de programação, entre outros, são também critérios uteis.

As tentativas de responder essas perguntas, e a necessidade de comparar algoritmos, deram origem ao ramo da Ciência da Computação conhecida por Análise de Algoritmos. Naturalmente, essas questões já existiam, pelo menos de forma subjacente, desde os primórdios da Computação, mas, apenas a partir dos anos 70 tem sido feito um esforço sistemático visando o desenvolvimento da Análise de Algoritmos.

Num primeiro enfoque, parece claro que qualquer medida que tenha por finalidade comparar dois algoritmos (já supostos corretos) não deve depender de sua implementação, isto é, não deve depender de coisas como qualidade do programador, computador utilizado, linguagem escolhida, etc. Dito de outra forma, a comparação deve incluir apenas fatores inerentes ao algoritmo. No entanto, nenhum dos fatores citados é realmente independente da implementação; em maior ou menor grau todos eles dependem do computador, da linguagem, da estrutura de dados utilizada e de vários outros aspectos da implementação. Tentativas de resolver esse problema (isto é, analisar o método subjacente ao algoritmo) foram feitas, como a criação de linguagens-padrão, como o MIX, de Knuth [1] . Mas tais linguagens, para atingir seus objetivos, devem estar em nível inconvenientemente próximo da linguagem de máquina, o que as torna não apropriadas, do ponto de vista de programação.

Atualmente, desenvolvem-se esforços num sentido ligeiramente diverso: associar a eficiência de um algoritmo à quantidade de trabalho computacional por ele executado. De certa forma, é uma medida do tempo de execução; é o que se chama complexidade do algoritmo.

Básicamente, os diversos trabalhos que um algoritmo realiza, ler e escrever valores, executar operações aritméticas (incluida aí a atribuição), comparar dois números, gastam tempos diferentes, em sua execução; em geral, a leitura e a impressão são mais demoradas que a multiplicação, e esta mais demorada que a soma. No entanto, com a finalidade de simplificar o cálculo, costuma-se considerar cada uma dessas atividades como uma "operação", e a complexidade do algoritmo é indicada por uma função que associa a n, o nº de dados de entrada, f(n), o número de operações executadas.

Aqui surge novo problema: em geral, para algoritmos que não sejam absolutamente triviais, a quantidade de trabalho computacional não depende apenas da quantidade de dados de entrada, mas também, muitas vezes, de certas propriedades desses dados. Por exemplo, um algoritmo de ordenação pode executar um número consideravelmente menor de operações para ordenar uma lista de n ítens, se esta possuir pequena porcentagem de elementos fora de ordem, do que se a lista estiver totalmente desordenada. Da mesma forma, se um sistema linear possuir muitos coeficientes nulos, pode exigir um número menor de operações, para um certo algoritmo, do que se poucos elementos forem iguais a zero.

Estudar o "comportamento medio" do algoritmo pode ser uma forma de se evitar esse problema, mas possui certos inconvenientes. O comportamento medio de um algoritmo pode ser obtido simplesmente calculando-se o número de operações executadas para cada entrada de tamanho n, e tomando-se a media. Esse resultado, no entanto, pode não ser util, se, na

prática, algumas entradas ocorrerem mais frequentemente que outras; um resultado mais significativo seria tomar a media ponderada. Seja D_n o conjunto de entradas de tamanho n para o problema em questão; para cada elemento I de D_n, seja p(I) a probabilidade de que I ocorra, e t(I) o número de operações executadas pelo algoritmo, com entrada I. Então, o comportamento medio pode ser definido:

$$A(n) = \sum_{I \in D_n} p(I)t(I)$$

Ora, mesmo que p(I) fosse estimado, seria impraticavel o cálculo de t(I) para cada entrada I, o que torna extremamente complicado o estudo do comportamento medio do algoritmo.

Outro enfoque para descrever o comportamento do algoritmo é baseado na análise do pior caso. A complexidade do pior caso pode ser definida por

$$W(n) = \max_{I \in D_n} t(I)$$

W(n) é simplesmente o número máximo de operações que o algoritmo executa, para qualquer entrada de tamanho n. Em geral, W(n) pode ser calculado mais fácilmente que A(n),e pode ser mais util que este para implementações, pois fornece um limite superior para o tempo de execução do algoritmo. Neste trabalho, os algoritmos serão analisados sempre sob o ponto de vista do pior caso.

A análise do pior caso tem ainda outra importante consequencia: fornece um critério para a otimalidade de um algoritmo. Definimos a complexidade F(n) de um problema P como o número mínimo de operações necessárias para resolver P com qualquer entrada de tamanho n. Dito de outra forma, F(n) é um limite inferior para a complexidade de pior caso de qualquer algoritmo que resolva P. Então, se obtivermos F(n) para um problema P (o que tem sido feito para problemas simples, como encontrar simultâneamente o máximo e o mínimo de uma lista de n itens, ordenar uma sequência de números, etc.), e construirmos um algoritmo A para P, com W(n) = F(n), concluimos que A é ótimo, isto é, não pode haver um algoritmo mais eficiente que A, para resolver o problema P, segundo esse critério.

1.2. Ordens de Grandeza

Digamos que, para um determinado problema P, construimos dois algoritmos, o primeiro com complexidade de pior caso, $W_1(n) = 3n^2$, e o segundo, com $W_2(n) = 25n$. Temos, portanto, que para $n > 8$, $W_1(n) > W_2(n)$ e, para $n \leq 8$, $W_1(n) < W_2(n)$. Na verdade, para quaisquer c_1 e c_2, existe k tal que $c_1n^2 > c_2n$ para todo $n > k$, e isto decorre do fato de que n^2 cresce mais rápidamente que n. Para tornar a análise de algoritmos independente dos dados de entrada, é necessário introduzir o conceito de ordem de grandeza de complexidade. Abaixo, apresentamos as ideias básicas desse conceito e a notação utilizada no decorrer do trabalho.

Sejam f e g funções em $\mathbb{N}$, números naturais. Dizemos que a ordem de f é menor ou igual que a ordem de g, e escrevemos "f é $O(g)$" se existem constantes positivas c e k tais que $f(n) \leq c\,g(n)$ para todo $n > k$. Temos, portanto, que $25n$ é $O(3n^2)$, mas $3n^2$ não é $O(25n)$; por outro lado, $3n^2$ e $O(n^2)$ e n^2 é $O(3n^2)$; em geral, duas funções que diferem por fatores constantes são de mesma ordem. O conjunto de funções de mesma ordem que f, notado $\theta(f)$, inclui todas as funções g tais que g é $O(f)$ e f é $O(g)$. Embora $\theta(f)$ seja um conjunto é convenção escrever "g é $\theta(f)$" ao invés de "$g \in \theta(f)$". Em polinomios, os termos de expoente menor que o grau não afetam a ordem; usamos então o termo de maior grau como uma representação canônica da ordem; assim, $n(n-1)$ é $\theta(n^2)$ e $10n^3 - n^2 + 8$ é $\theta(n^3)$.

Uma técnica util para verificar se g é $\theta(f)$ utiliza limites; se

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$$

para algum $c \neq 0$ finito, então g é $\theta(f)$.

Em termos de algoritmos, dados dois algoritmos A e B para o mesmo problema, com complexidades de pior caso $W_A(n)$ e $W_B(n)$, respectivamente, dizemos que A é "melhor" ou "mais eficiente" que B se W_A é $O(W_B)$ mas W_B não é $O(W_A)$. Dizemos que A possui complexidade menor que B.

1.3. As Classes P e NP

À medida em que eram desenvolvidos esforços no sentido de se obter a complexidade de algoritmos existentes, e de se construir algoritmos de complexidade menor, foi se tornando claro que os problemas podem ser classificados em dois grupos: problemas para os quais existem algoritmos de complexidade polinomial (tais problemas são ditos polinomialmente limitados) e problemas para os quais não são conhecidos, pelo menos até o momento, algoritmos de complexidade polinomial; para esses últimos as complexidades são expressas por funções como 2^n, $n!$, etc. Nas pesquisas por algoritmos polinomiais para essa última classe foi constatada a existencia de um conjunto de problemas com uma característica interessante: bastava que se construisse um algoritmo polinomial para um deles, e todos seriam polinomialmente limitados; não obstante, não foi possível obter tal algoritmo.

Esses fatos provocaram uma formalização da teoria, que procuramos descrever sucintamente, abaixo.

Um problema é dito "de decisão" se sua resolução consiste numa resposta "sim" ou "não"; em geral, os problemas de otimização e os problemas não limitados polinomialmente possuem uma versão correspondente nessa forma.

Um problema de decisão é dito estar na classe P se for polinomialmente limitado. A classe NP é constituída pelos problemas de decisão para os quais, dada uma entrada, uma solução proposta pode ser checada (isto é, podemos verificar se é realmente solução) em tempo polinomial no tamanho da en

trada. Para ajudar a compreender melhor esse último conceito, podemos citar o exemplo clássico do Problema do Caixeiro Viajante, em sua versão como problema de decisão: dadas n cidades e as distâncias entre cada par de cidades, existe um roteiro de comprimento $\leq k$ que, partindo de uma cidade, torne a ela, passando uma única vez em cada uma das outras? Cada solução proposta (uma permutação de cidades) pode ter seu comprimento calculado em tempo polinomial, e, portanto, este problema está em NP. Evidentemente, a classe P está contida na classe NP. No entanto, no caso do exemplo, não existe, até agora, algoritmo que garanta o fornecimento, em tempo polinomial, de um roteiro de comprimento $\leq k$; como também não foi provado que tal algoritmo não pode existir, não sabemos, para os problemas em NP, se eles estão também em P; isto é, não sabemos se P = NP.

Agora, vamos descrever o conceito de redução polinomial de um problema a outro, que desempenha um papel central nesta teoria. Suponha que A é um algoritmo que converte uma entrada x de um problema P_1 para uma y de um problema P_2, de tal forma que a resposta correta para $P_1(x)$ é sim se e somente se a resposta correta para $P_2(y)$ é sim. Claramente, se B é um algoritmo para P_2, B composto com A (B aplicado à saída de A) é um algoritmo para P_1. Se A é polinomial, dizemos que P_1 é polinomialmente transformável (ou polinomialmente redutivel) a P_2. Se P_1 for polinomialmente redutivel a P_2, e P_2 for polinomialmente redutivel a P_1, dizemos que P_1 e P_2 são polinomialmente equivalentes.

Finalmente, é possível mostrar que qualquer problema em NP é polinomialmente transformavel em um certo problema conhecido por CNF-SATISFABILIDADE; este problema e os que lhe são polinomialmente equivalentes (e o do Caixeiro Viajante é um deles) constituem uma nova classe, chamada Problemas NP-Completos. Podemos dizer, portanto, que NP-Completos são os problemas de decisão que estão em NP e tais que, se existir um algoritmo polinomial para um deles, então existem algoritmos polinomiais para cada problema em NP. Nos últimos anos, muito esforço tem sido dispendido para aumentar a lista de Problemas NP-Completos, que conta atualmente com várias centenas de membros.

1.4. Algoritmos Aproximativos

No estágio atual de desenvolvimento da computação, existe pouca esperança de que venham a ser obtidos algoritmos polinomialmente limitados para os problemas NP-Completos; os algoritmos existentes tem, em geral, a ordem de grandeza do seu tempo de execução descrita por uma função exponencial (como 2^n) e esse tipo de função cresce tão rapidamente que torna proibitivo o uso de tais algoritmos, mesmo para valores moderados de n.

Tendo em vista esses fatos, a resolução de problemas NP-Completos tem sido estudada com um enfoque ligeiramente diverso: a criação de algoritmos polinomialmente limitados que garantem, não a obtenção da solução ótima (embora possam obtê-la, eventualmente) mas uma solução "próxima da ótima". Tais algoritmos são chamados algoritmos aproximativos ou algoritmos heurísticos. Para muitos problemas tem sido desenvol-

vidas heurísticas muito simples e diretas, que, no entanto, fornecem resultados surpreendentemente bons.

Para tornar preciso o conceito de "solução próxima da ótima", necessitamos de algumas definições, fornecidas a seguir. Um problema de otimização P é um problema de minimização ou um problema de maximização, que consiste das seguintes tres partes:

a) um conjunto D_P de entradas

b) para cada $I \in D_P$, um conjunto finito $S_P(I)$ de soluções candidatas para I

c) uma função m_P que associa a cada $I \in D_P$ e a cada solução candidata $s \in S_P(I)$, um número racional positivo $m_P(I,s)$, chamado valor de solução para s.

Se P é um problema de minimização [maximização], então uma solução ótima para uma entrada $I \in D_P$ é uma solução candidata $s^* \in S_P(I)$ tal que, para todo $s \in S_P(I)$, $m_P(I,s^*) \leq m_P(I,s)$ $[m_P(I,s^*) \geq m_P(I,s)]$. Por simplicidade de notação podemos usar OPT(I) para designar o valor $m_P(I,s^*)$.

Um algoritmo A é dito um algoritmo aproximativo para P, se, dado qualquer $I \in D_P$, encontra uma solução candidata $s \in S_P(I)$. O valor $m_P(I,s)$ da solução candidata s encontrada por A pode ser designada por A(I). Se A(I) = OPT(I), para todo $I \in D_P$, A é dito um algoritmo ótimo.

Se o algoritmo A não é ótimo, torna-se necessário medir sua qualidade, isto é, dar uma medida de quão boa (ou

quão próxima da ótima) é a solução encontrada por A; e, para isso, são necessárias mais algumas definições.

Se P é um problema de minimização [maximização] definimos a razão $R_A(I)$ por

$$R_A(I) = \frac{A(I)}{OPT(I)} \quad \left[R_A(I) = \frac{OPT(I)}{A(I)} \right]$$

A razão R_A, performance absoluta, é dada por

$$R_A = \inf \{ r \geq 1 \mid R_A(I) \leq r \text{ para qualquer } I \in D_P \}$$

Então, a comparação de dois algoritmos aproximativos envolve duas medidas: a qualidade e a complexidade. Quando ambos são polinomialmente limitados, é usual dar maior importancia à qualidade, isto é, dizemos que um algoritmo aproximativo A, polinomialmente limitado, é melhor, ou mais eficiente, que um algoritmo aproximativo B, também polinomialmente limitado, se $R_A < R_B$.

Finalizando, gostariamos de salientar que os conceitos, conclusões e notação utilizados neste capítulo são baseados principalmente nos trabalhos de Karp [2], Weide [3], Baase [4] e Garey e Johnson [5].

CAPÍTULO II

TEORIA DE SCHEDULING

2.1. O Conceito de Scheduling

A Teoria de Scheduling aborda e procura resolver a seguinte classe de problemas: "Dadas m máquinas e n jobs para serem processados, forneça uma distribuição dos jobs que minimize um determinado aspecto do processamento". Colocado desta forma, o problema parece bastante geral e suficientemente amplo para incluir uma variedade muito grande de situações; de fato, problemas de alocação de recursos e de distribuição de tarefas, em qualquer uma de suas múltiplas formas podem ser incluídos nessa categoria; por isso, seria de se supor que uma análise dessa teoria já viesse se desenvolvendo há muito tempo. Não obstante, tal fato não ocorreu; parece que as pessoas que tiveram oportunidade de lidar com tais problemas, desenvolveram processos intimamente ligados a determinados aspectos acidentais, o que, via de regra, dificulta uma maior generalização. A primeira notícia que se tem de um problema de Scheduling estudado em seu aspecto teórico é o da "Carta de Gantt" [6] , modelos desenvolvidos durante a I Grande Guerra, para orientar a distribuição de carga em navios Aliados; a utilização desses modelos reduziu consideravelmente o tempo de utilização desses navios.

Foi apenas a partir da década de 50 que os modelos matemáticos da Teoria de Scheduling começaram a aparecer na literatura. Dessa época em diante, o interesse nessa área

tem crescido enormemente, principalmente devido à possibilidade de utilização de métodos da Pesquisa Operacional, Engenharia Industrial, Matemática Combinatória e Ciência da Computação, pois a Teoria de Scheduling se enquadra em qualquer um desses campos, dependendo do enfoque que se dê aos seus problemas.

2.2. Notação

O enunciado fornecido para o tipo de problemas estudados em Scheduling não deixa antever a imensa quantidade de problemas particulares que podem ser incluídos nessa categoria; para dar uma idéia das múltiplas formas que um problema pode assumir, podemos citar umas poucas variações em alguns dos aspectos envolvidos pelo problema:

- um job, para ser executado, deve passar por todas as máquinas, ou então por uma única;

- o tempo de processamento de um job é o mesmo em qualquer máquina, ou esses tempos guardam entre si alguma relação, ou não são relacionados;

- é possível interromper o processamento de um job e recomeçá-lo mais tarde, na mesma ou em outra máquina, ou isso não é possível;

- o número de máquinas pode ser fixado ou não;

- os jobs guardam entre si alguma relação de precedência (isto é, um job só pode ter seu processamento iniciado após um outro - ou outros - ter seu processamento termina

do), ou não existe tal relação;

- os tempos de processamento podem estar dentro de certos limites, ou não;

- cada job pode estar associado a um intervalo de tempo no qual o processamento deva ser executado, ou não existe tal intervalo.

O aspecto a ser minimizado também pode variar: o tempo total de processamento, a soma dos tempos de processamento de cada job, o custo do processamento, incluindo ou não multas atribuídas à violação de regras, como no caso de o job ser executado fora de seu intervalo.

Dessa forma, a escolha da notação é o primeiro problema que se coloca: como descrever, de forma clara, sucinta e precisa, um determinado problema?

A notação utilizada neste trabalho, e usada por Graham et al [7], consiste em descrever o problema através de um termo $\alpha/\beta/\gamma$, onde, a grosso modo, o parâmetro α fornece as características das máquinas, β indica particularidades dos jobs, e γ é o aspecto que se procura minimizar, no problema.

2.2.1. O Parâmetro α

Com relação às máquinas, os problemas de Scheduling se classificam em três grupos:

- existe uma única máquina (máquina única);

- cada job é executado em apenas uma máquina (máquina paralelas);

- cada job deve, para ser executado, passar por todas as máquinas (máquinas em sequência).

No caso de máquinas paralelas, podemos ainda ter:

- o tempo de processamento de cada job é o mesmo em cada máquina (máquinas idênticas);

- existe uma relação de proporcionalidade entre os tempos de processamento de todos os jobs em uma e em outra máquina: dito de outra forma, cada máquina M_i é k_i vezes mais rápida que M_1 (máquina uniformes);

- não há relação entre os tempos de processamento de um job, em máquinas diferentes (máquinas não relacionadas).

Para as máquinas em sequência, existem os casos:

- open-shop: cada job deve passar por todas as máquinas, e não há restrições quanto à ordenação;

- flow-shop: as máquinas estão ordenadas, e essa ordem é única, isto é, é a mesma para todos os jobs;

- job-shop: as máquinas estão ordenadas, mas a ordem é possívelmente diferente, para jobs diferentes.

Convencionamos, então a seguinte simbologia para α:

símbolos p/ α	descrição
1	máquina única
P	máquinas idênticas
Q	máquinas uniformes
R	máquinas não relacionadas
F	flow-shop
O	open-shop
J	job-shop

Fig. 2.1. O parâmetro α

Finalmente, quando o problema não trata de máquina única, mas especifica o número de máquinas, isso é indicado por uma constante que segue o símbolo usado. Assim, α = P2 indica um problema com duas máquinas idênticas, α = F3 indica um problema de flow-shop com três máquinas. Se tal constante não aparecer, o problema trata com um número não especificado de máquinas.

