Universidade Federal do Espírito Santo

Centro Universitário Norte do Espírito Santo

Colegiado do Curso de Bacharelado em Matemática

Rafael Marin Permanhane

PROBLEMAS DE ESCALONAMENTO

São Mateus

2014

Rafael Marin Permanhane

PROBLEMAS DE ESCALONAMENTO

Trabalho submetido ao Colegiado do Curso de Bacharelado em Matemática da UFES (Campus São Mateus), como requisito parcial para a obtenção do grau de Bacharel em Matemática.

São Mateus

2014

Rafael Marin Permanhane

PROBLEMAS DE ESCALONAMENTO

Trabalho submetido ao Colegiado do Curso de Bacharelado em Matemática da UFES (Campus São Mateus), como requisito parcial para a obtenção do grau de Bacharel em Matemática.

Aprovada em 27 de Fevereiro de 2014.

Comissão Examinadora

Prof. Leonardo Delarmelina Secchin
Universidade Federal do Espírito Santo
Orientador

Prof. André Renato Sales Amaral
Universidade Federal do Espírito Santo

Prof. Lúcio Souza Fassarella
Universidade Federal do Espírito Santo

Agradecimentos

Agradeço a Deus por abençoar meu caminho. À minha família por estar sempre ao meu lado. À minha namorada por todo o incentivo em estudar cada vez mais. Ao meu orientador por cada ensinamento passado a mim. Aos membros da banca por aceitarem tão prontamente o pedido.

Resumo

Problemas de escalonamento têm vasta aplicação, e são objeto de estudo entre vários pesquisadores. Este trabalho aborda alguns problemas resolvíveis polinomialmente, bem como algoritmos para sua resolução. São apresentadas as principais características dos problemas considerados. Exemplos são fornecidos. O trabalho divide-se em duas classes: problemas com uma única máquina e problemas com várias máquinas em paralelo.

Palavras-chave: Problemas de escalonamento, otimização combinatória.

Lista de Figuras

Sumário

1 Introdução

Problemas de escalonamento têm papel importante em processos da manufatura e indústria. Ocorrem na medida em que necessita-se alocar recursos, como máquinas e trabalhadores, à tarefas, em períodos de tempo pré determinados, almejando otimizar um ou mais objetivos (PINEDO, 2008).

Várias aplicações são conhecidas na literatura. Pinedo (2008) traz em seu livro uma série de exemplos. Apesar da reconhecida importância pela comunidade acadêmica e da intensa pesquisa na área, grande parte dos problemas de escalonamento permanecem desafiadores. Muluk, Akpolat e Xu (2003) estima que, no que tange à cadeias de produção, apenas 15% das aplicações têm sucesso esperado, seja porque os modelos são computacionalmente difíceis de resolver, seja porque as técnicas atuais de modelagem não dão conta de representar adequadamente a realidade do problema.

Neste trabalho apresentamos alguns problemas polinomialmente resolvíveis, divididos em duas classes: uma com única máquina e outra com várias delas, bem como alguns algoritmos de resolução e teoremas provando que tais algoritmos são corretos.

Por fim, acreditamos que este trabalho possa ser utilizado como tópico em disciplina a nível de graduação, bem como parte do estudo para uma iniciação científica. Além disso, problemas de escalonamento são de interesse em mestrados acadêmicos nas linhas de pesquisa operacional e otimização, mas não é comum encontrá-los nos currículos dos cursos de graduação em matemática e engenharias.

2 Problemas de Escalonamento

De forma simplificada, um **problema de escalonamento** consiste em atribuir a dadas máquinas certos trabalhos, a serem processados em determinada ordem e obedecendo à restrições impostas pela situação, de forma a obter a melhor distribuição mediante certo critério de qualidade. Os termos "máquina" e "trabalho" têm sentido amplo em nosso contexto. Para exemplificar, o leitor pode entender "máquina" como um processador de computador e "trabalho" como um fragmento de programa a ser processado por um processador. Outro exemplo é aquele em que "máquina" é um trabalhador, e "trabalho" uma tarefa que determinado trabalhador pode realizar. Nesse caso pode-se procurar, por exemplo, estabelecer a distribuição de tarefas que minimiza os atrasos na realização do serviço.

Na próxima seção apresentamos uma maneira conveniente de representar uma solução do problema (escalonamento ou "distribuição dos trabalhos"). Nas seções seguintes, exibimos características/restrições comuns à problemas de escalonamento comumente considerados na literatura, bem como os critérios de qualidade mais usados. Para uma referência mais ampla, consulte (BRUCKER, 2006).

2.1 Preliminares

Suponha que m **máquinas** M_j $(j = 1, \ldots, m)$ tenham que processar n **trabalhos** J_i $(i = 1, \ldots, n)$. Um **escalonamento** é, para cada trabalho, uma alocação de um ou mais intervalos de tempo em uma ou mais máquinas. Os escalonamentos serão representados pelos **gráficos de Gantt**, que podem ser orientados por trabalhos ou por máquinas, como mostra a Figura 1.

É comum inserirmos a linha do tempo logo abaixo do gráfico de Gantt, como na Figura 7 na página 16.

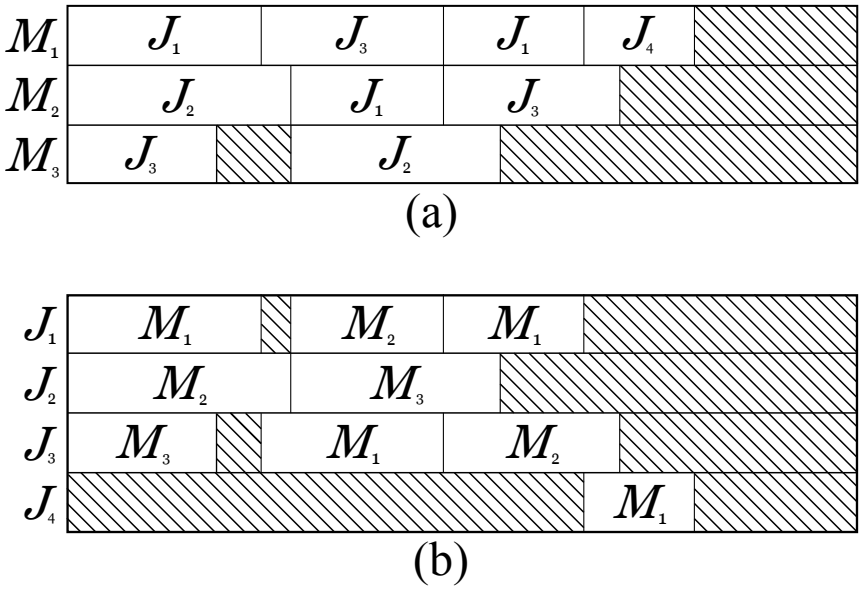


Figura 1: Gráficos de Gantt orientados (a) por máquinas e (b) por trabalhos.

2.2 Dados dos trabalhos

O trabalho J_i é composto por n_i **operações** $O_{i1}, \ldots, O_{in_i}$. Associado a cada operação O_{ij}, há o requerido **tempo de processamento** p_{ij}. Se o trabalho J_i consiste de apenas uma operação ($n_i = 1$), então identificamos J_i com O_{i1} e denotamos o tempo de processamento por p_i. Por vezes, especificamos um **tempo de liberação** r_i, que diz quando a primeira operação do trabalho J_i pode ser iniciada. Cada operação O_{ij} está associada a um **conjunto de máquinas** $\mu_{ij} \subset \{M_1, \ldots, M_m\}$, que são as máquinas capazes de processar O_{ij}. Normalmente μ_{ij} é composto por um único elemento ou é igual ao conjunto de todas as máquinas. No primeiro caso dizemos que as máquinas são **dedicadas**, e no segundo, que estão em **paralelo**. Estabelecemos uma **função custo** $f_i(t)$ que mede o custo de completar J_i no tempo t. Essas funções comporão a "medida de qualidade", ou **custo** do escalonamento. Associamos ainda a cada trabalho J_i uma **data final**, ou **prazo**, d_i para o término de seu processamento e um **peso** $w_i \geq 0$, ambos usados para definir $f_i(t)$.

Geralmente, p_i, p_{ij}, r_i, d_i e w_i são números inteiros. Um escalonamento é **viável** se: (i) dois intervalos de tempo não se sobrepõem numa mesma máquina; (ii) dois intervalos de tempo alocados para uma mesma operação não se sobrepõem e se (iii) atende às características do problema. Um escalonamento é **ótimo** se minimiza a função de custo. Funções de custo serão discutidas adiante, na seção "Critério de Otimalidade" (seção 2.5).

Cada classe de problemas será especificada por três campos de classificação α, β e γ,

dispostos da forma

$$\alpha|\beta|\gamma,$$

onde α denomina o ambiente das máquinas, β as características dos trabalhos e γ o critério de otimalidade. Essa notação é usada por Brucker (BRUCKER, 2006) e é um padrão seguido na literatura.

Nas três seções seguintes nos dedicamos explicar a notação utilizada para os termos α, β e γ. Por simplicidade, daqui para frente poderemos denotar o trabalho J_i simplesmente por i.

2.3 Características dos trabalhos

β consiste de no máximo 6 elementos: $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$ e β_6.

β_1 indica se **preempção** é permitida. A preempção de um trabalho é quando seu processamento pode ser interrompido e retomado posteriormente, inclusive em máquinas diferentes. Se preempção é permitida, definimos β_1=**pmtn**[1].