2.2.2. O parâmetro β

Este parâmetro é formado por um conjunto de símbolos que indicam certas características dos jobs. Como tais características são muito numerosas, definiremos aqui apenas as mais usuais, e aquelas com as quais vamos trabalhar.

a) Interrupção (preemption): se o processamento de um job puder ser interrompido e continuar mais tarde, na mesma ou em outra máquina, dizemos que o problema permite inter-

rupção; nesse caso, pmtn $\in \beta$. Caso contrário, pmtn $\notin \beta$.

b) Relação de precedência: o problema pode especificar uma relação de precedência entre os jobs, isto é, um determinado job só pode ter seu processamento iniciado depois que outro (ou outros) tiveram seu processamento terminado. Isso pode ser representado por um grafo direcionado acíclico G, no qual a aresta (v_i, v_j) indica que o job J_i precede o job J_j. Se tal relação existe, e G não é uma árvore, prec $\in \beta$; se tal relação existe, e G é uma árvore, tree $\in \beta$; nesse último caso, às vezes, há necessidade de distinguir as árvores nas quais as arestas são dirigidas das folhas para a raiz, e usamos intree ao invés de tree, em β, e as árvores nas quais as arestas são dirigidas da raiz para as folhas, e usamos outree, ao invés de tree em β. Se não houver relação de precedência, prec, tree $\notin \beta$.

c) Datas de disponibilidades (release dates): se o problema especificar instantes r_j a partir dos quais o job J_j fica disponível para processamento, $r_j \in \beta$; se $r_j = 0$ para todo job J_j, $r_j \notin \beta$.

d) Limites para os tempos de processamento: se todos os tempos de processamento dos jobs nas máquinas foram iguais, $p_j = 1 \in \beta$; se os tempos de processamento forem limitados, $\underline{p} \leq p_{ij} \leq \bar{p} \in \beta$. Se não houver restrições quanto aos tempos de processamento, $p_j = 1$, $\underline{p} \leq p_{ij} \leq \bar{p} \notin \beta$.

Embora possam existir outros parâmetros em β, nos limitaremos a esses, por enquanto; é interessante notar que

pode ocorrer, num problema, $\beta = \phi$.

Resumimos, abaixo, os parâmetros que podem aparecer em β.

símbolo em β	descrição
pmtn	interrupções permitidas
prec	as relações de precedência formam um grafo
tree (intree,outree)	as relações de precedência formam uma árvore
r_j	existem datas de disponibilidade
$p_j = 1$	os tempos são iguais
$\underline{p} \leq p_{ij} \leq \bar{p}$	os tempos são limitados

Fig. 2.2. O parâmetro β

2.2.3. O parâmetro γ

Este parâmetro, dito critério de otimalidade, procura descrever o aspecto do processamento a ser minimizado. Normalmente, podemos destacar quatro desses aspectos em uma distribuição:

a) Tempo de processamento

Para cada job J_j, definimos C_j como o momento em que termina o processamento de J_j; então podemos querer minimizar C_{max}, o tempo total de processamento, ou $\sum C_j$, a soma dos momentos em que terminam os processamentos dos jobs; ou ainda,

$\sum w_j C_j$, no qual atribuimos um peso $w_j \geq 0$ a cada job J_j, procurando descrever a importância relativa de uns jobs sobre outros.

b) Atraso (lateness)

Nesse caso, o problema fornece, para cada job J_j, um limite d_j para o término de seu processamento, e definimos $L_j = C_j - d_j$; o usual é que se queira minimizar L_{max}, que, inclusive, pode ser negativo.

c) Retardo (tardiness)

Definimos, para cada job J_j, $T_j = \max \{0, C_j - d_j\}$ e podemos querer minimizar $\sum T_j$ ou $\sum w_j T_j$; neste último caso, existe um peso w_j atribuído a cada job J_j, de forma análoga ao ítem (a).

d) Penalidade unitária

Podemos definir, para cada job J_j, uma penalidade U_j, na forma

$$U_j = \begin{cases} 0 & \text{se } C_j \leq d_j \\ 1 & \text{se } C_j > d_j \end{cases}$$

caracterizando a ultrapassagem do limite para o tempo de processamento. Nesse caso, é usual minimizarmos $\sum U_j$ ou $\sum w_j U_j$.

Vejamos um exemplo, para tornar mais claros esses conceitos: considere um problema com 5 jobs J_1, J_2, J_3, J_4 e J_5, e 3 máquinas M_1, M_2, M_3, não relacionadas, com os se-

guintes tempos de processamento:

		JOBS				
		J_1	J_2	J_3	J_4	J_5
MAQS.	M_1	3	5	8	2	1
	M_2	6	4	6	5	2
	M_3	4	2	4	6	3

Fig. 2.3. O elemento p_{ij} representa o tempo de processamento do job J_j na máquina M_i

Vamos examinar os parâmetros tempo de processamento, atraso, retardo e penalidade unitária, para a seguinte distribuição: jobs J_5 e J_4 na máquina M_1, job J_1 na máquina M_2 e jobs J_2 e J_3 na máquina M_3. O diagrama da figura 2.4 representa tal distribuição:

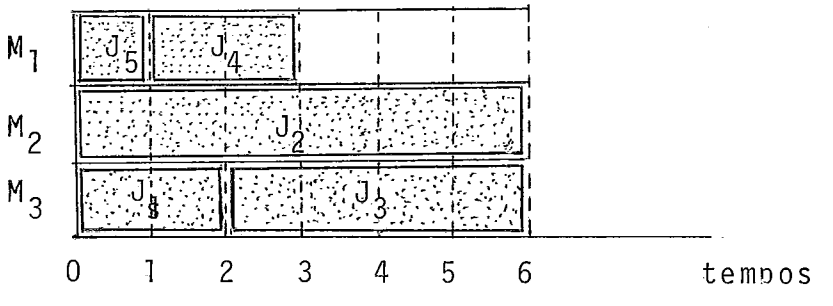


Fig. 2.4. Diagrama de um processamento

Com relação ao tempo de processamento, nessa distribuição, temos:

$$C_1 = 6$$

$$C_2 = 2$$

$$C_3 = 6$$

$$C_4 = 3$$

$$C_5 = 1$$

o que fornece $C_{max} = 6$ e $\sum C_j = 6 + 2 + 6 + 3 + 1 = 18$.

Para o caso de tempos ponderados de processamento, podemos, por exemplo, supor os pesos $w_1 = w_2 = w_3 = 1$ e $w_4 = w_5 = 2$.

Temos então $\sum w_j C_j = 6 + 2 + 6 + 2*3 + 2*1 = 22$.

Para ilustrarmos o "atraso" $L_j = C_j - d_j$ vamos arbitrar um limite $d_j = 4$ para todo job J_j. Na distribuição do exemplo, vem:

$$L_1 = 6 - 4 = 2$$

$$L_2 = 2 - 4 = -2$$

$$L_3 = 6 - 4 = 2$$

$$L_4 = 3 - 4 = -1$$

$$L_5 = 1 - 4 = -3$$

o que forneceu $L_{max} = 2$

Para o "retardo" $T_j = \max \{0, C_j - d_j\}$, nesse mesmo exemplo, teríamos

$$T_1 = \max \{0,2\} = 2$$

$$T_2 = \max \{0,-2\} = 0$$

$$T_3 = \max \{0,2\} = 2$$

$$T_4 = \max \{0,-1\} = 0$$

$$T_5 = \max \{0,-3\} = 0$$

dando, portanto, $\sum T_j = 4$

Para o retardo ponderado, podemos definir os pesos w_j da seguinte forma: $w_1 = 1$, $w_2 = 2$, $w_3 = 3$, $w_4 = 2$, $w_5 = 1$.

Então, $\sum w_j T_j = 2*1 + 0*2 + 2*3 + 0*2 + 0*1 = 8$.

Finalmente, para penalidade unitária U_j, vem

$$U_1 = 1$$

$$U_2 = 0$$

$$U_3 = 1$$

$$U_4 = 0$$

$$U_5 = 0$$

Embora possamos definir outros critérios de otimalidade, nos limitaremos a esses, no presente trabalho. abaixo, um resumo para o parâmetro γ.

γ	procura-se minimizar
C_{max}	tempo total de processamento
$\sum C_j$	soma dos tempos de processamento
$\sum w_j C_j$	soma ponderada dos tempos de processamento
L_{max}	atraso máximo, $\max_j \{C_j - d_j\}$
$\sum T_j$	soma dos retardos, $\sum \max_j \{0, C_j - d_j\}$
$\sum w_j T_j$	soma ponderada dos retardos
$\sum U_j$	soma das penalidades
$\sum w_j U_j$	soma ponderada das penalidades

Fig. 2.5 O parâmetro γ

Deve-se notar, finalmente, que, se $\sum w_j C_j$ for mínimo também o será $\sum w_j L_j$, e vice-versa, pois diferem pela constante $\sum w_j d_j$; ainda, que um schedule que minimiza L_{max}, minimiza também T_{max} e U_{max}, mas a recíproca não é verdadeira.

2.3. Relação de Complexidade em Scheduling

Do que foi exposto, deve ter ficado evidente que certos problemas em Scheduling são casos particulares de outros, pois que as restrições impostas aos primeiros estão incluídas nas restrições dos segundos; por exemplo, dados dois problemas A: P3/tree, $p_j = 1/C_{max}$, e B: Q/prec/L_{max}, é fácil ver que A é um caso particular de B. Nesse caso, dize-

mos que A está contido em B. Esse fato tem importância na medida em que um algoritmo desenvolvido para B resolve também A. Claramente, $A \subset B$ se e somente se $\alpha_A \subset \alpha_B$, $\beta_A \subset \beta_B$, $\gamma_A \subset \gamma_B$. Abaixo, descrevemos os grafos de inclusão dos parâmetros, em que (v_1, v_j) significa v_i v_j:

a) o parâmetro α

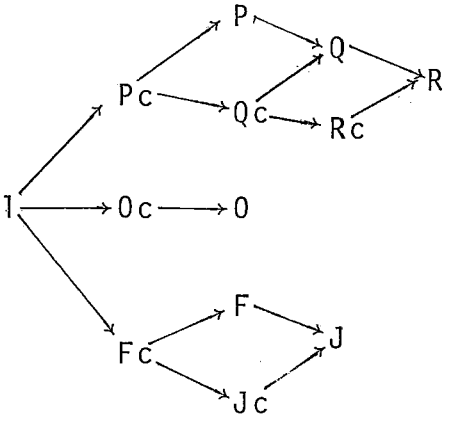
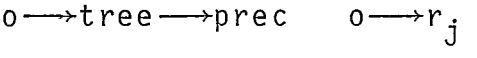


Fig. 2.6. Grafo de inclusão para α

b) o parâmetro β



o pmtn	(relação de precedência)	(data de disponibilidade)
(interrupção)		

$$p_j = 1 \longrightarrow \underline{p} \leq p_{ij} \leq \bar{p} \longrightarrow o$$

(limites para os tempos de processamento)

Fig. 2.7. Grafo de inclusão para β

O símbolo o indica "não especificado"; é interessante notar que entre problemas "sem interrupção" e "com interrupção" não há relação de inclusão.

c) o parâmetro γ

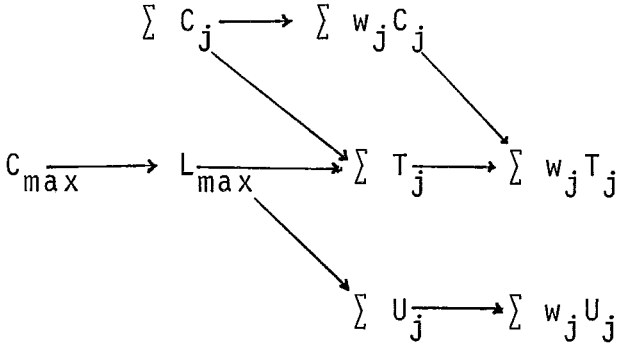


Fig. 2.8 Grafo de inclusão para γ

Com esses digrafos, é possível construir um outro digrafo, no qual cada vértice corresponde a um problema, de tal forma que, se existe caminho de A para B, o problema A está contido no problema B; dessa forma, se A for NP-completo, B será, no mínimo NP-completo; se existe um algoritmo para B com certa complexidade, A poderá ser resolvido com complexidade igual ou menor.

CAPÍTULO III

MÁQUINAS UNIFORMES, TEMPOS UNITÁRIOS

3.1. O Problema

3.1.1. Definição

Neste capítulo vamos estudar um problema de Scheduling, constituido por n jobs $J_1, J_2, \ldots, J_n$, com tempos de processamentos iguais (tempos unitários), e por m máquinas $M_1, M_2, \ldots, M_m$, paralelas e uniformes, isto é, cada job deve ser processado uma única vez em uma delas (máquinas paralelas) e existe uma relação de proporcionalidade entre as velocidades de processamento das máquinas (máquinas uniformes). Isto significa que, se um job J_j gasta um tempo p_j para ser executado em uma máquina, digamos M_1, gastará um tempo $q_i p_j$ (onde q_i é um real positivo) para ser executado na máquina M_i, $i=2,3,\ldots, m$, e isto é verdadeiro para todos os jobs.

Então, por simplicidade de notação, podemos fazer $p_j = 1$, $j=1,2,\ldots, n$ em uma das máquinas (na mais rápida, por exemplo) e indicar os tempos, nas outras máquinas, pelos coeficientes de proporcionalidade.

Para exemplificar, vamos considerar um problema de 3 máquinas e 5 jobs, no qual a máquina M_1 é a mais rápida, 4 vezes mais rápida que a máquina M_2, e 4.5 vezes mais rápida que a máquina M_3; os tempos de processamento podem ser descritos na forma abaixo:

		Tempos dos Jobs
MAQS	M_1	1
	M_2	4
	M_3	4,5

Fig. 3.1 - Tempos de processamento de cada um dos Jobs, nas máquinas

Evidentemente, não é essencial que em alguma máquina os tempos sejam iguais a 1. A matriz abaixo descreve exatamente o mesmo conjunto de jobs e máquinas:

		Tempos dos Jobs
MAQS	M_1	2
	M_2	8
	M_3	9

Fig. 3.2 - Tempos de processamento de cada um dos jobs, nas máquinas

Dessa forma, o problema pode ser descrito dizendo se que, para cada máquina M_i , i=1,2 ,..., m, todos os jobs têm o mesmo tempo de processamento, p_i. Essa forma tem a vantagem

de fazer com que os tempos de processamento assumam valores inteiros, se os coeficientes de proporcionalidade entre as velocidades das máquinas forem números racionais, o que sempre ocorre em computação.

Um aspecto importante desse problema é que os jobs são idênticos entre si, no que se refere ao tempo de processamento. Desse fato decorre duas conclusões:

a) um algoritmo que procura obter uma distribuição deve se limitar a escolher a máquina onde o job (qualquer job) deve ser alocado, a cada passo;

b) uma distribuição fica definida quando se fornece o número de jobs, em cada máquina.

Neste capítulo vamos desenvolver um algoritmo que obtem uma distribuição com o menor tempo total de processamento (C_{max}); e mostrar que essa distribuição possui também a menor soma total dos tempos de processamento (ΣC_j). Com os dados do exemplo acima, o algoritmo forneceria: 4 jobs na máquina M_1, 1 job na máquina M_2, e nenhum job na máquina M_3, o que pode ser representado pela figura 3.3, abaixo

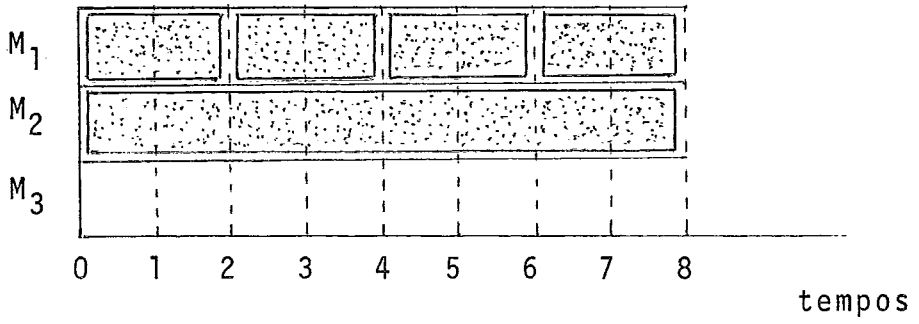


Fig. 3.3. Diagrama do processamento

Dessa forma, temos $C_{max} = 8$ e $\Sigma C_j = 2 + 4 + 6 + 8 + 8 = 28$

3.1.2. Notação

Como dissemos no Capítulo II, vamos utilizar, em todo o decorrer do trabalho, para descrever os problemas abordados, a notação usada por Graham et al [7], onde o problema fica caracterizado por um terno $\alpha/\beta/\gamma$, no qual α fornece as caracteristicas das máquinas, β indica particularidades dos jobs, e γ descreve o aspecto que se procura minimizar, no problema.

No problema em pauta, temos máquinas paralelas e uniformes, o que é indicado por $\alpha = Q$; os tempos de processamento dos jobs são unitários, logo $\beta = \{p_j = 1\}$; e, finalmente vamos abordar, numa primeira etapa, a minimização do tempo total de processamento, caso em que $\gamma = C_{max}$, e, numa segunda fase

a minimização da soma dos tempos de processamento, caso em que $\gamma = \Sigma C_j$. Portanto, os dois problemas, segundo a notação utilizada, são indicados por

$Q/P_j = 1/C_{max}$ e $Q/P_j = 1/\Sigma C_j$.

3.1.3. Literatura existente

Graham et al [7] descrevem um algoritmo para os problemas $Q/P_j = 1/f_{max}$ e $Q/P_j = 1/\Sigma f_j$, onde $f_{max} \in \{C_{max}, L_{max}\}$ e $\Sigma f_j \in \{\Sigma C_j, \Sigma T_j, \Sigma U_j, \Sigma w_j C_j, \Sigma w_j T_j, \Sigma w_j U_j\}$, no qual os problemas são colocados segundo um modelo de redes de transporte. Neste algoritmo, o tempo requerido para preparar os dados é $O(m\,n^2)$, e o tempo de resolução propriamente dito é $O(n^3)$, para dados inteiros; assumindo $m<n$, temos que a complexidade do algoritmo é $O(n^3)$. Lawler [8] descreve um algoritmo para $P/p_j = 1/\Sigma U_j$ com complexidade $O(n \log n)$. Os problemas tratados nesta tese não são tão gerais quanto os citados por Graham et al [7], mas a complexidade do segundo algoritmo proposto é bem menor: $O(m \log m + \log n)$

3.2. O Algoritmo A

3.2.1. Descrição

Neste ítem, vamos descrever um primeiro algoritmo, a que chamamos Algoritmo A, com complexidade $O(n \log m)$ que resolve, de forma ótima, o problema $Q/p_j = 1/C_{max}$, com m máquinas e n jobs; em seguida, vamos mostrar que a distribuição obtida por esse algoritmo também possui ΣC_j mínimo, isto é, que o algoritmo também resolve, de forma ótima, o problema $Q/p_j = 1/\Sigma C_j$.

No próximo item 3.3, no entanto, vamos desenvolver outro algoritmo, chamado Algoritmo B, que obtem a mesma distribuição obtida pelo Algoritmo A, porem com complexidade menor $O(m\log m+\log n)$ (suposto $m<n$). O motivo que nos levou a manter, neste trabalho, o Algoritmo A, é, que, por sua simplicidade, as provas de otimalidade são rápidas e diretas, e, ainda, permite a obtenção, de forma simples, de resultados importantes para o Algoritmo B.

Posto isso, vamos ao Algoritmo A, para o problema $Q/p_j = 1/C_{max}$, com m máquinas $M_1, M_2, \ldots, M_m$, e n jobs $J_1, J_2, \ldots, J_n$, em que o tempo de processamento de cada job na máquina M_i, $i=1,2,\ldots,m$, é indicado por p_i.

A estratégia do algoritmo é bem simples: consiste, em cada passo, a dar preferência à máquina que produza, alocando o job, o menor tempo total de processamento. Fazendo T_i ser o tempo de processamento da máquina M_i, e s_i, o número de jobs nessa máquina, o Algoritmo A pode ser descrito:

Algoritmo A

início

para i=1 até m passo 1 faça $[T_i \leftarrow 0, s_i \leftarrow 0]$

organize, em ordem não decrescente, uma lista dos valores $T_i + p_i$, i=1,2 ,..., m

para i=1 até n passo 1 faça

início

TOPO ← número da 1ª máquina da lista

$s_{TOPO} \leftarrow s_{TOPO} + 1$ (alocação do i-ésimo job

$T_{TOPO} \leftarrow T_{TOPO} + p_{TOPO}$ (na máquina M_{TOPO}

atualize a lista dos valores $T_i + p_i$

fim

fim

Do ponto de vista da otimalidade do Algoritmo, é irrelevante, na organização e atualização da lista, a forma pela qual os empates são resolvidos. No entanto, como vamos necessitar, no próximo item 3.3, de que o Algoritmo B, lá descrito, obtenha exatamente a mesma distribuição que o Algoritmo A, podemos estabelecer, desde já, uma regra para resolver empates: dá-se preferência à máquina melhor colocada lexicográficamente, isto é, se $T_i + p_i = T_j + p_j$, e $i < j$, dá-se preferência à máquina M_i.

3.2.2. Um exemplo

A título de ilustração, vamos apresentar um problema $Q/p_j = 1/C_{max}$, e a sua resolução pelo Algoritmo A. Considere, então, um problema com 3 máquinas M_1, M_2, M_3, e seis jobs, J_1, J_2, J_3, J_4, J_5 e J_6, com os tempos de processamento $p_1 = 2$, $p_2 = 3$, e $p_3 = 7$.

Antes de mais nada, o algoritmo inicializa os tempos de processamento das máquinas, T_1, T_2, T_3, e o número de jobs alocados, s_1, s_2, s_3, com o valor zero.

Em seguida, organiza a lista dos valores $T_i + p_i$, i=1,2,3, em ordem não decrescente:

Máquina	$T_i + p_i$
M_1	2
M_2	3
M_3	7

Agora, a alocação dos jobs, um por passo:

1º passo: TOPO ← 1

$T_1 \leftarrow 2$

$s_1 \leftarrow 1$

O primeiro job foi alocado em M_1

Atualização da lista:

Máquina	$T_i + p_i$
M_2	3
M_1	4
M_3	7

2º passo: TOPO ← 2

$T_2 \leftarrow 3$

$s_2 \leftarrow 1$

O segundo job foi alocado em M_2

Atualização da lista

Máquina	$T_i + p_i$
M_1	4
M_2	6
M_3	7

3º passo: TOPO ← 1

$T_1 \leftarrow 4$

$s_1 \leftarrow 2$

O terceiro job foi alocado em M_1

Atualização da lista

Máquina	$T_i + p_i$
M_1	6
M_2	6
M_3	7

4º passo: TOPO ← 1

$T_1 \leftarrow 6$

$s_1 \leftarrow 3$

O quarto job foi alocado em M_1

Atualização da lista

Máquina	$T_i + p_i$
M_2	6
M_3	7
M_1	8

5º passo: TOPO ← 2

$T_2 \leftarrow 6$

$s_2 \leftarrow 2$

O quinto job foi alocado em M_2

Atualização da lista

Máquina	$T_i + p_i$
M_3	7
M_1	8
M_2	9

6º passo: TOPO ← 3

$T_3 \leftarrow 7$

$s_3 \leftarrow 1$

O sexto job foi alocado em M_3

O resultado final, 3 jobs em M_1, 2 em M_2 e 1 job em M_3, com $C_{max} = T_3 = 7$ pode ser descrito pelo diagrama abaixo, em que os números envolvidos por um círculo indicam a ordem de alocação dos jobs.

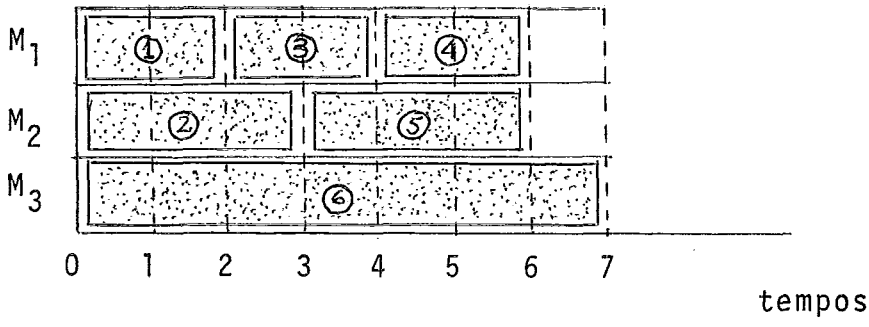


Fig. 3.4. Diagrama do processamento da distribuição obtida pelo Algoritmo A

3.2.3. A complexidade

Descrevemos no quadro abaixo, a complexidade de cada passo do Algoritmo A. Deve-se notar que, uma vez que a lista de valores $T_i + p_i$, $i=1,2,\ldots, m$ possui m elementos, sua organização tem complexidade $O(m \log m)$ e sua atualização, $O(\log m)$

<table>
<tr><td colspan="2">para i:=1 até m passo 1 faça
$[T_i \leftarrow 0;\ s_i \leftarrow 0]$</td><td>O(m)</td></tr>
<tr><td colspan="2">organize em ordem não decrescente, uma lista dos valores $T_i + p_i$, i=1,2 ,..., m</td><td>O(mlogm)</td></tr>
<tr><td colspan="2">para i=1 até n passo 1 faça</td><td rowspan="3">O(nlogm)</td></tr>
<tr><td>TOPO ← nº da 1ª máquina
$s_{TOPO} \leftarrow s_{TOPO} + 1$
$T_{TOPO} \leftarrow T_{TOPO} + p_{TOPO}$</td><td>O(c)</td></tr>
<tr><td>Atualize a lista</td><td>O(logm)</td></tr>
</table>

Fig.3.5. Determinação da complexidade do Algoritmo A

A complexidade do algoritmo é, então, O(nlog m), uma vez que $n \geq m$.

3.2.4. Otimalidade quanto a C_{max}

Neste item, vamos mostrar que a distribuição obtida pelo Algoritmo A possui C_{max} mínimo. Com essa finalidade, devemos mostrar que essa distribuição, a que chamaremos distribuição S, é tal que, à medida que os jobs vão sendo alocados, a diferença entre C_{max} atual e o tempo de processamento de qualquer máquina M_i, i=1,2 ,..., m, se mantém menor ou igual a p_i. Designando por $T_i(S,k)$ o tempo de processamento da máquina M_i, i=1,2 ,..., m após a alocação do k-ésimo job, k=1,2 ,..., n, e fazendo

$$C_{max}(S,k) = \max \{T_1(S,k) , T_2(S,k) , \ldots , T_m(S,k)\}$$

vamos mostrar o

Lema 3.1

$$C_{max}(S,k) \leq T_i(S,k) + p_i \quad , \quad i=1,2 \ , \ldots , \ m, \quad k=1,2 \ , \ldots , \ n$$

Prova

Em primeiro lugar, é óbvio que qualquer que seja a distribuição S,

$$C_{max}\ (S,k) \leq C_{max}\ (S,k+1) \quad \text{para todo } k$$

Por outro lado, por construção do Algoritmo A,

$$C_{max}\ (S,k+1) = \min \{T_1(S,k) + p_1 \ , \ldots , \ T_m\ (S,k) + p_m\} \ \forall \ k$$

de onde decorre

$$C_{max}(S,k) \leq \min \{T_1(S,k) + p_1 \ , \ldots , \ T_m\ (S,k) + p_m\} \ \forall \ k$$

e a tese segue-se imediatamente.

Agora podemos mostrar a otimalidade do Algoritmo A, mas vamos mostrar algo mais geral: o Algoritmo A é ótimo mesmo durante sua execução, isto é, depois de alocado o k-ésimo job, k=1,2 ,..., m, a distribuição produzida tem C_{max} menor ou igual a qualquer outra distribuição com k jobs; e esse resultado já era se esperar, devido ao fato de os jobs não serem individualizados.

Teorema 3.1

Seja Z uma distribuição qualquer, diferente de S, com C_{max} (Z,k) o tempo total de processamento da distribuição Z, após a colocação do k-ésimo job. Então

$$C_{max}(S,k) \leq C_{max}(Z,k) \quad , \quad k=1,2 \ ,\ldots, \ n$$

Prova

Digamos que a distribuição S possua s_i jobs na máquina M_i , i=1,2 ,..., m, depois da alocação do k-ésimo job, e representemos por z_i seu equivalente na distribuição Z. Se Z é diferente de S, podemos garantir que existe $j \in \{1,2 \ ,\ldots, \ m\}$ tal que $z_j > s_j$; decorre imediatamente

$$T_j(Z,k) > T_j(S,k)$$

o que implica em

$$T_j(Z,k) \geq T_j(S,k) + p_j$$

Como, pelo Lema 3.1, $C_{max}(S,k) \leq T_i(S,k) + p_i$, i=1,2 ,..., m segue-se $C_{max}(S,k) \leq C_{max}(Z,k)$.

Corolario 3.1

$C_{max}(S,n) \leq C_{max}(Z,n)$ para qualquer distribuição Z.

Prova

Decorre imediatamente do Teorema 3.1, para k=n, e

garante a otimalidade do Algoritmo A, em relação ao tempo total de processamento.

3.2.5 Otimalidade quanto a $\sum C_j$

Vamos mostrar agora que a distribuição S, produzido pelo Algoritmo A, possui também $\sum C_j$ mínimo.

Teorema 3.2

Seja $\sum C_j(S)$ a soma dos tempos de processamento dos jobs na distribuição S, e $\sum C_j(Z)$ a soma dos tempos de processamento numa distribuição Z qualquer. Então

$$\sum C_j(S) \leq \sum C_j(Z)$$

Prova

Considere o seguinte procedimento: a partir de uma distribuição Z qualquer, retire, um por um, o job que produz o tempo máximo de processamento. Se os jobs forem numerados na ordem inversa de que são retirados (isto é, n é o nº do primeiro a ser retirado, n-1 é o nº do segundo, e assim por diante), vamos obter:

$$C_n(Z) = C_{max}(Z,n), \ldots, C_2(Z) = C_{max}(Z,2), C_1(Z) = C_{max}(Z,1)$$

$$C_n(S) = C_{max}(S,n), \ldots, C_2(S) = C_{max}(S,2), C_1(S) = C_{max}(S,1)$$

Como, do Teorema 3.1, $C_{max}(S,k) \leq C_{max}(Z,k)$, $k=1,2,\ldots,n$ segue-se $\sum C_j(S) \leq \sum C_j(Z)$.

3.3. O Algoritmo B

3.3.1. Descrição

Neste item vamos desenvolver um algoritmo, a que chamamos Algoritmo B, que obtem a mesma distribuição obtida pelo Algoritmo A, para os problema $Q/p_j=1/C_{max}$ e $Q/p_j=1/\sum C_j$, porem com complexidade menor.

Considere um problema $Q/p_j=1/C_{max}$ constituido por m máquinas $M_1, M_2, \ldots, M_m$, e n jobs $J_1, J_2, \ldots, J_n$, no qual o tempo de processamento de qualquer job na máquina M_j é designado por p_j, $j=1,2,\ldots,m$.

Considere, ainda, uma distribuição S, no qual o tempo de operação da máquina M_j seja representado por $T_j(S,n)$. O tempo total de processamento da distribuição S, é, por definição,

$$C_{max}(S) = \max_j \{T_j(S,n)\}$$

Designando por k_i , $i=1,2,\ldots,m$ o número de jobs alocados na máquina M_i, pela distribuição S, podemos escrever:

$$\sum_{i=1}^{m} k_i = n$$

$$k_1 p_1 = T_1(S,n)$$

$$k_2 p_2 = T_2(S,n)$$

$$\ldots\ldots\ldots\ldots$$

$$k_m p_m = T_m(S,n)$$

Evidentemente, se

$$T_1(S,n) = T_2(S,n) = \ldots = T_m(S,n)$$

então a distribuição S é ótima, em termos de C_{max}. Então, podemos tentar determinar essa distribuição, resolvendo o sistema (3.1), abaixo, de m equações, nas m variáveis $k_1, k_2, \ldots, k_m$:

$$\begin{aligned} &\sum_{i=1}^{m} k_i = n \\ &k_1 p_1 = k_2 p_2 \\ &k_2 p_2 = k_3 p_3 \\ &\ldots\ldots\ldots \\ &k_{m-1} p_{m-1} = k_m p_m \end{aligned} \qquad (3.1)$$

Sua solução, isto é, os valores inteiros de $k_1, k_2, \ldots, k_m$ descreverão a distribuição ótima S. No entanto, num problema qualquer $Q/p_j=1/C_{max}$ não podemos garantir

$$T_1(S,n) = T_2(S,n) = \ldots = T_m(S,n)$$

e os valores dos k_i poderão não ser inteiros; não obstante, podemos, resolvendo o sistema (3.1) obter os valores reais dos k_i e alocar $s_i = \lfloor k_i \rfloor$ jobs na máquina M_i, onde $\lfloor k_i \rfloor$ representa a maior inteiro menor ou igual a k_i.

Como $s_i \geq k_i - 1$, $i=1,2,\ldots,m$, vem

$$\sum_{i=1}^{m} s_i \geq n - m$$

isto é, restam por alocar, no máximo, m jobs; e esses jobs podem ser alocados segundo o processo descrito no Algoritmo A.

O Algoritmo B pode, então, ser descrito:

Algoritmo B

início

Resolva o sistema (3.1)

para i=1 até m passo 1 faça $\left[s_1 \leftarrow \lfloor k_i \rfloor \, , \, T_i \leftarrow s_i * p_i \right]$

Organize, em ordem não decrescente, uma lista dos valores $T_i + p_i$, i=1,2 ,..., m

para $j = \sum_{i=1}^{m} s_i + 1$ até m passo 1 faça

início

TOPO ← número da 1ª máquina da lista

$s_{TOPO} \leftarrow s_{TOPO} + 1$

$T_{TOPO} \leftarrow T_{TOPO} + p_{TOPO}$

Atualize a lista

fim

3.3.2. O sistema

Vamos, agora, procurar determinar a solução do sistema (3.1). Em primeiro lugar, podemos escrevê-lo na forma mais conveniente

$$\begin{cases} k_1 + k_2 + k_3 + \ldots + k_m = n \\ k_1 p_1 - k_2 p_2 = 0 \\ k_2 p_2 - k_3 p_3 = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ k_{m-1} p_{m-1} - k_m p_m = 0 \end{cases} \qquad (3.2)$$

ou, na forma matricial AK = B, com a matriz A de coeficientes, mxm:

$$A = \begin{bmatrix} 1 & 1 & 1 & \ldots\ldots\ldots & 1 & 1 \\ p_1 & -p_2 & 0 & \ldots\ldots\ldots & 0 & 0 \\ 0 & p_2 & -p_3 & \ldots\ldots\ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & & p_{m-1} & -p_m \end{bmatrix}$$

o vetor K de m incógnitas e o termo independente B

$$K = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_m \end{bmatrix} \qquad B = \begin{bmatrix} n \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Primeiramente, vamos mostrar que o sistema possui solução única, isto é, que $\det(A) \neq 0$

Lema 3.2

$$\det(A) = (-1)^{m-1} \prod_{j=1}^{m} p_j \sum_{t=1}^{m} \frac{1}{p_t}$$

Prova

Por indução. Para m=1, o lema é trivialmente verdadeiro, pois $A = [1]$, e

$$\det(A) = (-1)^{1-1} \prod_{j=1}^{1} p_j \sum_{t=1}^{1} \frac{1}{p_t} = \frac{p_1}{p_1} = 1$$

Suponha agora que o lema seja verdadeiro para m-1.

Então, desenvolvendo det(A) pela primeira coluna, vem:

$$\det(A) = \begin{vmatrix} -p_2 & 0 & 0 & \cdots & 0 & 0 \\ p_2 & -p_3 & 0 & \cdots & 0 & 0 \\ 0 & p_3 & -p_4 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & -p_{m-1} & 0 \\ 0 & 0 & 0 & \cdots & p_{m-1} & -p_m \end{vmatrix} - p_1 \begin{vmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ p_2 & -p_3 & 0 & \cdots & 0 & 0 \\ 0 & p_3 & -p_4 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & -p_{m-1} & 0 \\ 0 & 0 & 0 & \cdots & p_{m-1} & -p_m \end{vmatrix}$$

Ora, o primeiro determinante é triangular inferior, e vale o produto dos termos da diagonal principal

$$(-1)^{m-1} \prod_{j=2}^{m} p_j$$

e o segundo, por hipótese, vale

$$(-1)^{m-2} \prod_{j=2}^{m} p_j \sum_{t=2}^{m} \frac{1}{p_t}$$

Portanto, temos que

$$\det(A) = (-1)^{m-1} \prod_{j=2}^{m} p_j - p_1(-1)^{m-2} \prod_{j=2}^{m} p_j \sum_{t=2}^{m} \frac{1}{p_t} =$$

$$= (-1)^{m-1} \prod_{j=2}^{m} p_j \left[1 + p_1 \sum_{t=2}^{m} \frac{1}{p_t} \right] =$$

$$= (-1)^{m-1} \prod_{j=1}^{m} p_j \left[\frac{1}{p_1} + \sum_{t=2}^{m} \frac{1}{p_t} \right] =$$

$$= (-1)^{m-1} \prod_{j=1}^{m} p_j \sum_{t=1}^{m} \frac{1}{p_t} .$$

Agora, a solução do sistema:

Lema 3.3

$$k_i = \frac{n}{p_i} \frac{1}{\sum_{j=1}^{m} \frac{1}{p_j}} , \quad i=1,2 ,..., m$$

Prova

O sistema (3.2) é constituido por duas classes de equações; a primeira classe é formada apenas pela primeira equação

$$\sum_{i=1}^{m} k_i = n$$

e a segunda, pelas m-1 equações

$$k_{i-1}p_{i-1} - k_i p_i = 0 \quad i=2,3 ,..., m$$

Vamos mostrar que o valor sugerido para k_i satisfaz a ambas as classes. Para a primeira

$$\sum_{i=1}^{m} k_i = \sum_{i=1}^{m} \frac{n}{p_i} \frac{1}{\sum_{j=1}^{m} \frac{1}{p_j}} = n \sum_{i=1}^{m} \frac{1}{p_i} \frac{1}{\sum_{j=1}^{m} \frac{1}{p_j}} = n.$$

Para a segunda classe,

$$k_{i-1}p_{i-1} - k_i p_i = \frac{n}{p_{i-1}} \frac{1}{\sum_{i=1}^{m} \frac{1}{p_j}} p_{i-1} - \frac{n}{p_i} \frac{1}{\sum_{j=1}^{m} \frac{1}{p_j}} p_i = 0$$

De posse da solução do sistema, podemos reescrever o Algoritmo B:

```
Algoritmo B
início
SIP ← 0; ST ← 0
para j=1 até m passo 1 faça SIP ← SIP + 1/p_j
para i=1 até m passo 1 faça
        início
        k_i ← n/(p_i * SIP)
        s_i ← ⌊k_i⌋
        T_i ← s_i * p_i
        ST ← ST + s_i
        fim
Organize uma lista em ordem não decrescente dos valores
T_i + p_i, i=1,2 ,..., m
para j=ST + 1 até n passo 1 faça
        início
        TOPO ← número da 1ª máquina da lista
        s_TOPO ← s_TOPO + 1
        T_TOPO ← T_TOPO + p_TOPO
        Atualize a lista
        fim
fim
```

Com a finalidade de fazer com que os Algoritmos A e B produzam a mesma distribuição (o que será mostrado em 3.3.5) podemos introduzir no Algoritmo B o mesmo critério de desempate do Algoritmo A, na organização e atualização da lista: preferência para a máquina melhor colocada lexicográficamente.