β_2 indica se existe **relação de precedência** entre os trabalhos. Essas relações devem ser representadas por um grafo acíclico orientado $G = (V, A)$, onde $V = \{1, \ldots, n\}$ corresponde ao conjunto dos trabalhos e $(i, k) \in A$ se o trabalho i deve ser completado para que se inicie o trabalho k, sendo comum escrevermos $i \to k$ (veja a Figura 5, página 15, para um exemplo). Nesse caso, escrevemos β_2=**prec**. Em alguns problemas, há restrições nas relações de precedência, que indicamos no termo β_2. Por exemplo, o grafo G de precedência pode ser uma *intree*, ou seja, uma árvore tal que o grau de saída em cada vértice é no máximo 1 (β_2=**intree**); uma *outtree*, ou seja, uma árvore tal que o grau de entrada em cada vértice é no máximo 1 (β_2=**outtree**); ou um grafo direcionado em série-paralelo (β_2=**sp-graph**), que é construído utilizando-se as seguintes regras:

- **Grafo base:** Qualquer grafo de um único vértice está em série-paralelo.
- **Composição paralela:** Sejam $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$ grafos em série-paralelo. Então o grafo $G = (V_1 \cup V_2, A_1 \cup A_2)$ está em série-paralelo.
- **Composição em série:** O grafo $G = (V_1 \cup V_2, A_1 \cup A_2 \cup (T_1 \times S_2))$ formado por dois grafos em série-paralelo $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$, adicionando todos os arcos (t, s) onde t pertence ao conjunto T_1 dos terminais de G_1 (isto é, o conjunto

[1] A abreviação vem do termo em inglês *preemption*.

dos vértices de G_1 sem sucessores) e s pertence ao conjunto S_2 das fontes de G_2 (isto é, o conjunto dos vértices de G_2 sem antecessores) está em série-paralelo.

A seguir algumas figuras para entender o que são grafos e suas variantes:

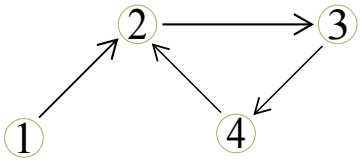


Figura 2: Exemplo de grafo cíclico.

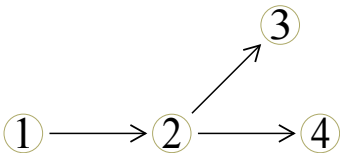


Figura 3: Exemplo de *intree*.

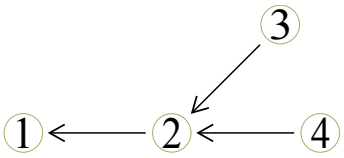


Figura 4: Exemplo de *outtree*.

Se β_3=$\boldsymbol{r_i}$, então a cada trabalho estará associado um tempo de liberação. Caso $r_i = 0$ para todo trabalho, então β_3 não aparece em β.

Se β_4 =“$\boldsymbol{p_i = 1}$” ou β_4 =“$\boldsymbol{p_{ij} = 1}$” então cada trabalho tem um tempo de processamento unitário. Similarmente, pode-se ter “p_i=p” ou “p_{ij}=p”.

Se β_5=$\boldsymbol{d_i}$ então uma data final para que cada trabalho seja concluído é especificada, ou seja, existe um prazo a ser obedecido.

Em algumas aplicações, existem trabalhos que devem ser processados todos juntos em uma máquina. Esses grupos de trabalhos são denominados **lotes**. O **problema de loteamento** consiste em agrupar os trabalhos em lotes e escalonar tais lotes. Trata-se portanto de um problema de escalonamento onde, ao invés de escalonar os trabalhos individualmente, escalona-se lotes de trabalhos. Dois tipos são considerados: ***p*-batch** e ***s*-batch**. Em p-batch (s-batch), o comprimento de um lote é igual ao máximo (à soma) dos tempos de processamento de todos os trabalhos do lote. Em problemas de loteamento,

definimos o parâmetro s que indica a distância entre dois lotes (veja Figura 7, página 16, para um exemplo). Indicamos a existência de lotes por β_6=***p*-batch** ou β_6=***s*-batch**.

2.4 Ambiente das máquinas

O ambiente das máquinas é caracterizado por $\alpha = \alpha_1\alpha_2$. Possíveis valores para α_1 são $\circ, P, Q, R, G, X, O, J, F$. Se $\alpha_1 \in \{\circ, P, Q, R\}$, onde $\circ$ denota o símbolo vazio (α=α_2 se α_1=$\circ$), então cada J_i consiste de uma única operação.

Se α_1=$\circ$, cada trabalho deve ser processado em uma máquina específica (**máquinas dedicadas**).

Se $\alpha_1 \in \{P, Q, R\}$ então as máquinas estão em paralelo, e todas elas podem processar todos os trabalhos. Se α_1=***P***, todas as máquinas têm a mesma velocidade de processamento para todos os trabalhos (**máquinas idênticas**) e, sendo p_{ij} o tempo de processamento do trabalho i na máquina M_j, temos p_{ij}=p_i para todas as máquinas M_j's. α_1=***Q*** indica que cada máquina tem sua velocidade de processamento para todos os trabalhos (**máquinas uniformes**) e calculamos p_{ij}=p_i/s_j, onde s_j é a velocidade da máquina M_j. Se α_1=***R***, cada máquina tem sua velocidade de processamento, e esta ainda depende do trabalho que está sendo processado (**máquinas sem relação**). Aqui, p_{ij}=p_i/s_{ij}, onde s_{ij} é a velocidade de processamento do trabalho i na máquina M_j.

Se $\alpha_1 \in \{G, X, O, J, F\}$, temos um modelo multi operações, onde existe, associado a cada trabalho J_i, um conjunto de operações $O_{i1}, \ldots, O_{in_i}$ e onde todas as máquinas são dedicadas, ou seja, todos os μ_{ij} são conjuntos de um elemento. O modelo geral, onde existem relações de precedência entre operações arbitrárias, é chamado **general shop**, e indicamos por $\alpha_1 = \boldsymbol{G}$. Os outros modelos são casos especiais do *general shop*. Em um **job shop**, indicado por $\alpha_1 = \boldsymbol{J}$, as relações de precedência são, para cada trabalho J_i, da forma

$$O_{i1} \to O_{i2} \to O_{i3} \to \cdots \to O_{in_i}.$$

Geralmente assumimos que $\mu_{ij} \neq \mu_{i,j+1}$, para $j = 1, \ldots, n_i - 1$. Um modelo *job shop* onde é possível $\mu_{ij} = \mu_{i,j+1}$ é chamado **job shop with machine repetition**. Indicado por $\alpha_1 = \boldsymbol{F}$, o **flow shop** é um caso especial do *job shop*, onde se tem $n_i = m$, para $i = 1, \ldots, n$ e ainda $\mu_{ij} = \{M_j\}$ para $i = 1, \ldots, n$ e $j = 1, \ldots, m$. Indicado por $\alpha_1 = \boldsymbol{O}$, o **open shop** é definido de maneira similar ao *flow shop*, se diferenciando pelo fato de não haver relações de precedência entre operações. Um **mixed shop** é uma combinação de um *job shop* com um *open shop* e é indicado por $\alpha_1 = \mathbf{X}$. Cabe ressaltar que os problemas

multi operações não são tratados neste trabalho. Para uma referência completa, consulte (BRUCKER, 2006).

α_2 denota o número de máquinas do problema (α_2=**1**,**2**,**3**,...). Se o número de máquinas é arbitrário, definimos α_2=o.

2.5 Critério de Otimalidade

Denotemos o **tempo de completação** do trabalho i por C_i, e seu **custo associado** por $f_i(C_i)$. A fim de medir a qualidade de um escalonamento viável C, consideramos dois tipos genéricos de funções custo-total:

$$f_{\max}(C) := \max\{f_i(C_i); i = 1, \ldots, n\}$$

e

$$\sum f_i(C) := \sum_{i=1}^{n} f_i(C_i)$$

chamadas **função gargalo** e **função soma**, respectivamente. Nosso objetivo sempre será minimizar a função custo-total. Passemos agora a discutir escolhas específicas para o custo do trabalho i.

A escolha mais natural para o custo associado ao trabalho i é seu próprio tempo de completação, ou seja, $f_i(C_i) = C_i$. Outras funções $f_i(C_i)$ comuns, que dependem dos prazos de conclusão d_i, são listadas a seguir.

$L_i :=$	$C_i - d_i$	**lateness**
$T_i :=$	$\max\{0, C_i - d_i\}$	**tardiness**
$E_i :=$	$\max\{0, d_i - C_i\}$	**earliness**
$U_i :=$	$\begin{cases} 0 & \text{se } C_i \leq d_i \\ 1 & \text{caso contrário} \end{cases}$	**penalização unitária**

Observe que *lateness* e *tardiness* são parecidas. A diferença é que a primeira penaliza o custo total quando o trabalho j é finalizado após o seu prazo de conclusão d_j e favorece quando é finalizado antes, enquanto o segundo só há a penalização (lembre-se que queremos minimizar o custo total). *Earliness* penaliza o custo total se algum trabalho for finalizado antes do seu prazo, e não contribui caso seja finalizado tardiamente. Observe ainda que, nos três primeiros casos, a penalização é feita proporcionalmente à distância da data de finalização para o prazo de conclusão, enquanto que na penalização unitária, como o nome sugere, há uma penalização caso o trabalho seja finalizado após seu prazo

em uma unidade, não importando o quanto de atraso ocorra. Em uma analogia prática, o leitor pode pensar na diferença entre multa diária no atraso do pagamento de uma conta e multa aplicada de uma só vez.

Para cada uma das escolhas $G_i = L_i, T_i, E_i, U_i$, lembrando que essas são apensa exemplos e que existem outras, temos quatro possíveis objetivos, representados pelo parâmetro de classificação γ:

$$\gamma = \max G_i, \quad \gamma = \max w_i G_i, \quad \gamma = \sum G_i \quad \text{e} \quad \gamma = \sum w_i G_i.$$

Em particular, três funções custo-total relativas à escolha $f_i(C_i) = C_i$ são de especial interesse:

- $C_{\max} := \max\{C_i; i = 1, \ldots, n\}$ (**makespan**);
- $\sum_{i=1}^{n} C_i$ (**fluxo total de tempo**);
- $\sum_{i=1}^{n} w_i C_i$ (**fluxo total de tempo ponderado**).

2.6 Exemplos

Exemplo 2.6.1. $\boldsymbol{P \mid \text{prec}; p_i = 1 \mid C_{\max}}$ é o problema de escalonar trabalhos com tempo de processamento unitário e certas relações de precedência, dadas por um grafo orientado, em m máquinas idênticas, onde o *makespan* é minimizado.

Como ilustração, considere o problema em que queremos escalonar, em 2 máquinas, 7 trabalhos com relações de precedência dadas pelo seguinte grafo orientado:

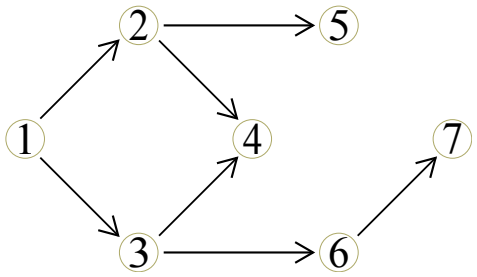


Figura 5: Relações de precedência para o Exemplo 2.6.1.

Mesmo não havendo tempos de liberação especificados, ou seja, todos os trabalhos podendo ser iniciados no tempo $t = 0$, devemos ter cuidado com as relações de precedência, pois só é permitido executar um trabalho após seus antecessores serem completados. Por

exemplo, só podemos iniciar o trabalho 2 após 1 ser completado, já que 1 precede 2 ($1 \to 2$). Da mesma forma, 4 só pode ser iniciado após 2 ser completado pois $2 \to 4$. Assim, 4 depende de 1 e 2. A figura a seguir mostra dois escalonamentos viáveis, um deles ótimo, ou seja, que minimiza o *makespan*.

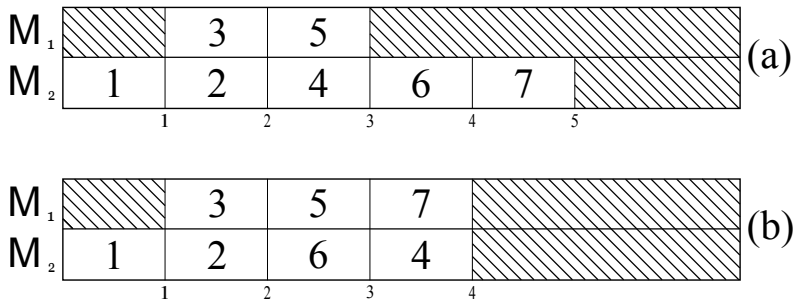


Figura 6: Escalonamentos (a) viável e (b) ótimo para $\boldsymbol{P \mid \text{prec}; p_i = 1 \mid C_{max}}$.

□

Exemplo 2.6.2. $\boldsymbol{1 \mid s\text{-batch} \mid \sum w_i C_i}$ é o problema de dividir um conjunto de trabalhos em lotes e escalonar esses lotes em uma única máquina de modo a minimizar o fluxo total de tempo ponderado. O tempo de processamento de cada lote é igual à soma dos tempos de processamento de cada trabalho nesse lote. São dados do problema, além de $s = 1$,

i	1	2	3	4	5	6
p_i	3	2	2	3	1	1
w_i	1	2	1	1	4	4

Dividindo os trabalhos em 3 lotes, digamos $\{2\}$, $\{1,3,5\}$ e $\{4,6\}$, um escalonamento viável é

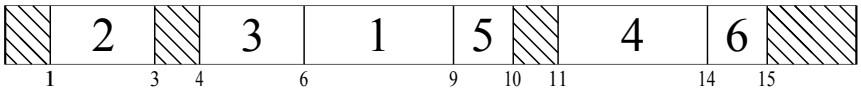


Figura 7: Escalonamento viável para $\boldsymbol{1 \mid s\text{-batch} \mid \sum w_i C_i}$.

De acordo com a tabela de dados, o custo total para esse escalonamento é

$$\sum w_i C_i = \underbrace{2 \cdot 3}_{\text{lote } \{2\}} + \underbrace{(1+1+4) \cdot 10}_{\text{lote } \{1,3,5\}} + \underbrace{(1+4) \cdot 15}_{\text{lote } \{4,6\}} = 141.$$

Observe que modificar a ordem dos trabalhos dentro de cada lote não altera o custo total. No entanto, se mudarmos a ordem dos lotes, digamos para $\{1,3,5\}$, $\{2\}$ e $\{4,6\}$, o

custo total diminui para

$$\sum w_i C_i = \underbrace{(1+1+4)\cdot 7}_{\text{lote } \{1,3,5\}} + \underbrace{2\cdot 10}_{\text{lote } \{2\}} + \underbrace{(1+4)\cdot 15}_{\text{lote } \{4,6\}} = 137,$$

o que mostra que o escalonamento da Figura 7 não é ótimo para 3 lotes. □

3 Problemas de escalonamento em uma máquina

Neste capítulo serão apresentados alguns algoritmos para problemas de escalonamento em uma única máquina, em casos onde a complexidade de resolução é polinomial. Para uma visão mais completa do assunto, consulte (BRUCKER, 2006; PINEDO, 2008). Alertamos o leitor que a nomenclatura e termos introduzidos no Capítulo 2 referentes à classificação dos problemas e parâmetros serão largamente utilizados.

Um fato geral, e útil, sobre problemas em única máquina é o seguinte: é intuitivo pensar que para problemas com uma única máquina, se tivermos $r_j = 0$ para todo trabalho j, e se a função custo total for não decrescente no tempo de completação dos trabalhos, então podemos descartar escalonamentos com preempção ou com tempos ociosos. A Figura 8(a) exibe um escalonamento com tempo ocioso. Note que podemos deslocar os trabalhos para a esquerda, eliminando o tempo ocioso (Figura 8(b)) pois todos os trabalhos podem ser iniciados já no tempo $t = 0$, e a função custo total, sendo não decrescente nos tempos de completação, não aumenta. Ou seja, o escalonamento sem tempo ocioso não é pior que o anterior.

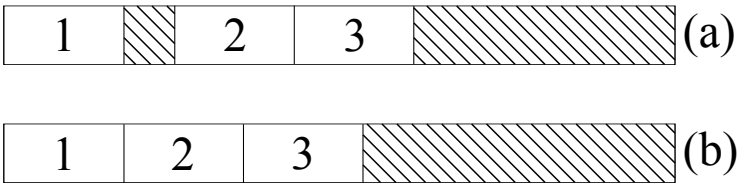


Figura 8: Escalonamentos (a) com e (b) sem tempo ocioso.

Na Figura 9 rearranjamos os trabalhos no escalonamento com preempção de forma que nenhuma interrupção é mais necessária. Isso não piora a função custo total visto que o tempo de completação dos trabalhos 1 e 3 não muda, e o de 2 diminui.

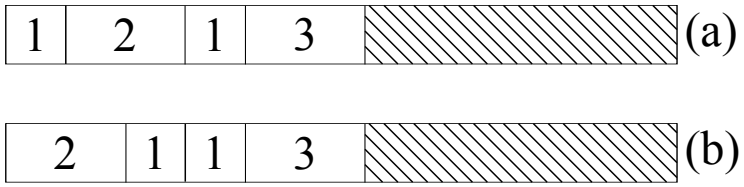


Figura 9: Escalonamentos (a) com e (b) sem preempção.

Deslocar trabalhos pode gerar inviabilidade se algum $r_j > 0$. Se por exemplo $r_2 > 0$, o escalonamento da Figura 9(b) não seria viável. No entanto, contornamos tal dificuldade trocando 1 e 2 de posição, gerando o escalonamento viável, e sem preempção, da Figura 10.

Figura 10: Escalonamento sem preempção respeitando $r_2 \neq 0$.

Continuando com esse procedimento, obtemos uma solução ótima para o problema preemptivo onde não é necessário ter, de fato, algum trabalho com preempção.

Nas seções seguintes, apresentaremos algoritmos (polinomiais) para certos problemas. Provas de corretude serão fornecidas e em alguns casos, exemplos serão discutidos.

3.1 Critério Minimax

Os problemas dessa seção são aqueles onde a função custo total é da forma

$$f_{\max} = \max_{j=1}^{n} f_j(C_j)$$

e as funções f_j são não decrescentes no tempo de completação. O termo "minimax" vem do fato de estarmos minimizando uma função definida pelo máximo das f_j's.