3.3.3. A complexidade

Descreveremos, no quadro abaixo, a complexidade do Algoritmo B. Deve-se notar que a complexidade do passo "Atualize a lista" é $O(\log m)$, uma vez que a lista possui m elementos. Além disso, como

$$ST = \sum_{i=1}^{m} s_i \geq n-m$$

o último laço é executado, no máximo, m vezes, o que dá, para complexidade desse laço, $O(m \log m)$

leitura do valor n	$O(\log n)$
SIP ← 0; ST ← 0	$O(c)$
<u>para</u> j=1 <u>até</u> m <u>passo</u> 1 <u>faça</u> SIP ← SIP + $1/p_j$	$O(m)$
<u>para</u> i=1 <u>até</u> m <u>passo</u> 1 <u>faça</u> <u>início</u> $k_i \leftarrow n/(p_i * SIP)$ $s_i \leftarrow \lfloor k_i \rfloor$ $T_i \leftarrow s_i * p_i$ $ST \leftarrow ST + s_i$ <u>fim</u>	$O(m)$
Organize uma lista em ordem não decrescente, dos valores $T_i + p_i$, i=1,2 ,..., m	$O(m \log m)$
<u>para</u> j=ST + 1 <u>até</u> n <u>passo</u> 1 <u>faça</u> <u>início</u> TOPO ← nº da 1ª máq. da lista $s_{TOPO} \leftarrow s_{TOPO} + 1$ $T_{TOPO} \leftarrow T_{TOPO} + p_{TOPO}$ Atualize a lista <u>fim</u>	$O(m \log m)$

Fig. 3.6. Determinação da complexidade do Algoritmo B

A complexidade do Algoritmo B, é, portanto, $O(m \log m + \log n)$

3.3.4. Um exemplo

Para ilustrar a utilização do Algoritmo B, vamos resolver um problema $Q/p_j=1/C_{max}$, constituido por 5 máquinas e 30 jobs, com os dados:

$p_1 = 10$

$p_2 = 5$

$p_3 = 50$

$p_4 = 7$

$p_5 = 6$

Na resolução do sistema, temos:

$$SIP = \frac{1}{10} + \frac{1}{5} + \frac{1}{50} + \frac{1}{7} + \frac{1}{6} = 0.6295237$$

$k_1 = 4.765506 \implies s_1 = 4 \implies T_1 = 40$

$k_2 = 9.531015 \implies s_2 = 9 \implies T_2 = 45$

$k_3 = 0.9531 \implies s_3 = 0 \implies T_3 = 0$

$k_4 = 6.807867 \implies s_4 = 6 \implies T_4 = 42$

$k_5 = 7.942512 \implies s_5 = 7 \implies T_5 = 42$

Temos, portanto, 26 jobs alocados. A lista dos $T_i + p_i$ fica:

Máquina	$T_i + p_i$
M_5	48
M_4	49
M_1	50
M_2	50
M_3	50

Para j=27, vem:

$TOPO \leftarrow 5$

$s_5 \leftarrow 8$

$T_5 \leftarrow 48$

E a nova lista:

Máquina	$T_i + p_i$
M_4	49
M_1	50
M_2	50
M_3	50
M_5	54

Para j=28, temos:

$TOPO \leftarrow 4$

$s_4 \leftarrow 7$

$T_4 \leftarrow 49$

E a nova lista

Máquina	$T_i + p_i$
M_1	50
M_2	50
M_3	50
M_5	54
M_4	56

Para j=29

TOPO ← 1

s_1 ← 5

T_1 ← 50

A nova lista

Máquina	$T_i + p_i$
M_2	50
M_3	50
M_5	54
M_4	56
M_1	60

Finalmente, para j=30

TOPO ← 2

s_2 ← 10

T_2 ← 50

A distribuição ótima:

Máquina	Nº de jobs	Tempo de operação
M_1	5	50
M_2	10	50
M_3	0	0
M_4	7	49
M_5	8	48

Com C_{max} = 50.

3.3.5. Equivalencia ao Algoritmo A

Neste item vamos mostrar que os Algoritmos A e B obtem a mesma distribuição; dessa forma o Algoritmo B resolve, de forma ótima, os problemas $Q/p_j = 1/C_{max}$ e $Q/p_j = 1/\sum c_j$.

O Algoritmo B é constituido por duas etapas; na primeira, a resolução do sistema e a alocação dos $\sum_{i=1}^{m} s_i$ primeiros jobs, ele "ganha tempo" sobre o Algoritmo A; na sua segunda fase, ele é idêntico ao Algoritmo A. Portanto, para mostrar que os dois algoritmos obtem a mesma solução, é suficiente mostrar que a distribuição obtida pelo Algoritmo B, em sua primeira fase, será construida também pelo Algoritmo A, no seu $\sum_{i=1}^{m} s_i$ - ésimo passo. Ou ainda, como Algoritmo A encontra a solução ótima, é suficiente mostrar que o número de jobs alocados, em cada máquina, pelo Algoritmo B, em sua primeira fase, é menor ou igual ao número de jobs alocados pelo Algoritmo A.

Com essa finalidade, vamos mostrar o lema abaixo:

Lema 3.4

$$k_i p_i \leq C_{max}(S) \quad , \quad i=1,2 \ , \ldots , \ m$$

Prova

Ora, por (3.1), temos

$$k_i p_i = k_2 p_2 = \ldots = k_m p_m$$

Então, se existe algum $i \in \{1,2,\ldots,m\}$ tal que $k_i p_i > C_{max}(S)$, todos os $k_i p_i$ serão maiores que $C_{max}(S)$. Fazendo o_i o número de jobs alocados na máquina M_i, na distribuição S, ótima, temos que

$$C_{max}(S) \geq o_i p_i \qquad i=1,2,\ldots,m$$

Como estamos supondo

$$k_i p_i > C_{max}(S) \qquad i=1,2,\ldots,m$$

decorre $k_i p_i > o_i p_i \qquad i=1,2,\ldots,m$

ou $k_i > o_i \qquad i=1,2,\ldots,m$

o que é impossível, pois sabemos que

$$\sum_{i=1}^{m} k_i = \sum_{i=1}^{m} o_i = n$$

Portanto, $k_i p_i \leq C_{max}(S)$, $i=1,2,\ldots,m$

Agora podemos mostrar a equivalencia entre os algoritmos.

Teorema 3.3

Sejam o_i, $i=1,2,\ldots,m$, o número de jobs alocados na máquina M_i, na distribuição S, ótima, produzida pelo Algoritmo A; e $\lfloor k_i \rfloor$, $i=1,2,\ldots,m$, o número de jobs alocados na máquina M_i, pelo Algoritmo B, em sua primeira fase. Então

$$\lfloor k_i \rfloor \leq o_i \quad , \quad i=1,2,\ldots,m$$

Prova

1º caso: k_i é fracionário

Nesse caso, $\lfloor k_i \rfloor = k_i - \lambda_i$, $\lambda_i > 0$

Suponha que exista $i \in \{1,2 ,..., m\}$ tal que

$$o_i < \lfloor k_i \rfloor$$

Decorre: $o_i \leq \lfloor k_i \rfloor - 1$

$$o_i \leq k_i - \lambda_i - 1$$

$$o_i < k_i - 1$$

$$o_i + 1 < k_i$$

$$(o_i + 1)p_i < k_i p_i$$

Pelo lema 3.4, $(o_i + 1)p_i < C_{max}(S)$

No entanto, pelo lema 3.1, $(o_i + 1)p_i \geq C_{max}(S)$

o que caracteriza um absurdo. Logo,

$$o_i \geq \lfloor k_i \rfloor$$

para todo k_i fracionário.

2º caso: k_i é inteiro

Nesse caso, $\lfloor k_i \rfloor = k_i$

Suponha que exista $i \in \{1,2 ,..., m\}$ tal que

$$o_i < \lfloor k_i \rfloor$$

Decorre: $o_i < k_i$

$o_i \leq k_i - 1$

$o_i + 1 \leq k_i$

$(o_i + 1)p_i \leq k_i p_i$

Pelo lema 3.4, $k_i p_i \leq C_{max}(S)$

de onde decorre $(o_i + 1)p_i \leq k_i p_i \leq C_{max}(S)$

No entanto, pelo lema 3.1, $(o_i + 1)p_i \geq C_{max}(S)$

Portanto $(o_i + 1)p_i = C_{max}(S)$.

Podemos escrever então

$$(o_i + 1)p_i = k_i p_i = C_{max}(S)$$

Mas sabemos que todos os $k_i p_i$ são iguais. Logo.

$$k_i p_i = C_{max}(S) \quad i=1,2,\ldots, m$$

Por outro lado, como

$$\sum_{i=1}^{m} k_i = \sum_{i=1}^{m} o_i$$

se existe $i \in \{1,2,\ldots, m\}$ tal que $o_i < k_i$, existe também $j \in \{1,2,\ldots, m\}$ tal que

$$o_j > k_j$$

ou seja $$o_j p_j > k_j p_j$$

de onde decorre $o_j p_j > C_{max}(S)$, o que é absurdo.

Podemos garantir, portanto,

$$o_i \geq \lfloor k_i \rfloor \qquad i=1,2,\ldots, m$$

completando a demonstração.

3.3.6. Um limite para $R//C_{max}$

Considere o problema $R//C_{max}$, no qual se procura obter uma distribuição com tempo total de processamento mínimo ($\gamma = C_{max}$), de n jobs $J_1, J_2, \ldots, J_n$, em m máquinas $M_1, M_2, \ldots, M_m$, paralelas e não relacionadas ($\alpha = R$), onde p_{ij} representa o tempo de processamento do job J_i na máquina M_j ($i=1,2,\ldots, n$; $j=1,2,\ldots, m$). Este problema é NP-completo [7], de forma que as pesquisas tem se dirigido no sentido de obter algoritmos aproximativos polinomialmente limitados. Para esses algoritmos é extremamente útil se conhecer um limite inferior para o tempo de processamento mínimo; um desses limites, bem conhecido, pode ser obtido de forma simples: fazendo $p_i = \min_j \{p_{ij}\}$, e designando por C_{max}^* o tempo total mínimo de processamento (C_{max} ótimo), podemos garantir que

$$C_{max}^* \geq \frac{1}{m} \sum_{i=1}^{n} p_i$$

Outro limite inferior para C_{max}^* pode, no entanto, ser obtido a partir dos resultados deste capítulo. Do lema 3.4 sabemos que

$$C_{max}(S) \geq k_i p_i \quad , \quad i=1,2,\ldots m$$

e, do lema 3.3

$$k_i = \frac{n}{p_i} \quad \frac{1}{\sum_{j=1}^{m} \frac{1}{p_j}} \quad , \quad i=1,2 \ , \ldots , \ m$$

decorrendo, portanto

$$C_{max}(S) \geq \frac{n}{\sum_{j=1}^{m} \frac{1}{p_j}}$$

resultado esse que é válido para o problema $Q/p_j = 1/C_{max}$.

O lema 3.5, abaixo, faz a extensão para $R//C_{max}$.

Lema 3.5

Considere um problema $R//C_{max}$ com m máquinas $M_1, M_2 \ , \ldots , \ M_m$, e n jobs $J_1, J_2 \ , \ldots , \ J_n$, onde p_{ij} representa o tempo de processamento do job J_i na máquina M_j; se fizermos C_{max}^* o tempo de processamento da distribuição ótima, então

$$C_{max}^* \geq \frac{n}{\sum_{j=1}^{m} \frac{1}{p_j}}$$

onde $p_j = \min_i \{p_{ij}\}$

Prova

De fato, considere dois problemas P_1 e P_2, ambos $R//C_{max}$, P_2 tendo sido obtido de P_1 substituindo-se cada p_{ij}

por $\min_i \{p_{ij}\}$. Claramente,

$$C_{max}^* (P_2) \leq C_{max}^* (P_1)$$

mas P_2 é da forma $Q/p_j = 1/C_{max}$, de onde decorre

$$C_{max}^* (P_2) \geq \frac{n}{\sum_{i=1}^{m} \frac{1}{p_j}}$$

seguindo-se a tese imediatamente.

Tendo em vista esse resultado podemos garantir que

$$C_{max}^* \geq \max \left\{ \frac{1}{m} \sum_{i=1}^{n} \min_j \{p_{ij}\} \; , \; \frac{n}{\sum_{j=1}^{m} \min_i \{p_{ij}\}} \right\}$$

para qualquer problema $R//C_{max}$.

CAPÍTULO IV

MÁQUINAS PARALELAS, TEMPOS RELACIONADOS

4.1. O Problema

4.1.1. Descrição

No capítulo II, quando foram descritos os problemas de Scheduling, vimos que, para o caso de máquinas paralelas, havia tres tipos básicos de problemas: máquinas idênticas (α=P), no qual cada job tem o mesmo tempo de processamento em todas as máquinas; pode ser dito também que as máquinas possuem a mesma "velocidade de processamento"; máquinas uniformes (α=Q), no qual os tempos de processamento são proporcionais, isto é, as máquinas possuem "velocidades de processamento" diferentes. e essas velocidades não dependem dos jobs; e, finalmente máquinas não relacionadas (α=R), no qual não só as velocidades de processamento das máquinas são diferentes, como ainda dependem dos jobs, isto é um job J_k pode ser processado mais rapidamente na máquina M_i que na máquina M_j, e com o job J_r pode ocorrer o contrário.

Vamos abordar, neste capítulo, uma particularização do último tipo de problemas, para 2 máquinas. O problema pode ser descrito: "dadas duas máquinas paralelas M_1 e M_2, e n jobs $J_1, J_2, \ldots, J_n$, onde p_i e q_i representam os tempos de processamento de um job J_i, i=1,2 ,..., n, nas máquinas M_1 e M_2, respectivamente, com $q_i = \alpha p_i + \beta$, i=1,2 ,..., n onde α e β são reais quaisquer (porém com a condição

$p_i, q_i > 0 \ \forall i$) obtenha uma distribuição com tempo total de processamento mínimo".

Aqui cabe um parênteses. Ainda neste capítulo, vamos propor um algoritmo aproximativo para este problema, e determinar sua qualidade, isto é, vamos mostrar que a solução obtida está dentro de certos limites. Para isso, vamos precisar que $|\alpha| \leq 1$, embora tenhamos dito que α é um real qualquer. A exigência $|\alpha| \leq 1$, no entanto, não representa, realmente, uma restrição. Para mostrarmos isso, considere um problema no qual $q_i = \alpha p_i + \beta$ com $|\alpha| > 1$; renumeremos as máquinas (isto é, troquemos os tempos de processamento p_i e q_i).

Temos, então:

$$p_i = \alpha q_i + \beta$$

$$p_i - \beta = \alpha q_i$$

$$q_i = \frac{1}{\alpha} p_i - \frac{\beta}{\alpha}$$

Façamos $\alpha' = \frac{1}{\alpha}$ e $\beta' = -\frac{\beta}{\alpha}$

Então $q_i = \alpha' p_i + \beta'$, com $|\alpha'| \leq 1$

De sorte, que podemos supor, sem perda de generalidade, $q_i = \alpha p_i + \beta$, com $|\alpha| \leq 1$.

Descreveremos, a seguir, um exemplo do problema proposto, com 4 jobs J_1, J_2, J_3, J_4, com $\alpha=0,5$ e $\beta=1$. Os tempos de processamento:

		JOBS			
		J_1	J_2	J_3	J_4
MAQS	M_1	6	12	10	2
	M_2	4	7	6	2

Fig. 4.1 - Tempos de processamento do problema proposto

A distribuição ótima consiste em alocar os jobs J_1 e J_2 na máquina M_2 e os jobs J_3 e J_4 na máquina M_1, o que fornece um tempo total de processamento igual a 12. Essa distribuição pode ser ilustrada pela figura abaixo.

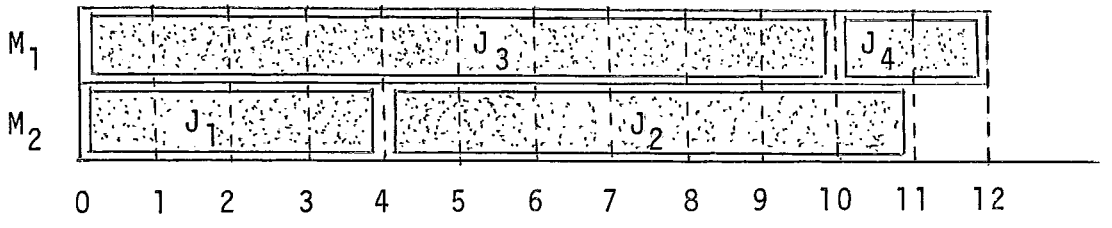


Fig. 4.2 - Diagrama do processamento da distribuição ótima

O problema que ora propomos pode ser considerado um caso particular de $R2//C_{max}$ e uma generalização de $Q2//C_{max}$. Como ambos são NP-completos também o será o problema abordado. Neste capítulo vamos, portanto, desenvolver um algoritmo aproximativo, polinomialmente limitado, para o problema em pauta, com qualidade melhor do que os existentes para $R2//C_{max}$.

4.1.2. Notação

Uma vez que o problema aqui descrito é uma particularização de $R2//C_{max}$, e essa particularização se refere aos tempos de processamento, propomos incluir tal restrição no segundo parâmetro, descrevendo o problema por $R2/q_j = \alpha p_j + \beta/C_{max}$.

4.1.3. Literatura existente

Bruno et al [9] e Lenstra et al [10] mostraram que $P2//C_{max}$ é NP-completo, decorrendo dai que $Q2//C_{max}$ e $R2//C_{max}$ também o são. Para o problema $Q//C_{max}$, Gonzales, Ibarra e Sahni [11] descreveram um algoritmo aproximativo, polinomialmente limitado, conhecido por LTP', que consiste, basicamente, em dispor os jobs em ordem não decrescente dos tempos de processamento, e atribui-los, um por um, à máquina que o terminará mais cedo. Gonzales, Ibarra e Sahni provaram que

$$\frac{C_{max}\ (LTP')}{C_{max}^*} \leq 2 - \frac{2}{m+1}$$

onde C_{max}^* indica o tempo ótimo de processamento, e m representa o número de máquinas. Para m=2, o limite acima se reduz a 4/3.

Para o problema $R//C_{max}$, Ibarra e Kim [12] descreveram 6 algoritmos, cada um deles consistindo essencialmente em alocar o job na máquina que o executa mais rapidamente, e mostraram, para cada um,

$$\frac{C_{max}\ (A)}{C_{max}^*} \leq m$$

onde C_{max} (A) é o tempo de processamento da distribuição obtida pelo algoritmo.