3.1.1 Algoritmo de Lawler para 1 | prec | $f_{\max}$

Para resolver tal problema, é suficiente construir uma permutação ótima π : $\pi(1), \pi(2), \ldots, \pi(n)$, onde $\pi(i)$ denota o trabalho na posição i, já que podemos desconsiderar preempção e ociosidade, de acordo com observação do início do Capítulo. Lawler (LAWER, 1973) desenvolveu um algoritmo simples que constrói essa sequência em ordem reversa, que apresentamos a seguir.

Seja $N = \{1, \ldots, n\}$ o conjunto de todos os trabalhos e $S \subset N$ o conjunto dos trabalhos não escalonados. Além disso definamos $p(S) = \sum_{j \in S} p_j$, o tempo de processamento dos trabalhos não escalonados. A regra de escalonamento é reformulada como segue: escalone um trabalho $j \in S$ que não tenha sucessor em S e tenha valor mínimo para $f_j(p(S))$, até que S fique vazio.

Para dar uma descrição mais precisa do algoritmo, representamos as restrições de precedência pela matriz (binária) de adjacência $A = (a_{ij})$, onde $a_{ij} = 1$ se, e somente se, j é sucessor direto de i. O número de sucessores diretos de i será denotado por $n(i)$. Cabe ressaltar que na linha 7 do algoritmo, o ∞ é para denotar que $n(j)$ recebe valor suficientemente grande, visto que pode variar de problema para problema.

Algoritmo 1 : 1 | prec | $f_{\max}$

1: **Para** $i = 1$ **até** n **faça** $n(i) := \sum_{j=1}^{n} a_{ij}$;
2: $S := \{1, \ldots, n\}$; $p := \sum_{j=1}^{n} p_j$;
3: **Para** $k = n$ **até** 1 **faça**
4: **Início**
5: Encontre um trabalho $j \in S$ com $n(j) = 0$ e valor mínimo para $f_j(p)$;
6: $S := S - \{j\}$;
7: $n(j) := \infty$;
8: $\pi(k) := j$;
9: $p := p - p_j$;
10: **Para** $i = 1$ **até** n **faça**
11: **Se** $a_{ij} = 1$ **então** $n(i) := n(i) - 1$;
12: **Fim**

Teorema 3.1.1. *O algoritmo* **1 | prec | $f_{\max}$** *constrói uma sequência ótima.*

Demonstração. Enumere os trabalhos de forma que $1, 2, \ldots, n$ é a sequência construída pelo algoritmo. Suponhamos que esta sequência não seja ótima. Tomemos assim $\sigma : \sigma(1), \ldots, \sigma(n)$ uma sequência ótima tal que $\sigma(i) = i$ para $i = n, n-1, \ldots, r$ e $\sigma(r-1) \neq r-1$ onde r é mínimo com esta propriedade. Temos a seguinte situação:

σ:	$\cdots$	$r-1$	k	$\cdots$	j	r	$\cdots$	$n-1$	n

Pela sequência gerada pelo algoritmo, temos que $r-1$ não tem sucessor no conjunto $\{1, \ldots, r-1\}$ (a sequência construída está ordenada) e, pela sequência ótima σ, temos

que j não tem sucessor nesse mesmo conjunto. O algoritmo escolhe um trabalho que tiver menor valor $f_i(p)$ e, como $j < r-1$, temos

$$f_{r-1}(p) \leq f_j(p), \quad \text{onde } p = \sum_{i=1}^{r-1} p_i.$$

Na sequência ótima σ, trocamos de posição $r-1$ com o bloco entre $r-1$ e r. O tempo de completação de $r-1$ será p e o dos trabalhos i entre $r-1$ e r passarão a ser $C_i - p_{r-1}$. Temos ainda

$$f_i(C_i - p_{r-1}) \leq f_i(C_i),$$

já que as f_i's são funções não decrescente. Concluímos então que há um escalonamento também ótimo, mas que contraria a minimalidade de r, pois $r-1$ é escalonado imediatamente antes de r. Logo a sequência construída pelo algoritmo é ótima. □

3.1.2 1 | prec; pmtn; r_j | $f_{\max}$

O primeiro passo para resolver esse problema é modificar os tempos de liberação r_j. Se $i \to j$ (j é sucessor de i) e $r_i + p_i > r_j$, então o trabalho j não pode iniciar antes de $r'_j = r_i + p_i$. Nesse caso, trocamos r_j por r'_j. Os tempos de inicialização são modificados de maneira sistemática, seguindo o seguinte algoritmo. Assumiremos aqui que os trabalhos são enumerados *topologicamente*, ou seja, para todos os trabalhos i, j com $i \to j$, temos $i < j$.

Algoritmo 2 : Modificar r_j

1: **Para** $i = 1$ **até** $n-1$ **faça**
2: **Para** $j = i+1$ **até** n **faça**
3: **Se** $i \to j$ **então** $r_j := \max\{r_j, r_i + p_i\}$;

Note que se os tempos de processamento são todos positivos, então

$$r'_j > r'_i \quad \text{se} \quad i \to j.$$

Assim, se escalonarmos os trabalhos em ordem não decrescente dos tempos de liberação r'_j, e respeitando os respectivos tempos de liberação, sempre teremos um escalonamento viável. Tal escalonamento conterá vários **blocos**, conjuntos maximais de trabalhos que processados em sequência, sem tempos ociosos. O algoritmo a seguir constrói esses blocos. Assumimos que todos os tempos de liberação estão modificados pelo Algoritmo 2.

Algoritmo 3 : Blocos(1,2,. . . ,n)

```
1:  i := 1; j := 1;
2:  Enquanto i ≤ n faça
3:  Início
4:      t := r_i; B_j := ∅;
5:      Enquanto r_i ≤ t e i ≤ n faça
6:      Início
7:          B_j := B_j ∪ {i};
8:          t := t + p_i;
9:          C_i := t;
10:         i := i + 1;
11:     Fim
12:     j := j + 1;
13: Fim
```

Os valores C_i calculados pelo algoritmo definem os tempos de completação dos trabalhos em um escalonamento sem preempção, pois C_i é definido somando p_i ao tempo atual (linhas 8 e 9). Para cada bloco B_j, definimos

$$s_j = \min_{i \in B_j} r_i$$
$$p(B_j) = \sum_{i \in B_j} p_i$$
$$t_j = t(B_j) = s_j + p(B_j)$$

onde s_j e t_j são os tempos de início e término de B_j, respectivamente.

A discussão até aqui é valida para escalonamentos sem preempção e trabalhos com tempos de processamento arbitrários. Vamos agora ao problema que é de nosso interesse, $\mathbf{1 \mid prec; pmtn; } \boldsymbol{r_j} \mathbf{ \mid } \boldsymbol{f_{\max}}$.

Lema 3.1.1. *Para o problema* $\mathbf{1 \mid prec; pmtn; } \boldsymbol{r_j} \mathbf{ \mid } \boldsymbol{f_{\max}}$*, existe um escalonamento ótimo tal que os intervalos* $[s_j, t_j]$*,* $j = 1, 2, \ldots, k$*, construídos pelo algoritmo "Blocos" são completamente ocupados por trabalhos.*

Demonstração. Considere um escalonamento ótimo π com a propriedade de que $[s, t]$ é um tempo ocioso contido em um intervalo $[s_j, t_j]$. Mais ainda, assumamos que este seja o primeiro intervalo com tal propriedade. Afirmamos que no escalonamento π_B construído pelo algoritmo "Blocos" existe um trabalho i com $r_i \leq s$ que termina depois de s.

De fato, se não existir, seja T o conjunto dos trabalhos que têm partes processadas em tempos posteriores a s, relativos à π_B. Temos assim

$$r = \min\{r_d; d \in T\} > s.$$

Logo nenhum trabalho em T pode iniciar em $[s, r)$, ou seja, $[s, r] \subset [s_j, t_j]$ é um intervalo ocioso em π_B, um absurdo.

Assim, no escalonamento π movemos o trabalho i, ou parte dele, para $[s, t]$. Como $[s, t]$ é o primeiro intervalo ocioso, o movimento é sempre para a esquerda, e a função custo total não piora, visto que C_i não aumenta e f_i é não decrescente. Ou seja, o escalonamento obtido permanece ótimo. Se o intervalo $[s_j, t_j]$ ainda contiver espaços vazios, repetimos o processo. Em finitos passos obtemos um escalonamento com a propriedade requerida. □

Seja B um bloco. Denotando por $f^*_{max}(B) = \max_{j \in B} f_j(C_j)$ o valor ótimo para os trabalhos do bloco B, temos $f^*_{max}(B - \{j\}) \leq f^*_{max}(B)$ para todo j em B, e então

$$f^*_{max}(B) \geq \max_{j \in B} \{f^*_{max}(B - \{j\})\}. \tag{3.1}$$

Mais ainda,

$$f_l(t(B)) = \min\{f_j(t(B)); j \in B \text{ sem sucessores em } B\} \leq f^*_{max}(B) \tag{3.2}$$

pois em um escalonamento ótimo, um trabalho $k \in B$ sem sucessores em B é finalizado no tempo $t(B)$.

Um escalonamento é então construído como segue. Resolvemos o problema para o conjunto de trabalhos $B - \{l\}$. A solução ótima para este problema também tem uma estrutura de blocos correspondente. Similarmente à prova do Lema 3.1.1, pode-se mostrar que o trabalho l pode ser escalonado em algum intervalo de tempo ocioso dessa estrutura de blocos gerando assim um escalonamento cujo valor objetivo é no máximo

$$\max\{f^*_{\max}(B - \{l\}), f_l(t(B))\} \leq f^*_{\max}(B),$$

que é o valor ótimo para B (ver (3.1) e (3.2)).

Agora estamos prontos para formular um procedimento para resolver o problema proposto, $\mathbf{1 \mid prec; pmtn; } \boldsymbol{r_j} \mid \boldsymbol{f_{\max}}$.

Algoritmo 4 : **1 | prec; pmtn; r_j | $f_{\max}$**

1: $S := \{1, \ldots, n\}$;
2: $f^*_{\max} := Decompor(S)$;

"Decompor" é um procedimento recursivo que retorna o valor ótimo $f_{\max}$ para o problema com o conjunto de trabalhos $S \neq \emptyset$.

Algoritmo 5 : Decompor(S)

1: **Se** $S = \emptyset$ **então retorne** $-\infty$;
2: **Se** $S = \{i\}$ **então retorne** $f_i(r_i + p_i)$;
3: **Caso contrário**
4: **Início**
5: Chame "Blocos(S)";
6: $f := -\infty$;
7: **Para todo** bloco B **faça**
8: **Início**
9: Encontre l com $f_l(t(B)) = \min\{f_j(t(B)); j \in B$ e não tem sucessor em $B\}$;
10: $h := Decompor(B - \{l\})$;
11: $f := \max\{f, h, f_l(t(B))\}$;
12: **Fim**
13: **Retorne** f;
14: **Fim**

Para entendermos melhor o algoritmo "decompor", vejamos um exemplo.

Exemplo 3.1.2. Considere um problema com $n = 5$, funções-custo dadas por $f_i(C) = C - d_i$, e relações de precedência dadas pela figura

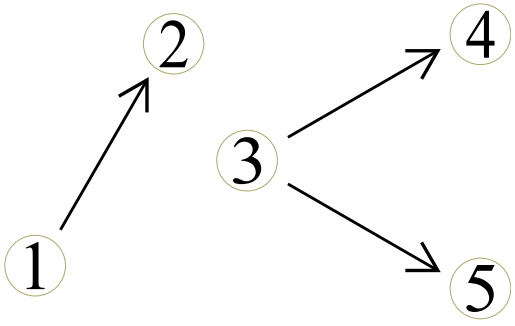


Figura 11: Relações de precedência do Exemplo 3.1.2.

Considere ainda os seguintes dados

i	1	2	3	4	5
p_i	2	3	4	1	2
r_i	0	2	6	6	6
d_i	4	5	3	7	6
r'_i	0	2	6	10	10

onde a última linha contém os tempos de liberação modificados pelo Algoritmo 2. A aplicação $Blocos\{1,\ldots,5\}$ gera

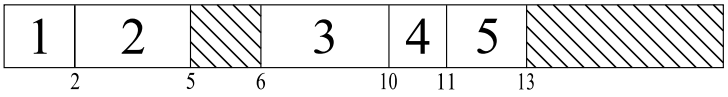


Figura 12: Blocos $\{1,2\}$ e $\{3,4,5\}$.

Denotaremos por B_1 o bloco {1,2}, por B_2 o bloco {3,4,5} e por B_3 o bloco {3,5}, que aparece na decomposição do bloco B_2 no Algoritmo 4, cuja aplicação resulta nos passos a seguir.

- **Para B_1**
  - ✓ $f = -\infty$
  - ✓ $l = 2$
  - ✓ $f_2(t(B_1)) = t(B_1) - d_2 = 5 - 5 = 0$
  - ✓ Decompor$(B_1 - \{2\}) \Rightarrow h = f_1(r_1 + p_1) = r_1 + p_1 - d_1 = 0 + 2 - 4 = -2$
  - ✓ $f = \max\{-\infty, -2, 0\} = 0$
- **Para B_2**
  - ✓ $\left.\begin{array}{l} f_4(t(B_2)) = 13 - 7 = 6 \\ f_5(t(B_2)) = 13 - 6 = 7 \end{array}\right\} \Rightarrow l = 4$
  - ✓ Decompor$(\{3,5\})$
    - ✓ $\hat{f} = -\infty$
    - ✓ $l = 5$
    - ✓ $f_5(t(B_3)) = 12 - 6 = 6 \Rightarrow l = 5$
    - ✓ Decompor$(B_3 - \{5\}) \Rightarrow h = f_3(r_3 + p_3) = r_3 + p_3 - d_3 = 6 + 4 - 3 = 7$

✓ $\hat{f} = \max\{-\infty, 7, 6\} = 7$

✓ $f := \max\{f, \hat{f}, f_4(t(B_2))\} = \max\{0, 7, 6\} = \mathbf{7}$.

Para o escalonamento da Figura 12, temos:

$$\begin{aligned} f_{\max} &= \max_i\{f_i(C_i)\} \\ &= \max\{f_1(C_1), f_2(C_2), f_3(C_3), f_4(C_4), f_5(C_5)\} \\ &= \max\{2-4, 5-5, 10-3, 11-7, 13-6\} \\ &= \max\{-2, 0, 7, 4, 7\} = 7 \end{aligned}$$

Como o valor objetivo é o mesmo que o valor retornado pelo algoritmo "Decompor", então temos, na Figura 12, um escalonamento ótimo. □

3.2 Máximo Lateness

Existem muitos problemas os quais ainda não são conhecidos nenhum algoritmo de resolução que os resolva em tempo polinomial. Tais problemas são reunidos em um grupo chamado $\mathcal{NP}$-difícil. Um exemplo desses problemas é o $\mathbf{1} \mid \boldsymbol{r_j} \mid \boldsymbol{L_{\max}}$ (BRUCKER, 2006), mas apesar de pertencer a esse grupo, existem casos especiais dele que são resolvíveis polinomialmente. Dentre eles, podemos citar os casos em que

(a) todos os trabalhos têm mesmo tempo de liberação, ou seja, $r_j = r$ para todo $j = 1, \ldots, n$. Neste caso, conseguimos um escalonamento ótimo aplicando a regra de escalonamento de Jackson:

> Escalone trabalhos em ordem não decrescente das datas finais (prazos).

(b) todos os trabalhos têm mesmo prazo de completação, ou seja, $d_j = d$ para todo $j = 1, \ldots, n$. Neste caso, um escalonamento ótimo é obtido aplicando a seguinte regra:

> Escalone trabalhos em ordem não decrescente dos tempos de liberação.

Essas regras fornecem diretamente algoritmos polinomiais que resolvem o problema nos casos especificados, e por isso não serão descritos. Agora, vamos provar que as regras apresentadas realmente funcionam.

(a): Suponha que em um escalonamento ótimo, i venha antes de j com $d_i > d_j$. Podemos supor sem perda de generalidade que i e j são consecutivos. De fato, se não fossem, dado

k entre i e j, temos $d_k \geq d_i$ ou $d_k < d_i$. No primeiro caso, k e i violam a regra. No segundo caso, k e j violam a regra. Repetindo esse argumento, conseguimos trabalhos consecutivos que violam a regra.

Trocamos i e j de posição obtendo novos L'_i e L'_j. Note que os outros L_k's permanecem inalterados.

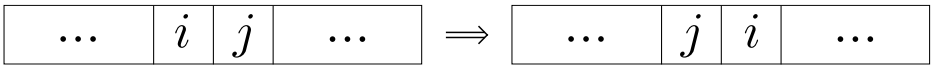


Figura 13: Regras para minimização do Máximo Lateness: troca de i com j.

Temos

$$\begin{aligned} L'_j &= C'_j - d_j \leq C_j - d_j = L_j \leq L_{max} \text{ e} \\ L'_i &= C'_i - d_i = C_i + p_j - d_i \\ &= C_i + (C_j - C_i) - d_i = C_j - d_i \\ &\leq C_j - d_j = L_j \leq L_{max} \end{aligned}$$

e logo o novo escalonamento é ótimo.

(b): Suponha que em um escalonamento ótimo, i venha antes de j com $r_i > r_j$. Sem perda de generalidade supomos que i e j são consecutivos. Se não fosse, dado k entre i e j, temos $r_k \geq r_i$ ou $r_k < r_i$. No primeiro caso, k e i violam a regra. No segundo caso, k e j. Repetindo esse argumento, conseguimos trabalhos consecutivos que violam a regra.

Note que aqui, os tempos de liberação estão modificados e daí $i \not\to j$ visto que $r_i > r_j$. Logo podemos trocar i e j de posição, obtendo novos L'_i e L'_j e permanecendo os outros L_k's inalterados. Temos

$$\begin{aligned} L'_j &= C'_j - d \leq C_j - d = L_j \leq L_{max} \text{ e} \\ L'_i &= C'_i - d = C_j - d = L_j \leq L_{max} \end{aligned}$$

e o novo escalonamento é ótimo.