Para o caso $R2//C_{max}$, no entanto, Ibarra e Kim descrevem, no mesmo artigo, um algoritmo G, com complexidade O(n logn) tal que

$$\frac{C_{max}\ (G)}{C_{max}^*} \leq \frac{1 + \sqrt{5}}{2}$$

No presente capítulo, vamos descrever um algoritmo, a que chamamos Algoritmo C, com complexidade $O(n^2)$, com qualidade

$$\frac{C_{max}\ (C)}{C_{max}^*} \leq \frac{3}{2}$$

4.2. O Algoritmo C

4.2.1. Descrição

A estrategia do algoritmo pode ser descrita de forma simples; consiste, basicamente, em:

a) renumerar as máquinas de forma que, se p_j e q_j representarem os tempos de processamento, nas máquinas M_1 e M_2, respectivamente, dos jobs J_j, j=1,2 ,..., n, e $q_j = \alpha p_j + \beta$, tenhamos $|\alpha| \leq 1$

b) ordenar os jobs em ordem não decrescente dos valores p_j, j=1,2 ,..., n.

c) construir e testar todas as distribuições com m jobs consecutivos, m=1,2 ,..., n, na máquina M_1 (e os restantes na máquina M_2), dando saída ao melhor deles.

O Algoritmo C está descrito a seguir, em algol-like, onde $T(M_1)$ e $T(M_2)$ representam os tempos de processamento das máquinas M_1 e M_2, respectivamente, nas distribuições testadas, e C_{max} em cada instante, indica o menor tempo total de processamento, entre as distribuições testadas até aquele instante.

O Algoritmo C

início

renumere as máquinas de forma que $|\alpha| \leq 1$, para $q_j = \alpha p_j + \beta$

ordene os jobs de forma que $p_1 \leq p_2 \leq \ldots \leq p_n$;

$S \leftarrow 0$; $SOMAT \leftarrow \sum_{i=1}^{n} p_i$;

$C_{max} \leftarrow \alpha * SOMAT + n\beta$

para m=1 até n passo 1 faça

início

$S \leftarrow S + p_m$ (testando a distribuição

$T(M_1) \leftarrow S$ $(J_1, J_2, \ldots, J_m$ em

$T(M_2) \leftarrow \alpha * [SOMAT - T(M_1)] + (n-m)\beta$ (M_1

$NC_{max} \leftarrow \max \{T(M_1), T(M_2)\}$

$C_{max} \leftarrow \min \{C_{max}, NC_{max}\}$

para i=m+1 até n passo 1 faça

início

$T(M_1) \leftarrow T(M_1) + p_i - p_{i-m}$ (testando a distribuição

$T(M_2) \leftarrow \alpha * [SOMAT - T(M_1)] + (n-m)\beta$ $(J_{i-m+1}, J_{i-m+2}, \ldots, J_i$

$NC_{max} \leftarrow \max \{T(M_1), T(M_2)\}$ (em M_1

$C_{max} \leftarrow \min \{C_{max}, NC_{max}\}$

fim

fim

escreva C_{max}

fim

4.2.2. Um exemplo

Vamos ilustrar a aplicação do Algoritmo C a um exemplo, com os seguintes dados (por uma questão de simplicidade, os jobs já foram ordenados na forma $p_1 \leq p_2 \leq \ldots \leq p_n$):

		JOBS				
		J_1	J_2	J_3	J_4	J_5
MAQS	M_1	4	6	10	10	18
	M_2	1	2	4	4	8

Fig. 4.3. Tempos de processamento do exemplo proposto

Nesse caso, α=0,5 e β=-1. A solução ótima (única) é a distribuição: jobs J_1 e J_4 em M_1 e os jobs J_2, J_3 e J_5 em M_2, com tempos de processamento $T(M_1) = T(M_2) = 14$, fornecendo, portanto, $C_{max}^* = 14$. Uma vez que esta distribuição não é formada por jobs consecutivos em M_1, não será obtido pelo Algoritmo C. A seguir, a tabela de distribuições testadas pelo Algoritmo, na prdem em que são obtidos.

m	Jobs em M_1	Jobs em M_2	$T(M_1)$	$T(M_2)$	C_{max}
0	-	$J_1\ J_2\ J_3\ J_4\ J_5$	0	19	19
	J_1	$J_2\ J_3\ J_4\ J_5$	4	18	18
	J_2	$J_1\ J_3\ J_4\ J_5$	6	17	17
1	J_3	$J_1\ J_2\ J_4\ J_5$	10	15	15
	J_4	$J_1\ J_2\ J_3\ J_5$	10	15	15
	J_5	$J_1\ J_2\ J_3\ J_4$	18	11	18
	$J_1\ J_2$	$J_3\ J_4\ J_5$	10	16	16
	$J_2 J_3$	$J_1\ J_4\ J_5$	16	13	16
2	$J_3\ J_4$	$J_1\ J_2\ J_5$	20	11	20
	$J_4\ J_5$	$J_1\ J_2\ J_3$	28	7	28
	$J_1\ J_2\ J_3$	$J_4\ J_5$	20	12	20
3	$J_2\ J_3\ J_4$	$J_1\ J_5$	26	9	26
	$J_3\ J_4\ J_5$	$J_1\ J_2$	38	3	38
4	$J_1\ J_2\ J_3\ J_4$	J_5	30	8	30
	$J_2\ J_3\ J_4\ J_5$	J_1	44	1	44
5	$J_1\ J_2\ J_3\ J_4\ J_5$	-	48	0	48

Fig. 4.4 - Tabela de distribuições obtidas pelo Algoritmo C.

A distribuição obtida pelo Algoritmo seria J_3 ou J_4 em M_1, dependendo do critério de desempate, e os jobs restantes em M_2, com $C_{max}(C) = 15$.

4.2.3. A Complexidade

Descrevemos, no quadro abaixo, a complexidade do Algoritmo C, passo a passo; a ordenação dos jobs, a renumeração das máquinas, e a inicialização das variaveis tem, evidentemente, complexidade O(n logn)

Renumere as máquinas Ordene os jobs $S \leftarrow 0$; $SOMAT \leftarrow \sum p_i$ $C_{max} \leftarrow \alpha * SOMAT + n\beta$	O(n logn)
para m=1 até n passo 1 faça $S \leftarrow S + p_m$ $T(M_1) \leftarrow S$ $T(M_2) \leftarrow \alpha * [SOMAT - T(M_1)] + (n-m)\beta$ $NC_{max} \leftarrow \max \{T(M_1), T(M_2)\}$ $C_{max} \leftarrow \min \{C_{max}, NC_{max}\}$	$O(n^2)$
para i=m+1 até n passo 1 faça $T(M_1) \leftarrow T(M_1) + p_i - p_{i-m}$ $T(M_2) \leftarrow \alpha * [SOMAT+T(M_1)]+(n-m)\beta$ $NC_{max} \leftarrow \max \{T(M_1), T(M_2)\}$ $C_{max} \leftarrow \min \{C_{max}, NC_{max}\}$	O(n)

Fig. 4.5. Determinação da complexidade do Algoritmo C

O Algoritmo tem, portanto, complexidade $O(n^2)$.

Uma outra forma de calcular a complexidade do Algoritmo seria a seguinte: o algoritmo constroi

1	distribuição	com	0	jobs	em	M_1
n	"	"	1	"	"	"
n-1	"	"	2	"	"	"
.....						
1	"	"	n	"	"	"

Portanto são construidos

$$1 + 1 + 2 + 3 + \ldots + n = 1 + \frac{n(n+1)}{2} \text{ distribuições.}$$

Como a construção de cada distribuição, a partir do anterior, gasta um tempo constante, temos, para o algoritmo, uma complexidade $O\left(\frac{n(n+1)}{2}\right) = O(n^2)$.

4.2.4. A Qualidade

Neste item vamos medir a qualidade do algoritmo, isto é, encontrar um limite superior para a solução encontrada pelo algoritmo, no pior caso. Fazendo $C_{max}(C)$ representar o tempo de processamento da distribuição obtida pelo Algoritmo C, e deixando C_{max}^* representar a solução ótima, vamos mostrar que

$$\frac{C_{max}(C)}{C_{max}^*} \leq \frac{3}{2}$$

Para tanto, precisamos mostrar os dois lemas seguintes.

Lema 4.1

Seja L uma lista de n números reais $p_1, p_2, \ldots, p_n$, ordenados de forma não decrescente $p_1 \leq p_2 \leq \ldots \leq p_n$. Seja S a soma de $m \leq n$ elementos distintos quaisquer de L. Então, existem $m \leq n$ elementos consecutivos de L, com soma T, tais que

$$|S - T| \leq \frac{1}{2} S$$

Prova

Sejam $L = \{p_1, p_2, \ldots, p_n\}$ e $S = \sum_{i=1}^{m} p_{\sigma(i)}$

Façamos

$t_1 = p_1 + p_2 + \ldots + p_m$

$t_2 = p_2 + p_3 + \ldots + p_{m+1}$

.....

$t_{n-m+1} = p_{n-m+1} + p_{n-m+2} + \ldots + p_n$

Ora, claramente $t_1 \leq S \leq t_{n-m+1}$

Portanto, sejam t_k e t_{k+1} as duas somas consecutivas tais que $t_k \leq S \leq t_{k+1}$ (veja figura 4.6)

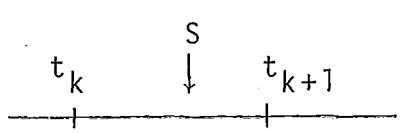


Fig. 4.6. As somas t_k e t_{k+1} são as mais próximas de S.

Mas:

$$t_k = p_k + p_{k+1} + \ldots + p_{k+m-1}$$

$$t_{k+1} = p_{k+1} + p_{k+2} + \ldots + p_{k+m}$$

Logo, $t_{k+1} - t_k = p_{m+k} - p_k$ (4.1)

Por outro lado, chamemos de T o valor t_k ou t_{k+1} mais próximo de S. Vamos mostrar que

$$|S - T| \leq \frac{1}{2} \left[\max_{1 \leq i \leq m} \{p_{\sigma(i)}\} - \min_{i \leq i \leq m} \{p_{\sigma(i)}\} \right]$$

Vamos considerar os dois casos:

1º caso: $S = t_k$ ou $S = t_{k+1}$

Nesse caso, trivialmente, $S = T$, e

$$|S - T| = 0 \leq \frac{1}{2} \left[\max_{1 \leq i \leq m} \{p_{\sigma(i)}\} - \min_{1 \leq i \leq m} \{p_{\sigma(i)}\} \right]$$

2º caso: $t_k < S < t_{k+1}$

Nesse caso,

$$|S - T| \leq \frac{1}{2} (t_{k+1} - t_k)$$

E, por (4.1)

$$|S - T| \leq \frac{1}{2} (p_{k+m} - p_k) \qquad (4.2)$$

Por outro lado,

$S > t_k$ implica em $\max_{1\leq i\leq m} \{p_{\sigma(i)}\} > p_{k+m-1}$, ou seja

$$\max_{1\leq i\leq m} \{p_{\sigma(i)}\} \geq p_{k+m}$$

$S < t_{k+1}$ implica em $\min_{1\leq i\leq m} \{p_{\sigma(i)}\} < p_{k+1}$, ou seja,

$$\min_{1\leq i\leq m} \{p_{\sigma(i)}\} \leq p_k$$

De onde se conclui

$$\max_{1\leq i\leq m} \{p_{\sigma(i)}\} - \min_{1\leq i\leq m} \{p_{\sigma(i)}\} \geq p_{k+m} - p_k$$

Levando em (4.2), obtemos

$$|S - T| \leq \frac{1}{2}\left[\max_{1\leq i\leq m} \{p_{\sigma(i)}\} - \min_{1\leq i\leq m} \{p_{\sigma(i)}\}\right]$$

Finalmente, como

$$\max_{1\leq i\leq m} \{p_{\sigma(i)}\} - \min_{1\leq i\leq m} \{p_{\sigma(i)}\} \leq \max_{1\leq i\leq m} \{p_{\sigma(i)}\} \leq S$$

temos

$$|S - T| \leq \frac{1}{2} S.$$

Lema 4.2

Sejam $\{x_1, x_2, \ldots, x_n\}$ e $\{y_1, y_2, \ldots, y_n\}$ duas listas de n números. Então

$$\max \{x_1, x_2, \ldots, x_n\} - \max \{y_1, y_2, \ldots, y_n\} \leq \max \{x_1-y_1, x_2-y_2, \ldots, x_n-y_n\}$$

Prova

Por indução. Para n=1,

$$\max \{x_1\} - \max \{y_1\} = x_1 - y_1 = \max \{x_1 - y_1\}$$

Suponha verdade para n-1:

$$\max \{x_1, x_2, \ldots, x_{n-1}\} - \max \{y_1, y_2, \ldots, y_{n-1}\} \leq \max \{x_1 - y_1, x_2 - y_2, \ldots, x_{n-1} - y_{n-1}\}$$

Mas

$$\max \{x_1, x_2, \ldots, x_n\} = \max \{\max \{x_1, x_2, \ldots, x_{n-1}\}, x_n\}$$

$$\max \{y_1, y_2, \ldots, y_n\} = \max \{\max \{y_1, y_2, \ldots, y_{n-1}\}, y_n\}$$

Logo,

$$\max \{x_1, x_2, \ldots, x_n\} - \max \{y_1, y_2, \ldots, y_n\} =$$

$$= \max \{\max \{x_1, x_2, \ldots, x_{n-1}\}, x_n\} - \max \{\max \{y_1, y_2, \ldots, y_{n-1}\}, y_n\} \leq$$

$$\leq \max \{\max \{x_1, x_2, \ldots, x_{n-1}\} - \max \{y_1, y_2, \ldots, y_{n-1}\}, x_n - y_n\} \leq$$

$$\leq \max \{x_1 - y_1, x_2 - y_2, \ldots, x_n - y_n\}.$$

Agora podemos mostrar o

Teorema 4.1

Dado um problema $R2/q_j = \alpha p_j + \beta / C_{max}$, seja $C_{max}(C)$ a solução encontrada pelo Algoritmo C, e C_{max}^*, a solução ótima. Então

$$\frac{C_{max}(C)}{C_{max}^*} \leq \frac{3}{2}$$

Prova

Digamos que a distribuição ótima corresponda à colocação, em M_1, dos m primeiros jobs de uma permutação σ, e, em M_2, dos jobs restantes. Representando por $T^*(M_1)$ e $T^*(M_2)$ os tempos de processamento dessa distribuição, em M_1 e M_2, respectivamente, temos:

$$T^*(M_1) = \sum_{i=1}^{m} p_{\sigma(i)} \tag{4.3}$$

$$T^*(M_2) = \sum_{i=1}^{n} q_i - \sum_{i=1}^{m} q_{\sigma(i)} =$$

$$= \sum_{i=1}^{n} (\alpha p_i + \beta) - \sum_{i=1}^{m} (\alpha p_{\sigma(i)} + \beta) =$$

$$= \alpha \left[\sum_{i=1}^{n} p_i - \sum_{i=1}^{m} p_{\sigma(i)} \right] + (n-m)\beta \tag{4.4}$$

E, ainda, $C_{max}^* = \max \{T^*(M_1), T^*(M_2)\}$ (4.5)

Por outro lado, pelo Lema 4.1., sabemos que existem m jobs consecutivos (numa ordenação $p_1 \leq p_2 \leq \ldots \leq p_n$), tais que

$$\left| \sum_{i=1}^{m} p_{\sigma(i)} - \sum_{i=k}^{k+m-1} p_i \right| \leq \frac{1}{2} \sum_{i=1}^{m} p_{\sigma(i)} \tag{4.6}$$

Considere, pois, a distribuição constituida por esses m jobs consecutivos em M_1, e os restantes em M_2. Designado por $T(M_1)$ e $T(M_2)$ os tempos de processamento de M_1 e M_2, respectivamente, temos:

$$T(M_1) = \sum_{i=k}^{k+m-1} p_i \qquad (4.7)$$

$$T(M_2) = \sum_{i=1}^{n} q_i - \sum_{i=k}^{k+m-1} q_i =$$

$$= \sum_{i=1}^{n} (\alpha p_i + \beta) - \sum_{i=k}^{k+m-1} (\alpha p_i + \beta) =$$

$$= \alpha \left[\sum_{i=1}^{n} p_i - \sum_{i=k}^{k+m-1} p_i \right] + (n-m)\beta \qquad (4.8)$$

Ainda, como o Algoritmo C testa essa distribuição, vem

$$C_{max}(C) \leq \max \{T(M_1), T(M_2)\} \qquad (4.9)$$

Subtraindo (4.9) de (4.5):

$$C_{max}(C) - C_{max}^* \leq \max \{T(M_1), T(M_2)\} - \max \{T^*(M_1), T^*(M_2)\}$$

Pelo lema 4.2, vem

$$C_{max}(C) - C_{max}^* \leq \max \{T(M_1) - T^*(M_1), T(M_2) - T^*(M_2)\}$$

Substituindo as expressões dos tempos de processamento, dadas em (4.3), (4.4), (4.7) e (4.8), segue:

$$C_{max}(C) - C_{max}^* \leq \max \left\{ \sum_{i=k}^{k+m-1} p_i - \sum_{i=1}^{m} p_{\sigma(i)}, \alpha \left[\sum_{i=1}^{m} p_{\sigma(i)} - \sum_{i=k}^{k+m-1} p_i \right] \right\}$$

Como $|\alpha| \leq 1$, vem

$$C_{max}(C) - C_{max}^* \leq \max \left\{ \sum_{i=k}^{k+m-1} p_i - \sum_{i=1}^{m} p_{\sigma(i)} , \sum_{i=1}^{m} p_{\sigma(i)} - \sum_{i=k}^{k+m-1} p_i \right\}$$

isto é,

$$C_{max}(C) - C_{max}^* \leq \left| \sum_{i=1}^{m} p_{\sigma(i)} - \sum_{i=k}^{k+m-1} p_i \right|$$

o que, por (4.6), implica em

$$C_{max}(C) - C_{max}^* \leq \frac{1}{2} \sum_{i=1}^{m} p_{\sigma(i)}$$

Finalmente, como $C_{max}^* \geq \sum_{i=1}^{m} p_{\sigma(i)}$, segue-se

$$C_{max}(C) - C_{max}^* \leq \frac{1}{2} C_{max}^*$$

isto é,

$$\frac{C_{max}(C)}{C_{max}^*} \leq \frac{3}{2}$$

4.2.5. Exemplo - limite

Para finalizar esse capítulo, vamos mostrar que o limite

$$\frac{C_{max}(C)}{C_{max}^*} \leq \frac{3}{2}$$

é o melhor possível, para o Algoritmo C, bastando, para isso, exibir um exemplo no qual esse limite é atingido.

Considere um problema $R2/q_j = \alpha p_j + \beta/C_{max}$, com os dados

		JOBS		
		J_1	J_2	J_3
MAQS	M_1	u	2u + a	3u + 2a
	M_2	3u + a	4u + 2a	5u + 3a

Fig. 4.7 - Tempos de processamento do exemplo-limite

Nesse caso, $\alpha=1$ e $\beta = 2u+a$, $u, a \geq 0$

Existem 8 distribuições possíveis, das quais a ótima é J_1 e J_3 em M_1 e J_2 em M_2, com

$$T^*(M_1) = T^*(M_2) = C_{max}^* = 4u + 2a$$

A figura 4.8 mostra as sete restantes, na ordem em que são encontradas pelo Algoritmo C

Jobs em M_1	Jobs em M_2	$T(M_1)$	$T(M_2)$	C_{max}
-	$J_1\ J_2\ J_3$	-	12u + 6a	12u + 6a
J_1	$J_2\ J_3$	u	9u + 5a	9u + 5a
J_2	$J_1\ J_3$	2u + a	8u + 4a	8u + 4a
J_3	$J_2\ J_3$	3u + 2a	7u + 3a	7u + 3a
$J_1\ J_2$	J_3	3u + a	5u + 3a	5u + 3a
$J_2\ J_3$	J_1	5u + 3a	3u + a	5u + 3a
$J_1\ J_2\ J_3$	-	6u + 3a	-	6u + 3a

Fig. 4.8 - Tabelas das distribuições obtidas pelo Algoritmo C, no exemplo-limite

Portanto, $C_{max}(C) = 5u + 3a$

Logo,

$$\frac{C_{max}(C)}{C_{max}^*} = \frac{5u + 3a}{4u + 2a}$$

e $$\lim_{u \to 0} \frac{C_{max}(C)}{C_{max}^*} = \frac{3}{2}.$$

CAPÍTULO V

FLOW-SHOP SCHEDULING

5.1. Considerações Gerais

5.1.1. Descrição dos Problemas

Reportando-nos mais uma vez ao Capítulo II, vimos que existem problemas de Scheduling nos quais as máquinas são ditas "paralelas", e problemas de Scheduling nos quais as máquinas são ditas "em sequencia". No primeiro caso, para que um job seja "executado", é suficiente que seja submetido a uma das máquinas durante um certo intervalo de tempo, chamado tempo de processamento. Nos problemas do segundo caso, no entanto, a "execução" de um job consiste de uma série de operações, cada uma relativa a uma máquina, e às quais o job deve ser submetido; dessa forma, o job, para ser executado, deve "passar" por todas as máquinas. Normalmente, cada máquina só pode executar um job por vez. Um exemplo típico dessa situação é a requisição de um talão de cheques em um banco; a execução consiste de uma sequencia de operações: a recepção, pelo caixa, do requerimento; a procura do talão, já pronto, no arquivo; e seu transporte, do arquivo até as mãos do cliente.