3.3 Fluxo Total de Tempo Ponderado

Nesta seção, resolveremos os problemas $\mathbf{1 \mid outtree \mid \sum w_j C_j}$ e $\mathbf{1 \mid intree \mid \sum w_j C_j}$. Temos que escalonar trabalhos com tempos de processamento arbitrários em uma máquina de modo que a soma ponderada dos tempos de completação seja minimizada. Precedências

são dadas por uma árvore. Inicialmente, assumiremos que essa árvore é uma *outtree* (cada nó da árvore tem no máximo 1 antecessor). Para cada trabalho $i = 1, \ldots, n$ definimos $q_i = w_i/p_i$ e $S(i)$ igual ao conjunto dos sucessores, não necessariamente imediatos, de i, incluindo i. Analogamente, para um conjunto de trabalhos $I \subset \{1, \ldots, n\}$ definimos

$$q(I) = w(I)/p(I), \quad \text{onde} \quad w(I) = \sum_{i \in I} w_i \quad \text{e} \quad p(I) = \sum_{i \in I} p_i.$$

Evidentemente, $w_i = w(i) := w(\{i\})$, $p_i = p(i) := p(\{i\})$ e $q_i = q(i) := q(\{i\})$.

Dois subconjuntos $I, J \subset \{1, \ldots, n\}$ são **paralelos** ($I \sim J$) se, para todo $i \in I, j \in J$, i não é sucessor de j e vice-versa. Os conjuntos paralelos devem ser disjuntos. Denotamos os casos $\{i\} \sim \{j\}$ e $\{i\} \sim J$ simplesmente por $i \sim j$ e $i \sim J$, respectivamente.

Um exemplo de subconjuntos paralelos de nós, é o grafo da Figura 11, página 24, onde $\{1, 2\}$ e $\{3, 4, 5\}$ são tais subconjuntos.

Nas discussões seguintes, um escalonamento será representado por uma sequência π.

Lema 3.3.1. *Sejam π uma sequência ótima e I, J dois blocos de π de modo que I é processado imediatamente antes de J. Seja π' a sequência que se obtém de π trocando I com J de posição. Assim*

(a) $I \sim J \Rightarrow q(I) \geq q(J)$;

(b) Se $I \sim J$ e $q(I) = q(J)$, então π' é também ótima.

Demonstração. (a) Primeiramente, a sequência π' é viável pois, como $I \sim J$, não há quebra de precedência ao trocarmos I e J de posição, e também não há tempos de liberação para os trabalhos (todos podem ser processados a partir do tempo $t = 0$). Denote por f a função objetivo $\sum w_j C_j$. Temos $f(\pi) \leq f(\pi')$ pois a sequência π é ótima. O tempo de completação $C'(k)$ de um trabalho $k \in I$ em π' é $C(k) + p(J)$ devido ao deslocamento do bloco I em $p(J)$ unidades de tempo para direita. Da mesma forma, $C'(k) = C(k) - p(I)$, $\forall\ k \in J$. Assim

$$\begin{aligned} 0 \leq f(\pi') - f(\pi) = &\left(\sum_{\substack{k \in \{1,\ldots,n\} \\ k \notin I,J}} w(k)C'(k) \right) + \left(\sum_{k \in I} w(k)C'(k) \right) + \left(\sum_{k \in J} w(k)C'(k) \right) \\ &- \left(\sum_{\substack{k \in \{1,\ldots,n\} \\ k \notin I,J}} w(k)C(k) \right) - \left(\sum_{k \in I} w(k)C(k) \right) - \left(\sum_{k \in J} w(k)C(k) \right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{k \in I} w(k)p(J) - \sum_{k \in J} w(k)p(I) \\
&= w(I)p(J) - w(J)p(I) \\
&\Rightarrow 0 \leq \frac{w(I)p(J) - w(J)p(I)}{p(I)p(J)} = \frac{w(I)}{p(I)} - \frac{w(J)}{p(J)} \\
&\Rightarrow q(I) \geq q(J).
\end{aligned}
$$

(b) Se $q(I) = q(J)$ então $w(I)p(J) = w(J)p(I)$ e, pelas contas do item anterior, temos $f(\pi) = f(\pi')$. Logo π' é sequência ótima. □

Teorema 3.3.1. *Sejam i, j trabalhos com $i \to j$ e $q_j = \max\{q_k; k \in S(i)\}$. Então existe um escalonamento ótimo no qual i é processado imediatamente antes de j.*

Demonstração. Seja π uma sequência ótima com a propriedade de que o número l de trabalhos escalonados entre i e j seja mínimo. Assuma por absurdo que $l > 0$. Temos a seguinte situação:

	i	$\cdots$	k	j	

Lembrando que $S(i)$ é o conjunto dos sucessores de i, incluindo i, por hipótese temos $i, j \in S(i)$. Para k entre i e j, consideremos dois casos:

Caso 1: $k \in S(i)$. Estamos tratando o caso em que as precedências são dadas por uma *outtree* e como $i \to j$, não pode ser $k \to j$, pois se fosse, j teria dois antecessores. Claramente não pode ser $j \to k$, pois na sequência ótima π, k é escalonado antes de j. Assim, temos $k \sim j$ e pelo lema anterior, $q_k \geq q_j$.

Como $q_j = \max\{q_k; k \in S(i)\}$, temos que $q_j \geq q_k$ e logo $q_j = q_k$. Novamente pelo lema anterior, podemos trocar j com k e ainda ter uma sequência ótima, o que contradiz a minimalidade de l.

Caso 2: $k \notin S(i)$. Seja h o último trabalho pertencente a $S(i)$ que é escalonado antes de j. Assim, para todo r pertencente ao conjunto K dos trabalhos que são escalonados entre h e j, temos $r \notin S(i)$.

Nesse momento temos a seguinte ordem de trabalhos:

	i	$\cdots$	h	$\cdots$	r	$\cdots$	k	j	

Como as relações de precedência formam uma *outtree* e $i \rightarrow j$, segue que todo antecessor não imediato de j é também de i, exceto o próprio i, pois caso contrário j teria dois ou mais antecessores imediatos. Assim, todo $r \in K$ não pode ser antecessor de j pois, se fosse, r seria antecessor de i e não poderia ser escalonado depois de i, sendo π uma sequência ótima. Como também $r \notin S(i)$, temos $K \sim j$ e pelo lema anterior $q(K) \geq q_j$.

Por termos $h \in S(i)$, h não é antecessor de qualquer trabalho $r \in K$, pois se fosse, r pertenceria a $S(i)$. Visto que todo r é escalonado depois de h, não pode ser antecessor de h. Dessa forma, $h \sim K$, o que implica, pelo lema anterior

$$q_h \geq q(K) \geq q_j.$$

Como $q_j = \max\{q_k; k \in S(i)\}$, segue que

$$q_j \geq q_h.$$

Desta forma, $q_j = q_h$ e então $q_j = q(K)$. Pelo lema anterior, podemos trocar j com o bloco K e ainda assim teremos uma sequência ótima, contradizendo a minimalidade de l.

Concluímos assim que deve ser $l = 0$, ou seja, entre i e j não há trabalhos escalonados. □

As condições do teorema anterior são satisfeitas se escolhermos um trabalho j, diferente da raiz, com máximo valor q_j. Como existe uma sequência ótima em que i é escalonado imediatamente antes de j, unimos os nós i e j e fazemos dos filhos de j, filhos adicionais de i. O novo nó i, que representa a sequência $\pi_i : i, j$, terá $q(i) = q(J_i)$, onde $J_i = \{i, j\}$. Este processo de junção dos trabalhos é aplicado recursivamente. Exibiremos adiante um algoritmo para realizar tal tarefa. Antes, precisaremos de algumas notações:

- $E(i)$ denotará o último trabalho de π_i;
- $P(i)$ denotará inicialmente o antecessor de i com respeito à relação de precedência, e finalmente, quando todas as junções tiverem sido realizadas, com respeito à solução ótima obtida. Assim, os $P(i)$'s dirão ao fim como obtermos a sequência ótima calculada.

Assumimos que $i = 1$ é a raiz da árvore, ou seja, o único nó que não possui antecessores.

Algoritmo 6 : 1 | outtree | $\sum w_j C_j$

1: $w(1) := -\infty$;
2: **Para** $i = 1$ **até** n **faça**
3: **Início**
4: $E(i) := i$;
5: $J_i := \{i\}$;
6: $q(i) := w(i)/p(i)$;
7: **Fim**
8: $L := \{1, \ldots, n\}$;
9: **Enquanto** $L \neq \{1\}$ **faça**
10: **Início**
11: Encontre $j \in L$ com maior valor $q(j)$;
12: $f := P(j)$;
13: Encontre i tal que $f \in J_i$;
14: $w(i) := w(i) + w(j)$;
15: $p(i) := p(i) + p(j)$;
16: $q(i) := w(i)/p(i)$;
17: $P(j) := E(i)$;
18: $E(i) := E(j)$;
19: $J_i := J_i \cup J_j$;
20: $L := L - \{j\}$;
21: **Fim**

Inicialmente, definimos $w(1) = -\infty$ para evitar a escolha da raiz na linha 11. De fato, ela não pode ser escolhida pois não possui antecessor, o que inviabilizaria a execução da linha 12.

A sequência ótima π^* é construída usando $E(1)$ e o vetor $P(i)$. Isso é feito da direita para a esquerda: $j := E(1)$ é o último trabalho em π^*, $P(j)$ é o antecessor de j, etc. A Figura 14 a seguir mostra como o algoritmo funciona.