Os problemas de máquinas em sequencia, são, por sua vez, classificados em tres grupos:

a) Open-shop Scheduling: nessa classe de problemas não existe ordem nas máquinas, isto é, a única exigencia é que cada job "passe" por cada uma das máquinas;

b) Job-shop Scheduling: nesse tipo de Scheduling, a cada job está associada uma permutação das máquinas, e cada job deve passar pelas máquinas na ordem indicada pela permutação;

c) Flow-shop Scheduling: nessa terceira classe, a permutação das máquinas é a mesma para todos os jobs, isto é, as máquinas estão ordenadas, na forma M_1, M_2 ,..., M_m, e os jobs devem passar por elas, nessa ordem.

Em um problema de Scheduling, os dados fornecidos formam uma matriz (na qual, por exemplo, o elemento p_{ij} indica o tempo de processamento do job J_j na máquina M_i); e a solução consiste em uma função que fornece, para cada máquina, em cada instante, o job processado.

Normalmente, nos problemas de máquinas em sequencia, cada máquina só pode executar um job por vez, assim como um job não pode ser executado simultaneamente em duas ou mais máquinas (nos casos em que essas restrição não ocorrerem, isso deve ficar explicitado). Em virtude desses fatos, nos problemas de Flow-shop Scheduling, é comum ocorrer que um job, ao terminar seu processamento na máquina M_i, fique esperando a máquina M_{i+1} terminar o processamento do job que o antecedeu, ocasionando a formação de uma "fila de espera" de jobs. Da mesma forma, é possível que ocorra que a máquina M_i, ao terminar o processamento de um job, fique aguardando que a máquina M_{i-1} termine o processamento do job seguinte.

Normalmente, não é permitido que um job "ultrapasse" outro, na fila de espera; nesse caso, o problema recebe

o nome de Permutation Flow Shop Scheduling, e é esse o caso que será estudado nesse trabalho. Com essas restrições, a solução de um problema de Flow Shop Scheduling consiste simplesmente em uma permutação de jobs, isto é, uma ordenação sob a qual os jobs devem ser submetidos às máquinas.

Vamos ilustrar esses fatos através de um exemplo. Considere um problema de Flow shop Scheduling com os dados:

		JOBS			
		J_1	J_2	J_3	J_4
MAQS.	M_1	3	4	3	6
	M_2	5	3	3	3
	M_3	6	2	6	4

Fig. 5.1 - Tempos de processamento de um problema de Flow shop scheduling

O processamento dos jobs na ordem $J_1 - J_2 - J_3 - J_4$ pode ser representado conforme o diagrama abaixo:

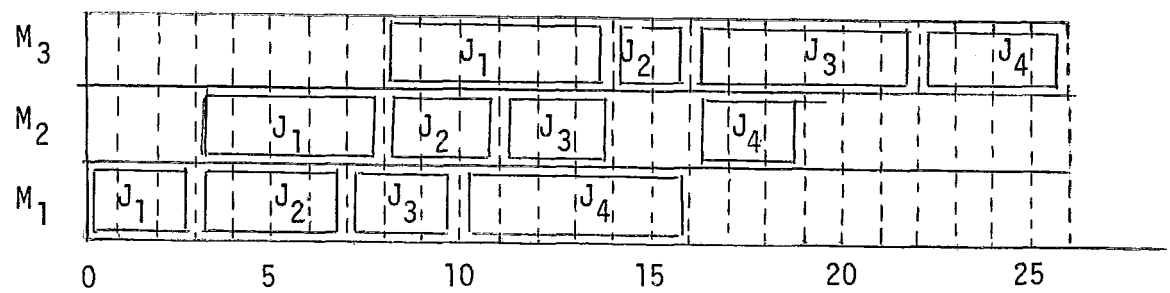


Fig. 5.2 - Diagrama de um processamento de Flow-shop scheduling

Neste exemplo, o tempo total de processamento é dado por $C_{max} = 26$, e a soma dos tempos de processamento, $\sum C_j = 14 + 16 + 22 + 26 = 78$.

5.1.2. A restrição "no wait"

Em algumas aplicações, dependendo da natureza do trabalho executado, pode ser essencial que não sejam formadas, durante o processamento, as chamadas "filas de espera" de jobs; por exemplo, quando se trabalha com mercadorias pereciveis, ou em siderurgia, quando a temperatura do material não pode cair abaixo de certos níveis. Esses fatos justificam a introdução, nos problemas em pauta, de uma restrição chamada "no wait", que estabelece: o instante de iniciar o processamento de um job,

na máquina M_1, deve ser tal que permita que este processamento siga ininterrupto, até o seu termino, na última máquina.

Tal procedimento pode ser ilustrado num problema com os mesmos dados do exemplo anterior (fig. 5.1), inserindo-se, contudo, a condição "no wait". Então, o processamento da sequência $J_1 - J_2 - J_3 - J_4$ ficaria conforme o diagrama da figura 5.3:

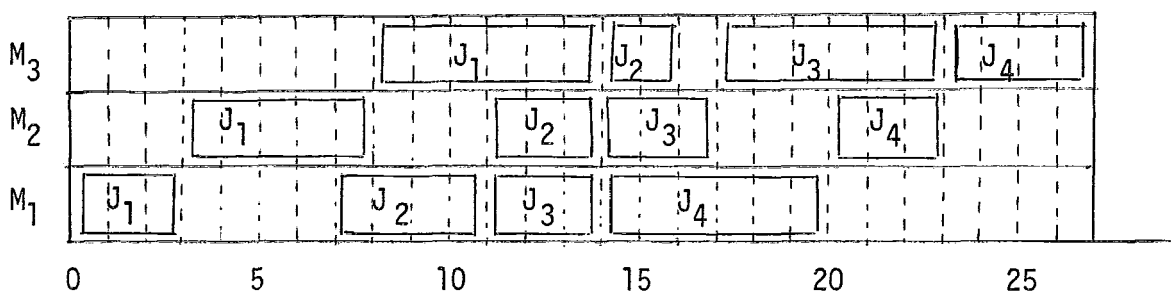


Fig. 5.3 - Diagrama de um processamento de Flow-shop scheduling com a restrição "no wait".

Nesse caso, $C_{max} = 27$ e $\sum C_j = 14 + 17 + 23 + 27 = 81$.

5.1.3. Posição na literatura

Provavelmente, o primeiro artigo que tratou de problemas de Flow-shop Scheduling foi o de Johnson [13], em 1954,

que desenvolveu um algoritmo, com complexidade O(n log n) que resolve, de forma ótima o problema F2 // C_{max}. Mais tarde, Garey et al [14] e Lenstra [10] mostraram que F3 // C_{max}, F2/r_j/C_{max} e F3/tree/C_{max} são NP-completos. Em 1972, Reddi e Ramamoorthy [15], baseados em Gilmore e Gomory [16] construiram um algoritmo com complexidade $O(n^2)$ para F2/no wait/C_{max} e Lenstra [10] mostrou que F/no wait/C_{max} e F/no wait/ΣC_j são NP-Completos. Os problemas F3/no wait/C_{max} e F2/no wait/ΣC_j estão ainda em aberto, apesar do premio oferecido por Lenstra para quem construisse, para eles, algoritmos polinomialmente limitados. Papadimitriou et al [25] mostrou que F4/no wait/C_{max} é NP-Completo.

5.1.4. Os problemas propostos

Neste capítulo vamos descrever as permutações de jobs, para problemas de Flow-shop Scheduling, que minimizam C_{max} e ΣC_j, tanto em problemas sujeitos à restrição "no wait" como para aqueles em que essa restrição não existe, porém, em todos os casos, com uma condição adicional: de que, para cada job, o tempo de processamento seja o mesmo em qualquer máquina. Dito de outra forma, se p_{ij} representa o tempo de processamento do job J_i, i=1,2 ,...,n, na máquina M_j, j=1,2 ,..., m, a restrição exige que

$$p_{i1} = p_{i2} = \ldots = p_{im} \quad \text{para todo} \quad i \in \{1,2 ,\ldots, n\}$$

Com essa restrição, os problemas de Flow-shop Scheduling se tornam polinomialmente limitados, em geral com complexidade O(n log n), complexidade do algoritmo que ordena os jobs segundo a permutação especificada.

Uma vez que, para esse tipo de problema, o tempo de processamento é constante para cada job, sugerimos incluir, na notação, o símbolo "p_i" no segundo parâmetro. Vamos, portanto, analisar os problemas F/no wait,p_i/C_{max}, F/no wait, $p_i/\Sigma C_j$, $F/p_i/C_{max}$ e $F/p_i/\Sigma C_j$.

5.2. O Problema F/no wait, p_i/C_{max}

5.2.1. Definição do problema

O problema apresentado neste item pode ser definido da seguinte forma: dadas m máquinas M_1, M_2 ,..., M_m, e n jobs J_1, J_2 ,..., J_n, com p_i representando o tempo de processamento do job J_i, i=1,2 ,..., n em qualquer máquina, determine a permutação de jobs que minimiza o tempo total de processamento, C_{max}, num processamento do tipo Flow-shop Scheduling, com a restrição "no wait".

Vamos mostrar que qualquer permutação σ, tal que

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(k-1)} \leq p_{\sigma(k)} \geq p_{\sigma(k+1)} \geq \cdots \geq p_{\sigma(n)}$$

para qualquer $k \in \{1,2,\ldots,n\}$, é ótima. Deve-se notar que essa classe de permutações inclui

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)} \quad \text{para} \quad k=n, \quad \text{e}$$

$$p_{\sigma(1)} \geq p_{\sigma(2)} \geq \cdots \geq p_{\sigma(n)} \quad \text{para} \quad k=1.$$

Vamos mostrar ainda, que, para qualquer dessas permutações,

$$C_{max}(\sigma) = C_{max}^* = (m-1) \max_{1 \leq i \leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

Como essas permutações podem ser obtidas com complexidade $O(n \log n)$, essa é a complexidade da resolução do problema.

5.2.2. Cálculo de C_{max}

$C_{max}(\sigma)$, o tempo gasto pela execução dos n jobs nas m máquinas, quando os jobs estão dispostos segundo uma permutação σ, pode ser escrito como a soma de duas parcelas: o tempo em que a máquina M_m esteve operando e o tempo em que esta máquina esteve parada. A primeira parcela é dada por

$$\sum_{i=1}^{n} p_i$$

e independe da permutação; para o cálculo da segunda parcela, façamos $d(\sigma(i-1), \sigma(i))$ designar o tempo decorrido entre o final do processamento do job $J_{\sigma(i-1)}$ na máquina M_m, e o início do processamento, nessa máquina, do job, $J_{\sigma(i)}$ conforme podemos ver na figura 5.4 abaixo

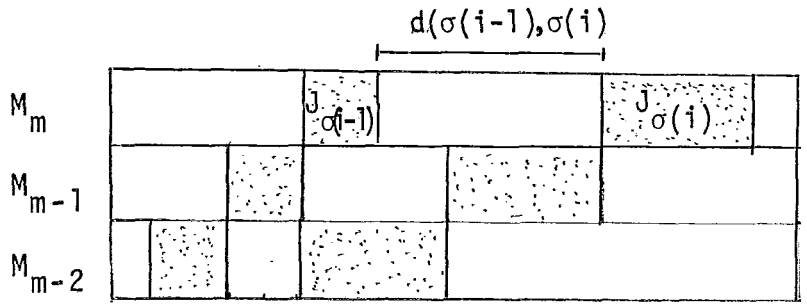


Fig. 5.4. Definição de $d(\sigma(i-1),\sigma(i))$

Admitindo, por conveniência, $p_{\sigma(o)} = 0$, podemos escrever

$$C_{max}(\sigma) = \sum_{i=1}^{n} d(\sigma(i-1), \sigma(i)) + \sum_{i=1}^{n} p_i \qquad (5.1)$$

Claramente, $d(\sigma(o), \sigma(1)) = (m-1)p_{\sigma(1)}$

Para calcular $d(\sigma(i-1), \sigma(i))$ devemos admitir dois casos:

1º caso: $p_{\sigma(i-1)} \leq p_{\sigma(i)}$

Nesse caso, temos a situação mostrada na figura 5.5 a seguir;

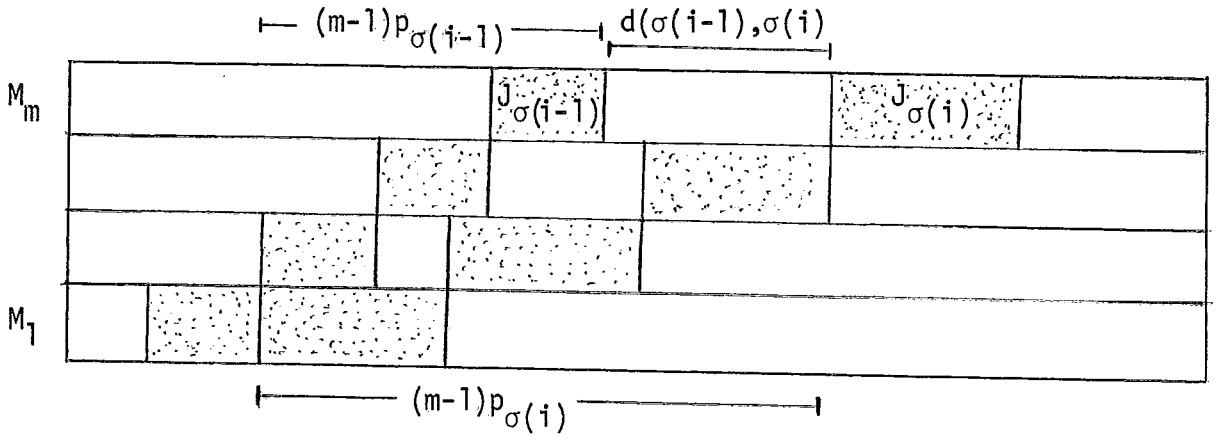


Fig. 5.5. $d(\sigma(i-1),\sigma(i)) = (m-1)(p_{\sigma(i)} - p_{\sigma(i-1)})$

de onde decorre imediatamente,

$$d(\sigma(i-1), \sigma(i)) = (m-1)(p_{\sigma(i)} - p_{\sigma(i-1)})$$

2º caso: $p_{\sigma(i-1)} > p_{\sigma(i)}$

A situação pode ser ilustrada pela figura 5.6,

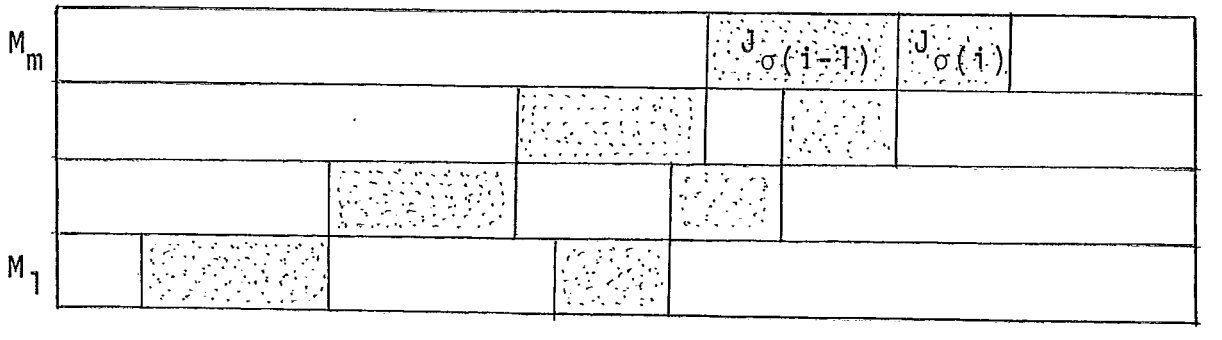


Fig. 5.6. $d(\sigma(i-1),\sigma(i)) = 0$

onde se vê, claramente,

$$d(\sigma(i-1)\ ,\ \sigma(i)) = 0$$

Podemos, então, escrever, para qualquer caso,

$$d(\sigma(i-1)\ ,\ \sigma(i)) = (m-1)\ \max\ \{0,\ p_{\sigma(i)} - p_{\sigma(i-1)}\} \qquad i = 1,2\ ,\ldots,\ n \qquad (5.2)$$

Levando esse resultado em (5.1), temos a seguinte expressão para C_{max}:

$$C_{max}(\sigma) = (m-1) \sum_{i=1}^{n} \max\ \{0,\ p_{\sigma(i)} - p_{\sigma(i-1)}\} + \sum_{i=1}^{n} p_i \qquad (5.3)$$

5.2.3. Um limite inferior para C_{max}

Lema 5.1

$$C_{max}(\sigma) \geq (m-1) \max_{1 \leq i \leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

para qualquer permutação σ.

Prova:

Por (5.3), para provar o Lema 5.1, é suficiente mostrar que

$$\sum_{i=1}^{n} \max\ \{0,\ p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} \qquad (5.4)$$

para qualquer permutação σ, o que pode ser feito por indução.

Para n=1, trivialmente

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} = p_{\sigma(1)} - p_{\sigma(0)} = p_{\sigma(1)}$$

$$\max_{1 \leq i \leq 1} \{p_{\sigma(i)}\} = p_{\sigma(1)}$$

Suponha, agora, que a tese seja verdadeira para n=1, isto é, suponha que

$$\sum_{i=1}^{n-1} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1 \leq i \leq n-1} \{p_{\sigma(i)}\}$$

somando-se $\max \{0, p_{\sigma(n)} - p_{\sigma(n-1)}\}$ de cada lado, vem

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1 \leq i \leq n-1} \{p_{\sigma(i)}\} + \max \{0, p_{\sigma(n)} - p_{\sigma(n-1)}\}$$

Consideremos, então, as duas hipóteses possíveis:

<u>1ª hipótese</u>: $p_{\sigma(n)} \geq \max_{1 \leq i \leq n-1} \{p_{\sigma(i)}\}$

Nesse caso,

$$\max_{1 \leq i \leq n} \{p_{\sigma(i)}\} = p_{\sigma(n)} \quad \text{e}$$

$$\max \{0, p_{\sigma(n)} - p_{\sigma(n-1)}\} = p_{\sigma(n)} - p_{\sigma(n-1)}$$

Logo

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1 \leq i \leq n-1} \{p_{\sigma(i)}\} + \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} - p_{\sigma(n-1)}$$

Como

$$\max_{1\leq i\leq n-1} \{p_{\sigma(i)}\} - p_{\sigma(n-1)} \geq 0$$

segue-se

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1\leq i\leq n} \{p_{\sigma(i)}\}$$

<u>2ª hipótese</u>: $p_{\sigma(n)} < \max_{1\leq i\leq n-1} \{p_{\sigma(i)}\}$

Nesse caso,

$$\max_{1\leq i\leq n-1} \{p_{\sigma(i)}\} = \max_{1\leq i\leq n} \{p_{\sigma(i)}\}$$

Logo

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1\leq i\leq n} \{p_{\sigma(i)}\} + \max \{0, p_{\sigma(n)} - p_{\sigma(n-1)}\}$$

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1\leq i\leq n} \{p_{\sigma(i)}\}$$

Finalmente, como

$$\max_{1\leq i\leq n} \{p_{\sigma(i)}\} = \max_{1\leq i\leq n} \{p_i\}$$

para qualquer permutação σ, a tese segue-se imediatamente.

5.2.4. A permutação ótima

Neste item, vamos mostrar que, em um problema

F/no wait, p_i/C_{max}, a solução ótima corresponde a qualquer permutação σ tal que

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(k-1)} \leq p_{\sigma(k)} \geq p_{\sigma(k+1)} \geq \cdots \geq p_{\sigma(n)}$$

para qualquer $k \in \{1,2,\ldots,n\}$; e que qualquer permutação diferente dessas corresponde a uma solução não ótima.

Para simplicidade de notação, dizemos que uma permutação σ satisfaz à Condição 1 se

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(k-1)} \leq p_{\sigma(k)} \geq p_{\sigma(k+1)} \geq \cdots \geq p_{\sigma(n)}$$

para algum $k \in \{1,2,\ldots,n\}$. E, uma permutação σ satisfaz à Condição 2, caso contrário, isto é, se existirem $r<k$ e $s>k$, $k \in \{2,3,\ldots,n-1\}$ tais que

$$p_{\sigma(r)} > p_{\sigma(k)} \quad e \quad p_{\sigma(k)} < p_{\sigma(s)} .$$

Lema 5.2

Se σ satisfaz a Condição 1, então

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(1)} - p_{\sigma(i-1)}\} = \max_{1 \leq i \leq n} \{p_i\}$$

Prova:

Por hipótese,

$$\max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} = p_{\sigma(i)} - p_{\sigma(i-1)} \quad \forall\, i \leq k \quad e$$

$$\max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} = 0 \qquad\qquad \forall\, i > k$$

Logo

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} = p_{\sigma(1)} - p_{\sigma(2)} - p_{\sigma(1)} + p_{\sigma(2)} - \ldots + p_{(k)} - p_{\sigma(k-1)}$$

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} = p_{\sigma(k)}$$

Mas, ainda por hipótese,

$$p_{\sigma(k)} = \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} = \max_{1 \leq i \leq n} \{p_i\}$$

Portanto

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} = \max_{1 \leq i \leq n} \{p_i\}$$

Levando esse resultado no Lema 5.1, podemos concluir que, se uma permutação σ satisfaz a Condição 1,

$$C_{max}(\sigma) = (m-1) \max_{1 \leq i \leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

e esse resultado é o melhor possível.

Lema 5.3

Se uma permutação satisfizer à Condição 2, então

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} > \max_{1 \leq i \leq n} \{p_i\}$$

Prova:

Se σ satisfizer à Condição 2, isto é, se existirem $r<k$ e $s>k$ com $p_{\sigma(r)} > p_{\sigma(k)}$ e $p_{\sigma(k)} < p_{\sigma(s)}$ para algum $k \in \{2,3,\ldots,n-1\}$, podemos afirmar imediatamente que

$$p_{\sigma(k)} < \max_{1\leq i\leq k-1} \{p_{\sigma(i)}\} \qquad (5.5) \quad \text{e}$$

$$p_{\sigma(k)} < \max_{k+1<i<n} \{p_{\sigma(i)}\} \qquad (5.6)$$

Por outro lado, levando em conta que

$$\max \{0, p_{\sigma(k)} - p_{\sigma(k-1)}\} \geq 0$$

$$\max \{0, p_{\sigma(k+1)} - p_{\sigma(k)}\} \geq p_{\sigma(k+1)} - p_{\sigma(k)}$$

podemos escrever

$$\sum_{i=1}^{n} \max \{0, p_{(i)} - p_{(i-1)}\} \geq \sum_{i=1}^{k-1} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} + p_{\sigma(k+1)} - p_{\sigma(k)} + \sum_{i=k+2}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \qquad (5.7)$$

De (5.4) sabemos que

$$\sum_{i=1}^{k-1} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1\leq i\leq k-1} \{p_{\sigma(i)}\} \quad \text{e}$$

$$p_{\sigma(k+1)} + \sum_{i=k+2}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{k+1\leq i\leq n} \{p_{\sigma(i)}\}$$

substituindo em (5.7), vem

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \geq \max_{1 \leq i \leq k-1} \{p_{\sigma(i)}\} - p_{\sigma(k)} + \max_{k+1 \leq i \leq n} \{p_{\sigma(i)}\} \qquad (5.8)$$

Ora, de (5.5) e (5.6) temos que

$$\max_{1 \leq i \leq n} \{p_{\sigma(i)}\} = \max \{ \max_{1 \leq i \leq k-1} \{p_{\sigma(i)}\}, \max_{k+1 \leq i \leq n} \{p_{(i)}\} \}$$

substituindo-se o maior deles por $\max_{1 \leq i \leq n} \{p_{\sigma(i)}\}$ em (5.8) e efetuando a subtração do menor com $p_{\sigma(k)}$ (que é positiva, por (5.5) e (5.6)), segue-se, de (5.8):

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} > \max_{1 \leq i \leq n} \{p_{\sigma(i)}\}$$

ou seja

$$\sum_{i=1}^{n} \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} > \max_{1 \leq i \leq n} \{p_i\}$$

Levando esse resultado no Lema 5.1, podemos concluir que, se a permutação σ satisfizer a Condição 2, então

$$C_{max}(\sigma) > (m-1) \max_{1 \leq i \leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

Resumindo, podemos afirmar que o problema F/no wait, p_i/C_{max} tem

$$C_{max}^{*} = (m-1) \max_{1 \leq i \leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

e esse resultado corresponde às permutações σ que satisfazem à Condição 1. O problema pode ser resolvido, portanto, com complexidade $O(n \log n)$.

5.3. O Problema F/no wait, $p_i/\sum C_j$

5.3.1. Definição do problema

Da mesma forma que no item anterior, vamos examinar um problema de Flow-shop Scheduling, com n jobs e m máquinas, com p_i representando o tempo de processamento do job J_i em qualquer máquina, ainda com a restrição "no wait", porem desta vez, procurando minimizar a soma dos tempos de processamento.

Vamos mostrar que a soma ótima dos tempos de processamento, $\sum C_j^*$, vale

$$\sum C_j^* = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} p_i$$

e que esse resultado corresponde a uma permutação σ tal que

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

Da mesma forma que no item anterior, a complexidade da resolução desse problema é $O(n \log n)$

5.3.2. Uma expressão para $\sum C_j$

Na figura 5.7, abaixo, vê-se claramente que

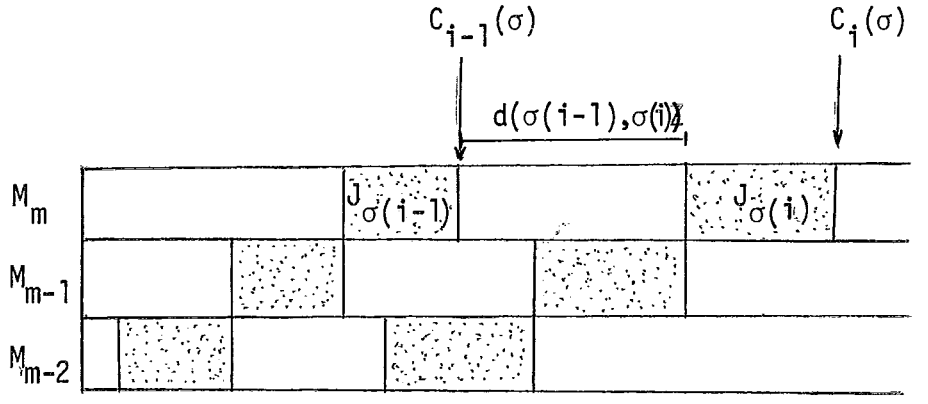


Fig. 5.7. Cálculo de $C_i(\sigma)$

$$C_i(\sigma) = C_{i-1}(\sigma) + d(\sigma_{(i-1)}, \sigma_{(i)}) + p_{\sigma(i)} \quad , \quad i=2,3 \ , \ldots , \ n$$

Levando em conta que

$$C_{\sigma(1)} = m \ p_{\sigma(1)}$$

e que, por (5.2), convencionando $p_{\sigma(o)} = 0$,

$$d(\sigma_{(i-1)},\sigma_{(i)}) = (m-1) \max \{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} \ , \ i=1,2 \ , \ldots , \ n,$$

podemos escrever

$$C_1(\sigma) = p_{\sigma(1)} + (m-1)\, p_{\sigma(1)}$$

$$C_2(\sigma) = p_{\sigma(1)} + p_{\sigma(2)} + (m-1)\left[p_{\sigma(1)} + \max\{0, p_{\sigma(2)} - p_{\sigma(1)}\}\right]$$

.....

$$C_k(\sigma) = \sum_{i=1}^{k} p_{\sigma(i)} + (m-1)\left[p_{\sigma(1)} + \max\{0, p_{\sigma(2)} - p_{\sigma(1)}\} + \ldots + \max\{0, p_{\sigma(k)} - p_{\sigma(k-1)}\}\right]$$

.....

$$C_n(\sigma) = \sum_{i=1}^{n} p_{\sigma(i)} + (m-1)\left[p_{\sigma(1)} + \max\{0, p_{\sigma(2)} - p_{\sigma(1)}\} + \ldots + \max\{0, p_{\sigma(n)} - p_{\sigma(n-1)}\}\right]$$

somando membro a membro, e lembrando que $p_{\sigma(0)} = 0$, vem

$$\sum_{i=1}^{n} C_i(\sigma) = \sum_{i=1}^{1} p_{\sigma(i)} + \sum_{i=1}^{2} p_{\sigma(i)} + \cdots + \sum_{i=1}^{n} p_{\sigma(i)} + (m-1)\left[\sum_{i=1}^{1} \max\{0, p_{\sigma(i)} - p_{\sigma(i-1)}\} + \ldots + \sum_{i=1}^{n} \max\{0, p_{\sigma(i)} - p_{\sigma(i-1)}\}\right]$$

ou, ainda,

$$\sum_{i=1}^{n} C_1(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} \sum_{k=1}^{i} \max\{0, p_{\sigma(k)} - p_{\sigma(k-1)}\}$$

Por conveniência de notação, façamos

$$f_n(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} \tag{5.9}$$

$$g_n(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} \max\{0, p_{\sigma(k)} - p_{\sigma(k-1)}\} \tag{5.10}$$

Em sequencia, vamos mostrar que a permutação σ que satisfaz

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

é a única permutação que minimiza f_n e também é a única permutação que minimiza g_n.

5.3.3. A permutação ótima para f_n

Lema 5.4

Considere duas permutações, σ e Π, nas seguintes condições:

a) $p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$

b) existe $j \in \{1,2,\ldots,n-1\}$ tal que $p_{\Pi(j)} > p_{\Pi(j+1)}$

Então, $f_n(\sigma) < f_n(\Pi)$

Prova

De (5.9),

$$f_n(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)}$$

ou seja

$$f_n(\sigma) = \sum_{k=1}^{1} p_{\sigma(k)} + \sum_{k=1}^{2} p_{\sigma(k)} + \cdots + \sum_{k=1}^{n} p_{\sigma(k)}$$

De imediato,

$$\sum_{k=1}^{j} p_{\sigma(k)} < \sum_{k=1}^{j} p_{\Pi(k)}$$

pois o primeiro é a soma dos j menores elementos de $\{p_1, p_2, \ldots, p_n\}$, e o segundo não é. Por outro lado, ainda pelo fato de que $\sum p_{\sigma(k)}$ é sempre a soma dos menores elementos,

$$\sum_{k=1}^{r} p_{\sigma(k)} \leq \sum_{k=1}^{r} p_{\Pi(k)} \quad , \quad r=1,2,\ldots, n$$

Decorre imediatamente

$$\sum_{k=1}^{1} p_{\sigma(k)} + \sum_{k=1}^{2} p_{\sigma(k)} + \ldots + \sum_{k=1}^{n} p_{\sigma(k)} < \sum_{k=1}^{1} p_{\Pi(k)} + \sum_{k=1}^{2} p_{\Pi(k)} + \ldots + \sum_{k=1}^{n} p_{\Pi(k)}$$

ou seja,

$$f_n(\sigma) < f_n(\Pi).$$

5.3.4. A permutação ótima para g_n

O Lema 5.5, em seguida, fornece um limite inferior para g_n. Logo após, vamos mostrar que esse limite é atingido quando (Lema 5.6) e só quando (Lema 5.7) a permutação σ é tal que

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

Lema 5.5

$$g_n(\sigma) \geq \sum_{i=1}^{n} p_i \quad \text{para qualquer permutação } \sigma$$

Prova

Por (5.10)

$$g_n(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} \max \{0, p_{\sigma(k)} - p_{\sigma(k-1)}\}$$

Como $\max \{0, p_{\sigma(k)} - p_{\sigma(k-1)}\} \geq p_{\sigma(k)} - p_{\sigma(k-1)}$

podemos escrever

$$g_n(\sigma) \geq \sum_{i=1}^{n} \sum_{k=1}^{i} (p_{\sigma(k)} - p_{\sigma(k-1)})$$

$$g_n(\sigma) \geq \sum_{i=1}^{n} \left(\sum_{k=1}^{i} p_{\sigma(k)} - \sum_{k=1}^{i} p_{\sigma(k-1)} \right)$$

$$g_n(\sigma) \geq \sum_{i=1}^{n} p_{\sigma(i)}$$

$$g_n(\sigma) \geq \sum_{i=1}^{n} p_i \quad \text{qualquer que seja a permutação } \sigma.$$

Lema 5.6

Se a permutação σ for tal que

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

então $$g_n(\sigma) = \sum_{i=1}^{n} p_i$$

Prova

Como $p_{\sigma(k-1)} \leq p_{\sigma(k)}$ para todo $k=1,2,\ldots,n$, segue-se

$$\max \{0, p_{\sigma(k)} - p_{\sigma(k-1)}\} = p_{\sigma(k)} - p_{\sigma(k-1)}$$

Logo, podemos escrever

$$g_n(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} (p_{\sigma(k)} - p_{\sigma(k-1)})$$

$$g_n(\sigma) = \sum_{i=1}^{n} \left(\sum_{k=1}^{i} p_{\sigma(k)} - \sum_{k=1}^{i} p_{\sigma(k-1)} \right)$$

$$g_n(\sigma) = \sum_{i=1}^{n} p_{\sigma(i)}$$

$$g_n(\sigma) = \sum_{i=1}^{n} p_i$$

Lema 5.7

Se a permutação σ for tal que existe $j \in \{2,3..,n\}$ tal que $p_{\sigma(j-1)} > p_{\sigma(j)}$, então

$$g_n(\sigma) > \sum_{i=1}^{n} p_i$$

Prova

Por (5.10),

$$g_n(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} \max \{0, p_{\sigma(k)} - p_{\sigma(k-1)}\}$$

Sabemos, por (5.4), que

$$\sum_{k=1}^{i} \max \{0, p_{\sigma(k)} - p_{\sigma(k-1)}\} \geq \max_{1 \leq k \leq i} \{p_{\sigma(k)}\}$$

para qualquer permutação σ; logo

$$g_n(\sigma) \geq \sum_{i=1}^{n} \max_{1 \leq k \leq i} \{p_{\sigma(k)}\}$$

ou ainda,

$$g_n(\sigma) \geq \sum_{i=1}^{j-1} \max_{1 \leq k \leq i} \{p_{\sigma(k)}\} + \max_{1 \leq k \leq j} \{p_{\sigma(k)}\} + \sum_{i=j+1}^{n} \max_{1 \leq k \leq i} \{p_{\sigma(k)}\}$$

Mas, claramente

$$\max_{1 \leq k \leq i} \{p_{\sigma(k)}\} \geq \max_{r \leq k \leq i} \{p_{\sigma(k)}\}$$

para todo r tal que $i \geq r \geq 1$

Logo,

$$g_n(\sigma) \geq \sum_{i=1}^{j=1} \max_{1 \leq k \leq i} \{p_{\sigma(k)}\} + \max_{1 \leq k \leq j} \{p_{\sigma(k)}\} + \sum_{i=j+1}^{n} \max_{j+1 \leq k \leq i} \{p_{\sigma(k)}\}$$

Além disso, como

$$\sum_{i=1}^{j-1} \max_{1 \leq k \leq i} \{p_{\sigma(k)}\} \geq \sum_{i=1}^{j-1} p_{\sigma(i)}$$

$$\sum_{i=j+1}^{n} \max_{j+1 \leq k \leq i} \{p_{\sigma(k)}\} \geq \sum_{i=j+1}^{n} p_{\sigma(i)} ,$$

vem

$$g_n(\sigma) \geq \sum_{i=1}^{j-1} p_{\sigma(i)} + \max_{1 \leq k \leq j} \{p_{\sigma(k)}\} + \sum_{i=j+1}^{n} p_{\sigma(i)}$$

Finalmente, como, por hipótese, $p_{\sigma(j-1)} > p_{\sigma(j)}$, segue-se

$$\max_{1 \leq k \leq j} \{p_{\sigma(k)}\} > p_{\sigma(j)}$$

decorrendo, imediatamente

$$g_n(\sigma) > \sum_{i=1}^{j-1} p_{\sigma(i)} + p_{\sigma(j)} + \sum_{i=j+1}^{n} p_{\sigma(i)}$$

isto é,

$$g_n(\sigma) > \sum_{i=1}^{n} p_{\sigma(i)}$$

ou, ainda

$$g_n(\sigma) > \sum_{i=1}^{n} p_i$$

Resumindo, podemos afirmar que o problema F/no wait, $p_i/\sum C_j$ tem

$$\sum C_j^* = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} p_i$$

e esse resultado corresponde a uma permutação σ que satisfaz a condição

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

Como essa permutação pode ser obtida em $O(n \log n)$, essa é a complexidade da solução do problema.

5.4. O Problema $F/p_i/C_{max}$

5.4.1. Definição do problema

Neste item vamos estudar o problema de Flow-shop Scheduling, com n jobs e m máquinas, onde p_i representa o tempo de processamento do job J_i em qualquer máquina, e no qual procuramos minimizar o tempo total de processamento, como no item 5.2, porém, desta vez, sem a restrição "no wait".

Vamos mostrar que, qualquer que seja a permutação sob a qual dispomos os jobs, o tempo total de processamento, C_{max}, será sempre o mesmo

$$C_{max} = (m-1) \max_{1 \leq i \leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

o que dá para a solução desse problema, a complexidade $O(n)$. Este resultado foi também obtido por Chin et al [24], de forma independente.