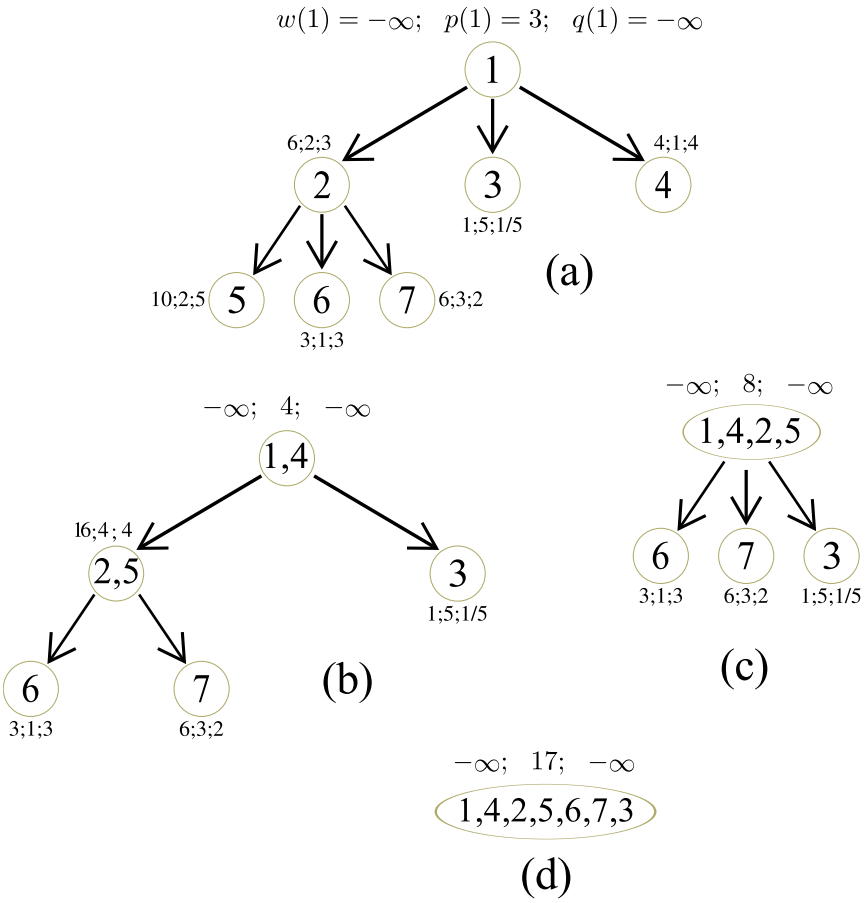


Figura 14: Aplicação do algoritmo no grafo inicial (a); as duas primeiras junções (b); a terceira (c); e finalmente o término do algoritmo (d).

Observando a figura, vemos que no último passo, quando o algoritmo encerra seu procedimento, temos $E(1) = 3$ e assim, a sequência é construída a partir de $P(3)$, que no exemplo é 7. Na sequência, $P(7) = 6$, $P(6) = 5$, $P(5) = 2$, $P(2) = 4$ e $P(4) = 1$, gerando o escalonamento 1,4,2,5,6,7,3.

A seguir provamos que, após o término, a sequência π obtida pelo algoritmo é ótima.

Teorema 3.3.2. *O Algoritmo* **1 | outtree |** $\sum \boldsymbol{w_j C_j}$ *gera uma sequência ótima.*

Demonstração. Usaremos indução sobre o número de trabalhos.

Claramente, o algoritmo está correto para um único trabalho: ao fim teremos somente $E(1)$ definido, que forma a sequência ótima. Seja $\mathcal{P}$ um problema com n trabalhos e suponha que o algoritmo esteja correto para problemas com $n-1$ trabalhos. Assuma que i, j sejam os primeiros trabalhos a serem unidos pelo algoritmo e seja $\mathcal{P}'$ o problema resultante com $n-1$ trabalhos, onde i é substituído por $J_i = \{i, j\}$ (nessa ordem) com

$w(J_i) = w(i) + w(j)$ e $p(J_i) = p(i) + p(j)$.

Sejam $\mathcal{R}$ o conjunto das sequências da forma

$$\pi : \pi(1), \ldots, \pi(l),\ i,\ j,\ \pi(l+3), \ldots, \pi(n)$$

e $\mathcal{R}'$ o conjunto das sequências da forma

$$\pi' : \pi(1), \ldots, \pi(l),\ J_i,\ \pi(l+3), \ldots, \pi(n),$$

onde $J_i = \{i, j\}$ $(i \to j)$. Ou seja, $\mathcal{R}'$ contém as mesmas sequências de $\mathcal{R}$, mas com os trabalhos i e j juntados.

O par (i, j) juntado pelo algoritmo satisfaz $q_j = \max\{q_k; k \in S(i)\}$ (linhas 11 a 13) e logo pelo Teorema 3.3.1, existe uma sequência ótima em $\mathcal{R}$. Seja $\pi' \in \mathcal{R}'$ construído pelo algoritmo. Pela hipótese de indução, π' é ótimo para $\mathcal{P}'$ pois é uma sequência de $n-1$ trabalhos construída pelo algoritmo. Mostremos que $\pi \in \mathcal{R}$ relacionado a este π' é ótimo para o problema $\mathcal{P}$ de n trabalhos. Suponha por absurdo que $\overline{\pi} \in \mathcal{R}$ seja tal que $f_n(\overline{\pi}) < f_n(\pi)$. Podemos tomar $\overline{\pi}$ pois é sabido que existe uma sequência ótima em $\mathcal{R}$. Temos

$$\begin{aligned}
0 &> f(\overline{\pi}) - f(\pi) \\
&= \sum_{k \neq i,j} w_k \overline{C}_k + w_i \overline{C}_i + w_j \overline{C}_j - \sum_{k \neq i,j} w_k C_k - w_i C_i - w_j C_j + (w_i p(j) - w_i p_j) \\
&= \sum_{k \neq i,j} w_k \overline{C}_k + (w_i + w_j)\overline{C}_j - \sum_{k \neq i,j} w_k C_k - (w_i + w_j) C_j \\
&= \sum_{k \neq i,j} w_k \overline{C}_k + w(J_i)\overline{C}_j - \sum_{k \neq i,j} w_k C_k - w(J_i) C_j \\
&= f_{n-1}(\overline{\pi}) - f_{n-1}(\pi')
\end{aligned}$$

Contrariando a otimalidade de π' para $\mathcal{P}'$, com $n-1$ trabalhos. Assim, a sequência ótima para o problema $\mathcal{P}$, com n trabalhos, é obtida desmembrando o trabalho J_i em i e j, ou seja, a sequência π. □

Por fim, para resolver um problema $\mathcal{P}$ do tipo **1 | intree | $\sum \boldsymbol{w_j C_j}$**, reduzimos $\mathcal{P}$ a um problema $\mathcal{P}'$ do tipo **1 | outtree | $\sum \boldsymbol{w_j C_j}$** da seguinte forma:

- i é sucessor de j em $\mathcal{P}' \iff j$ é sucessor de i em $\mathcal{P}$
- $w'_j = -w_j$, $j = 1, \ldots, n$.

A sequência $\pi : 1, \ldots, n$ é viável para $\mathcal{P}$ se e somente se $\pi' : n, \ldots, 1$ é viável para $\mathcal{P}'$, já que $i \to j$ em $\mathcal{P}$ implica $j \to i$ em $\mathcal{P}'$. Temos ainda

$$\begin{aligned} f'(\pi') &= \sum_{i=1}^{n}(-w_i)\big(\sum_{j\geq i} p_j\big) = \sum_{i=1}^{n}(-w_i)\big(\sum_{j>i} p_j\big) - \sum_{i=1}^{n} w_i p_i \\ &= -\sum_{i=1}^{n} w_i\big(\sum_{j>i} p_j\big) + \sum_{i=1}^{n} w_i\big(\sum_{j=1}^{n} p_j\big) - \sum_{i=1}^{n} w_i\big(\sum_{j=1}^{n} p_j\big) - \sum_{i=1}^{n} w_i p_i \\ &= \sum_{i=1}^{n} w_i\big(\sum_{j\leq i} p_j\big) - \sum_{i=1}^{n} w_i\big(\sum_{j=1}^{n} p_j\big) - \sum_{i=1}^{n} w_i p_i \\ &= f(\pi) - \sum_{i=1}^{n} w_i\big(\sum_{j=1}^{n} p_j\big) - \sum_{i=1}^{n} w_i p_i \end{aligned}$$

Observe que a parcela $\sum_{i=1}^{n} w_i\big(\sum_{j=1}^{n} p_j\big) + \sum_{i=1}^{n} w_i p_i$ não depende da sequência π escolhida. Isto se deve ao fato de os valores w_i não mudarem de acordo com o tempo de completação do trabalho i e também pelo fato de $\sum_{j=1}^{n} p_j$ ser um valor fixo, que é a soma dos tempos de completação de todos os trabalhos. Assim, uma sequência π para $\mathcal{P}$ é ótima se e somente se a função reversa π' é ótima para $\mathcal{P}'$.

4 Problemas de escalonamento em máquinas paralelas

Neste capítulo, discutiremos problemas de escalonamento em máquinas paralelas, resumindo-se aos casos com máquinas idênticas (veja seção 2.4 para definição). Nessa classe de problemas, o único problema não preemptivo com tempos de processamento arbitrários que pode ser resolvido polinomialmente, até onde se sabe, é o $\boldsymbol{P||\sum C_i}$ (BRUCKER, 2006). Por esse motivo, nos restringiremos aos casos com preempção ou tempos de processamento unitários.