5.4.2. Cálculo de C_{max}

Para simplificar a notação, façamos $T(k,i)$ o instante em que termina o processamento do job J_i na máquina M_k. Dessa forma, $C_{max}(\sigma) = T(m,\sigma(n))$ é o instante do término do processamento de n jobs, dispostos segundo uma permutação σ. Dudek e Teuton [17] forneceram uma relação de recorrencia

para o cálculo de $T(k,\sigma(i))$,

$$T(k,\sigma(i)) = \max \{T(k-1,\sigma(i)) \ , \ T(k,\sigma(i-1))\} + p_{k,\sigma(i)} \ ,$$

$$k=1,2 \ ,\ldots, \ m$$

$$i=1,2 \ ,\ldots, \ n$$

$$T(0,\sigma(i)) = 0 \quad i=1,2 \ ,\ldots, \ n$$

$$T(k,\sigma(0)) = 0 \quad k=1,2 \ ,\ldots, \ m$$

para o problema de Flow-shop Scheduling com tempos de processamento quaisquer; na expressão acima, $p_{k,\sigma(i)}$ representa o tempo de processamento do job $J_{\sigma(i)}$ na máquina M_k. Adaptando tal expressão para o problema em pauta, vem

$$T(k,\sigma(i)) = \max \{T(k-1,\sigma(i)) \ , \ T(k,\sigma(i-1))\} + p_i \quad , \ k=1,2,\ldots,m \quad i=1,2,\ldots,n$$

$$T(0,\sigma(i)) = 0 \quad i=1,2 \ ,\ldots, \ n$$

$$T(k,\sigma(0)) = 0 \quad k=1,2 \ ,\ldots, \ m \qquad (5.11)$$

Como o auxílio dessa expressão, vamos calcular, no Lema 5.8, o valor de $C_{max} = T(m,\sigma(n))$.

Lema 5.8

$$T(m,\sigma(n)) = (m-1) \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n} p_{\sigma(i)}$$

para qualquer permutação σ.

Prova

Por indução em m e n.

Para m=1

$$T(1,\sigma(j)) = \sum_{i=1}^{j} p_{\sigma(i)} \quad , \quad j=1,2 \ ,\ldots\ n$$

o que é trivialmente verdadeiro.

Para n=1

$$T(k,\sigma(1)) = (k-1) \max_{1\leq i\leq 1} \{p_{\sigma(i)}\} + \sum_{i=1}^{1} p_{\sigma(i)} = k\, p_{\sigma(1)} \quad , \quad k=1,2 \ ,\ldots,\ m$$

o que também é trivialmente verdadeiro.

Suponha agora que o lema seja verdadeiro para m-1 e n-1, isto é, suponha que

$$T(m-1,\sigma(n)) = (m-2) \max_{1\leq i\leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n} p_{\sigma(i)} \tag{5.12}$$

$$T(m,\sigma(n-1)) = (m-1) \max_{1\leq i\leq n-1} \{p_{\sigma(i)}\} + \sum_{i=1}^{n-1} p_{\sigma(i)} \tag{5.13}$$

De (5.11) temos

$$T(m,\sigma(n)) = \max \{T(m-1,\sigma(n)) \ , \ T(m,\sigma(n-1))\} + p_{\sigma(n)} \tag{5.14}$$

Consideremos as duas hipóteses:

<u>1ª hipótese</u>: $P_{\sigma(n)} \geq \max_{1\leq i\leq n} \{p_{\sigma(i)}\}$

Isso implica em

$$p_{\sigma(n)} = \max_{1\leq i\leq n} \{p_{\sigma(i)}\} \geq \max_{1\leq i\leq n-1} \{p_{\sigma(i)}\}$$

Então (5.12) pode ser escrito

$$T(m-1,\sigma(n)) = (m-2) \max_{1\leq i\leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n-1} p_{\sigma(i)} + \max_{1\leq i\leq n} p_{\sigma(i)}$$

$$T(m-1,\sigma(n)) = (m-1) \max_{1\leq i\leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n-1} p_{\sigma(i)} \qquad (5.15)$$

Comparando com (5.13) segue-se

$$T(m-1,\sigma(n)) \geq T(m,\sigma(n-1))$$

Utilizando-se (5.14) vem

$$T(m,\sigma(n)) = T(m-1,\sigma(n)) + p_{\sigma(n)}$$

De (5.15);

$$T(m,\sigma(n)) = (m-1) \max_{1\leq i\leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n-1} p_{\sigma(i)} + p_{\sigma(n)}$$

$$T(m,\sigma(n)) = (m-1) \max_{1\leq i\leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n} p_{\sigma(i)}$$

<u>2ª hipótese</u>: $p_{\sigma(n)} < \max_{1\leq i\leq n-1} \{p_{\sigma(i)}\}$

Isso implica em

$$\max_{1\leq i\leq n-1} \{p_{\sigma(i)}\} = \max_{1\leq i\leq n} \{p_{\sigma(i)}\} > p_{\sigma(n)}$$

Então (5.13) pode ser escrito

$$T(m,\sigma(n-1)) = (m-1) \max_{1\leq i\leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n-1} p_{\sigma(i)} \qquad (5.16)$$

$$T(m, \sigma(n-1)) = (m-2) \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n-1} p_{\sigma(i)} + \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} \quad (5.17)$$

Como $p_{\sigma(n)} < \max_{1 \leq i \leq n} \{p_{\sigma(i)}\}$ segue-se

$$\sum_{i=1}^{n-1} p_{\sigma(i)} + \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} > \sum_{i=1}^{n} p_{\sigma(i)}$$

decorrendo, de (5.12) e (5.17)

$$T(m,\sigma(n-1)) > T(m-1, \sigma(n))$$

Levando esse resultado em (5.14), vem

$$T(m,\sigma(n)) = T(m,\sigma(n-1)) + p_{\sigma(n)}$$

De (5.16):

$$T(m,\sigma(n)) = (m-1) \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n-1} p_{\sigma(i)} + p_{\sigma(n)}$$

$$T(m,\sigma(n)) = (m-1) \max_{1 \leq i \leq n} \{p_{\sigma(i)}\} + \sum_{i=1}^{n} p_{\sigma(i)}$$

Como esse resultado foi encontrado sem que fosse exigida qualquer restrição à permutação σ, segue-se que ele é válido para qualquer permutação σ, encerrando a demonstração.

Corolário

$$C_{max}^{*} = (m-1) \max_{1 \leq i \leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

Prova

De fato, como

$$\max_{1\leq i\leq n} \{p_{\sigma(i)}\} = \max_{1\leq i\leq n} \{p_i\}$$

$$\sum_{i=1}^{n} p_{\sigma(i)} = \sum_{i=1}^{n} p_i$$

e como $C_{max}(\sigma) = T(m,\sigma(n))$, segue-se, do Lema 5.8, que

$$T(m,\sigma(n)) = (m-1) \max_{1\leq i\leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

resultado esse verdadeiro para qualquer permutação σ, seguindo-se imediatamente

$$C_{max}(\sigma) = C_{max}^* = (m-1) \max_{1\leq i\leq n} \{p_i\} + \sum_{i=1}^{n} p_i$$

Então o problema $F/p_i/C_{max}$ pode ser resolvido com complexidade $O(n)$.

5.5. O Problema $F/p_i/\sum C_j$

5.5.1. Definição do problema

Como no item anterior, vamos estudar o problema de Flow-shop Scheduling, com n jobs e m máquinas, onde p_i representa o tempo de processamento do job J_i em qualquer máquina, também sem a restrição "no wait", procurando minimizar, desta vez, a soma dos tempos de processamento, $\sum C_j$.

Vamos mostrar que, nesse caso, a solução ótima é exatamente a mesma que para o problema F/nowait, $p_i/\sum C_j$, isto é,

$$\sum C_j^* = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} p_i$$

valor correspondente unicamente a uma permutação σ tal que

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

5.5.2. Um limite inferior para $\sum C_j$

O Lema 5.9, nos fornece um limite inferior para $\sum C_j$

Lema 5.9

$$\sum_{i=1}^{n} C_i(\sigma) \geq \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} p_i$$

qualquer que seja a permutação σ.

Prova

Segundo a notação utilizada no item 5.4, vem

$$\sum_{i=1}^{n} C_i(\sigma) = \sum_{i=1}^{n} T(m,\sigma(i))$$

Mas, de (5.11)

$$T(m,\sigma(i)) = \max \{T(m-1,\sigma(i)) , T(m,\sigma(i-1))\} + p_{\sigma(i)} , \quad i=1,2,\ldots, n$$

de onde decorre

$$\sum_{i=1}^{n} C_i(\sigma) = \sum_{i=1}^{n} \max \{T(m-1,\sigma(i)) , T(m,\sigma(i-1))\} + \sum_{i=1}^{n} p_i$$

Podemos, portanto, escrever

$$\sum_{i=1}^{n} C_i(\sigma) \geq \sum_{i=1}^{n} T(m-1,\sigma(i)) + \sum_{i=1}^{n} p_i \qquad (5.18)$$

Por outro lado, do Lema 5.8, vem

$$T(m-1,\sigma(i)) = (m-2) \max_{1 \leq k \leq i} \{p_{\sigma(k)}\} + \sum_{k=1}^{i} p_{\sigma(k)}$$

o que implica em

$$T(m-1,\sigma(i)) \geq (m-2)\, p_{\sigma(i)} + \sum_{k=1}^{i} p_{\sigma(k)}$$

Substituindo-se em (5.18) segue-se

$$\sum_{i=1}^{n} C_i(\sigma) \geq \sum_{i=1}^{n} \left[(m-2)\, p_{\sigma(i)} + \sum_{k=1}^{i} p_{\sigma(k)}\right] + \sum_{i=1}^{n} p_i$$

$$\sum_{i=1}^{n} C_i(\sigma) \geq \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} p_i$$

5.5.3. A permutação ótima

Lema 5.10

Se a permutação σ for tal que

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

então

$$\sum_{i=1}^{n} C_i(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} p_i$$

e essa é a única permutação que fornece esse resultado.

Prova

De fato, se

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

o processamento dos jobs obedece à condição "no wait", e já vimos (item 5.3) que, nesse caso,

$$\sum_{i=1}^{n} C_i(\sigma) = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} p_i$$

Por outro lado, pelo Lema 5.4, vimos também que a permutação σ nas condições da hipótese é a única que minimiza

$$\sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)}$$

Concluimos, portanto, que o problema $F/p_i/\sum C_j$ tem solução

$$\sum C_j^* = \sum_{i=1}^{n} \sum_{k=1}^{i} p_{\sigma(k)} + (m-1) \sum_{i=1}^{n} p_i$$

e essa solução corresponde a uma permutação σ tal que

$$p_{\sigma(1)} \leq p_{\sigma(2)} \leq \cdots \leq p_{\sigma(n)}$$

decorrendo, portanto, que o problema pode ser resolvido com complexidade $O(n \log n)$.

CAPÍTULO VI

CONCLUSÕES

Para encerrar este trabalho, gostaríamos de apresentar, neste último capítulo, sugestões de linhas de pesquisas, que visassem, principalmente, uma generalização dos resultados aqui obtidos.

No capítulo III, foi fornecido um novo limite inferior para o tempo total de processamento, nos problemas $R//C_{max}$, com m máquinas e n jobs

$$C_{max}^{*} \geq \frac{n}{\sum_{j=1}^{m} \frac{1}{\min_{i} \{p_{ij}\}}}$$

limite esse que veio se juntar ao já conhecido

$$C_{max}^{*} \geq \frac{1}{m} \sum_{i=1}^{n} \min_{j} \{p_{ij}\}$$

Com a utilização desse segundo limite foram construídos algoritmos (Ibarra e Kim [12]), com tempo de processamento C_{max}, tais que

$$\frac{C_{max}}{C_{max}^{*}} \leq m \qquad (6.1)$$

Nossa sugestão é no sentido de aproveitar também o primeiro limite para a construção de algoritmos para $R//C_{max}$, com qualidade melhor que a descrita em 6.1.

No Capítulo IV foi mostrado que, se num problema $R2//C_{max}$, a relação entre os tempos de processamento de cada job J_i nas duas máquinas, p_i e q_i, for tal que, num gráfico cartesiano, os pontos (p_i,q_i) se situarem sobre uma reta ($q_i = \alpha p_i + \beta$), então é possível construir um algoritmo com qualidade

$$\frac{C_{max}}{C^*_{max}} \leq \frac{3}{2}$$

Sugerimos, portanto, examinar os problemas $R2//C_{max}$, nos quais os tempos de processamento obedeçam a outras relações, com os pontos (p_i,q_i) se situarem sobre outras curvas (parábolas, exponenciais). Ou, ainda, caso os pontos (p_i,q_i) se situarem aproximadamente sobre uma reta (com pequena dispersão), pesquisarmos que efeito traria sobre C_{max}, a consideração de que esses pontos se situassem sobre a reta de melhor ajuste. E, finalmente, no problema geral $R//C_{max}$, poderiam ser examinados os casos em que os tempos de processamento de um mesmo job nas diversas máquinas definissem pontos sobre uma superfície (ou hiper-superfície).

Deve-se ressaltar, no entanto, que, para o caso de duas máquinas, as possibilidades não são muito grandes, pois já existe um algoritmo, chamado Algoritmo G, para $R2//C_{max}$ devido a Ibarra e Kim [12], com qualidade

$$\frac{C_{max}(G)}{C^*_{max}} \leq \frac{1 + \sqrt{5}}{2} \approx 1.618$$

muito próximo, portanto, do resultado aqui obtido, 1.5

Mais promissor, talvez, fosse a análise da qualidade do Algoritmo C, descrito no Capítulo IV, quando aplicado ao problema $R2/q_j = \alpha p_j + \beta/\sum C_j$.

No Capítulo V, desenvolvemos algoritmos polinomiais para problemas de Flow-shop Scheduling, determinando C_{max} e $\sum C_j$ quando os tempos de processamento do job J_i em cada máquina M_j, p_{ij}, forem tais que p_{ij} é constante para todo j. Sugerimos, portanto, as seguintes extensões:

a) Nos problemas de Flow-shop Scheduling com a restrição "no wait", sempre ocorre que, em alguma máquina M_j, $j=1,2,\ldots,n$, o início do processamento de um job $J_{\sigma(i)}$ se dará imediatamente após o término do processamento do job $J_{\sigma(i-1)}$; por simplicidade, diremos que houve uma "colisão" em M_j, nesse caso. Nos problemas estudados, em que p_{ij} é constante para todo j, verificamos que, se $p_{\sigma(i-1)} \leq p_{\sigma(i)}$, a "colisão" ocorrerá em M_1 (veja figura 5.5); e que, se $p_{\sigma(i-1)} > p_{\sigma(i)}$, a "colisão" acontecerá na máquina M_n (veja figura 5.6). Propomos estender os resultados obtidos, para F/no wait,p_i/C_{max} e F/no wait,$p_i/\sum C_j$, para problemas de Flow-shop Scheduling, com a restrição "no wait" e tempos de processamento quaisquer, porém sabendo-se que, para qualquer par (i,j) o processamento do job J_j imediatamente após J_i fará com que a "colisão" ocorra em M_1 ou M_n.

b) Outro problema que poderia ser analisado, seria ainda o Flow-shop Scheduling (com ou sem a restrição "no wait") nos quais, ao inves de assumirmos p_{ij} constante para todo j,

poderiamos considerar que houvesse uma proporcionalidade entre os tempos, isto é, poderíamos assumir $p_{ij} = \alpha_j p_{il}$.

c) Uma última e óbvia sugestão consistiria em ampliar os resultados obtidos para os problemas Open-shop e Job-shop Scheduling.

Acreditamos que, para estas tres últimas sugestões, ainda fossem encontrados algoritmos polinomialmente limitados.

BIBLIOGRAFIA

1. KNUTH, D.E., Fundamental Algorithms, Addison-Wesley Publishing Co., Reading, 1972.

2. KARP, R.M., Reducibility among Combinatorial Problems, in R.E. Miller e J.W. Thatcher eds., Complexity of Computer Computations, 85-103, Plenum Press, N.York, 1972

3. WEIDE, B., A Survey of Analysis Techniques for Discrete Algorithms, in Computing Surveys, vol. 9, nº 4, Dezembro de 1977, 291-313.

4. BAASE, S., Computer Algorithms: Introduction to Design and Analysis, Addison-Wesley Publishing Co., 1978.

5. GAREY, M.R. e JOHNSON, D.S., Computers and Intractability, W.H. Freeman and Co., S. Francisco, 1979.

6. LEVIN, R.I., e KIRKPATRICK, C.A., Planning and Control with CPM/PERT, Mc Graw-Hill, Inc., N. York, 1966.

7. GRAHAM, R.L., LAWLER, E.L., LENSTRA, J.K., e RINNOOY KAN, A.H.G., Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, in Annals of Discrete Mathematics 5, 287-326, 1979

8. LAWLER, E.L., Sequencing to Minimize the Weighted Number of Tardy Jobs, in Rev. Française Automat. Informat. Recherche Operationnelle 10, 27-33, 1976.

9. BRUNO, J., COFFMAN, E.G., e SETHI, R., Scheduling Independent Tasks to Reduce Mean Finishing Time, in Comm. ACM 17, 382-387, 1974

10. LENSTRA, J.K., RINNOOY KAN, A.H.G., e BRUCKER, P., Complexity of Machine Scheduling Problems, in Annals of Discrete Mathematics 4, 281-300, 1977.

11. GONZALEZ, T., IBARRA, O.H., e SAHNI, S., Bounds for LTP Schedules on Uniforme Processors, in SIAM J. Comput. 6, 155-166, 1977.

12. IBARRA, O.H., e KIM, C.E., Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors, in J. Assoc. Comput. Mach. 24, 280-289, 1977.

13. JOHNSON, S.M., Optimal two-and three-stage Production Schedules with setup Times Included, in Naval Res. Logist. Quart. 1, 61-68, 1954.

14. GAREY, M.R., e JOHNSON, D.S., Approximation Algorithms for Combinatorial Problems: an Annotated Bibliography, in J.F. Traub, ed., Algorithms and Complexity: New Directions and Recent Results, 41-52, Academic Press, N. York, 1976.

15. REDDI, S.S., e RAMAMOORTHY, C.V., On the Flow-shop Sequencing Problem with No Wait in Process, Operational Res. Quart. 23, 323-331, 1972.

16. GILMORE, P.C., e GOMORY, R.E., Sequencing a One-state Variable Machine: a solvable case of the Traveling Salesman Problem, Operations Res. 12, 655-679, 1964.

17. DUDEK, R.A., e TEUTON, O.F., Development of M-stage Decision Rule for Scheduling n Jobs through M Machines, Operations Res. 12, 471, 1964.

18. LAGEWEG, B.J., LENSTRA, J.K., RINNOOY KAN, A.H.G., A General Bounding Scheme for the Permutation Flow-shop Problem, Operation Res. 26, nº 1, 53-67, 1978.

19. GAREY, M.R., JOHNSON D.S., e SETHI, R., The Complexity of Flow-shop and Job-shop Scheduling, Math. of Oper. Research, vol. 1, nº 2, 117-129, 1976.

20. DAVIS, E., e JAFFE, J.M., Algorithms for Scheduling tasks on Unrelated Processors, J. ACM, 28, nº 4, 721-736, 1981.

21. GONZALEZ, T., e SAHNI, S., Flowshop and Jobshop Schedules: Complexity and Approximation, Op. Research 26, 36-52, 1978.

22. GUPTA, J.N.D., A Functional Heuristic Algorithm for the Flow-shop Scheduling Problem, Op. Res. Quaterly, 22, nº 1, 39-47.

23. JAFFE, J.M., Bounds on the Scheduling of Typed Task Systems, SIAM J. Computing, vol. 9, nº 3, 541-551, 1980.

24. CHIN, F.Y., e TSAI, L.L., On J-maximal and J-minimal Flow-shop Schedules, J. ACM, 28, nº 3, 462-476, 1981.

25. PAPADIMITRIOU, C.H., e KANELLAKIS, P.C., Flow-shop Scheduling with Limited Temporary Storage, J. ACM., 27, nº 3, 553-549, 1980.