4.1 $P \mid \text{pmtn}; r_i \mid L_{\max}$

Associado a cada trabalho i, existe um tempo de liberação r_i e uma data final de conclusão d_i com $r_i \leq d_i$. Temos que encontrar um escalonamento preemptivo em m máquinas idênticas de modo que o máximo atraso $\max_{i=1}^{n}\{C_i - d_i\}$ seja minimizado. Para este fim, primeiro consideraremos a versão de decisão deste problema: Dado um valor pré-fixado L, existe um escalonamento de modo que

$$\max_{i=1}^{n} L_i = \max_{i=1}^{n}\{C_i - d_i\} \leq L \ ?$$

A equação acima vale se, e somente se,

$$C_i \leq d_i^L := L + d_i, \quad \forall i = 1, \ldots, n.$$

Assim, cada trabalho i deve ser completado antes da sua data final modificada d_i^L e não pode ser iniciado antes do seu tempo de liberação r_i, ou seja, deve ser processado no intervalo $[r_i, d_i^L]$, o qual chamaremos de **janela de tempo**.

Agora somos guiados ao problema geral de encontrar um escalonamento preemptivo para n trabalhos em m máquinas idênticas de modo que todos os trabalhos i sejam

processados em suas respectivas janelas de tempo. Tal problema pode ser reduzido a um problema de fluxo máximo em rede[1], construído como segue. Fazemos a união exclusiva entre os conjuntos $\{r_i;\ i = 1, \ldots, n\}$ e $\{d_i;\ i = 1, \ldots, n\}$, onde por simplicidade os d_i's são as datas finais modificadas, e então ordenamos o conjunto obtido, digamos

$$t_1 < t_2 < \ldots < t_r.$$

Consideramos assim os intervalos

$$I_K := [t_K, t_{K+1}] \ \text{ de tamanho } \ T_K = t_{K+1} - t_K, \ \ \forall K = 1, \ldots, r-1.$$

Construímos o grafo do problema de fluxo associando um vértice a cada trabalho i e um vértice a cada intervalo I_K. Adicionamos ainda dois vértices artificiais s e t (fonte e sumidouro, respectivamente). Partindo de s, temos um arco para cada vértice de trabalho i, com capacidade p_i. Partindo de cada vértice de intervalo I_K, temos um arco para t com capacidade mT_K. Entre os vértices de trabalho e de intervalo, existe um arco de i para I_K se, e somente se, o trabalho i pode ser processado no intervalo I_K, isto é, se $r_i \leq t_K$ e $t_{K+1} \leq d_i$. A capacidade desse arco será T_K. Denotaremos a rede construída por $N = (V, A)$, onde V é o conjunto dos vértices e A o conjunto dos arcos. A Figura 15 representa uma típica rede N.

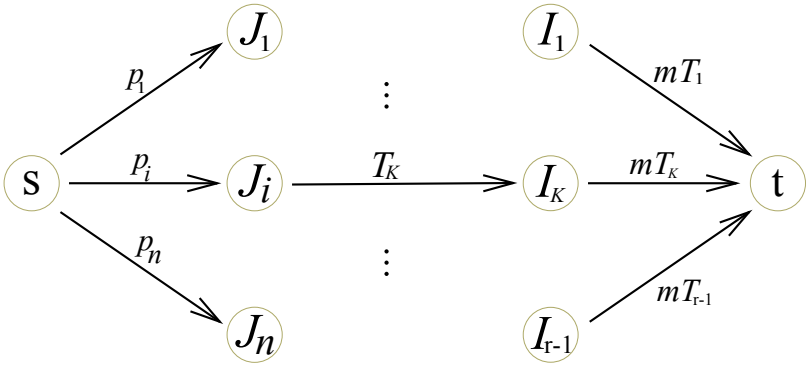


Figura 15: Rede N do problema de fluxo relacionado à $\boldsymbol{P \mid \text{pmtn; } r_i \mid L_{\max}}$.

Seja x_{iK} o fluxo no arco (i, I_K) correspondente ao período no qual o trabalho i é processado no intervalo I_K, para $i = 1, \ldots, n$ e $K = 1, \ldots, r-1$. Se existe um escalonamento viável para o problema $\boldsymbol{P \mid \text{pmtn; } r_i \mid L_{\max}}$, todos os trabalhos devem

[1]Uma referência para problema de fluxo máximo é o livro de Bazaraa, Jarvis e Sherali (2010).

ser processados por completo, ou seja,

$$\sum_{K=1}^{r-1} x_{iK} = p_i, \quad i = 1, \ldots, n. \tag{4.1}$$

Em termos do problema de fluxo, essa igualdade diz que o fluxo que sai de cada vértice de trabalho i deve ser p_i. Pelas restrições de conservação de fluxo, o fluxo no arco (s, i) deve também ser p_i (veja Figura 15). Tal situação corresponde ao fluxo $\sum_{i=1}^{n} p_i$ na rede, o máximo possível. Assim, se existe um escalonamento viável então o fluxo máximo na rede N é $\sum_{i=1}^{n} p_i$, característica capturada por (4.1). Reciprocamente, se o fluxo máximo é $\sum_{i=1}^{n} p_i$ então em cada arco (s, i) corre p_i unidades de fluxo, e novamente pela conservação de fluxo, deve valer (4.1), ou seja, todo trabalho é processado por completo.

Outras restrições são aquelas que referem-se à capacidade dos arcos, típicas de problemas de fluxo. Respeitando as capacidades dos arcos entre os vértices de intervalo e t, devemos ter

$$\sum_{i=1}^{n} x_{iK} \leq mT_K, \quad K = 1, \ldots, r-1. \tag{4.2}$$

Da mesma forma, a capacidade dos arcos entre vértices de trabalho e de intervalo fornece, pelo fato de que um trabalho não pode ser processado simultaneamente em mais de uma máquina,

$$x_{iK} \leq T_K, \quad \forall (i, I_K) \in A. \tag{4.3}$$

As restrições obtidas acima são típicas do problema de fluxo. É interessante interpretá-las sob a ótica do problema de interesse $\boldsymbol{P}$ **| pmtn;** $\boldsymbol{r_i}$ **|** $\boldsymbol{L_{max}}$, como segue. Como dissemos, a restrição (4.1) nos diz que cada trabalho i é totalmente processado, já que a soma de suas partes processadas é o seu tempo de processamento total p_i; (4.2) garante que a capacidade total de processamento das m máquinas seja satisfeita em cada janela de tempo I_K; (4.3) garante que a parte do trabalho i processada em I_K não extrapole o tamanho do intervalo, garantindo que é possível encaixar o trabalho i na janela de tempo $I_K = [t_K, t_{K+1}]$.

Resolvido o problema de fluxo máximo, resta-nos obter uma solução para o problema de escalonamento. Pois bem, se existe um fluxo máximo satisfazendo (4.1) a (4.3), uma solução viável para o problema $\boldsymbol{P}$ **| pmtn;** $\boldsymbol{r_i}$ **|** $\boldsymbol{L_{max}}$ é construída escalonando partes do trabalho i com tempos de processamento x_{iK} nos intervalos I_K em alguma máquina disponível. Note que, sendo as máquinas idênticas, qualquer máquina pode ser escolhida

para processar partes de qualquer trabalho em qualquer janela de tempo.

Agora, lembremos que primeiramente foi estabelecida a versão de decisão do problema original, fixando L *a priori* e modificando as datas finais. O problema de fluxo foi então construído. Logo podemos esperar que o problema de fluxo forneça apenas um escalonamento viável, sem a garantia de ser ótimo. No entanto, diminuindo L e repetindo todo o processo, duas situações podem ocorrer: ou obtemos um novo escalonamento; ou nos deparamos com um problema de fluxo sem solução. No primeiro caso, melhoramos o escalonamento. No segundo caso, garantimos que para o novo L não há escalonamento viável. Mediante tais observações estabelecemos os seguintes passos para encontrar um escalonamento ε-ótimo, com precisão arbitrária $\varepsilon > 0$:

1. Fixamos um L inicial suficientemente pequeno e modificamos as datas finais de conclusão originais fazendo $d_i^L := L + d_i$. Podemos assumir sem perda de generalidade que $d_i \leq r_i + n \max_{j=1}^{n} p_j$, o que implica $-n \max_{j=1}^{n} p_j \leq L_{max} \leq n \max_{j=1}^{n} p_j$. Assim, podemos tomar inicialmente $L = -n \max_{j=1}^{n} p_j$.
2. Resolvemos o problema de fluxo máximo. Se uma solução for encontrada, construímos o escalonamento associado, e este será ε-ótimo. Paramos assim o procedimento. Se não houver solução, aumentamos L para $L + \varepsilon$ e repetimos o processo.

Cabe observar que

- poderíamos estabelecer um procedimento análogo iniciando L suficientemente grande e diminuindo-o ao longo do processo. Pararíamos na primeira vez em que o problema de fluxo fosse inviável;
- assumindo $d_i \leq r_i + n \max_{j=1}^{n} p_j$, se não houver solução para o problema de fluxo com $L \in \left[-n \max_{j=1}^{n} p_j,\ n \max_{j=1}^{n} p_j\right]$ então o problema de escalonamento original não possui solução.

O exemplo a seguir ilustra a construção do problema de fluxo associado, sua resolução, e a obtenção de um escalonamento.

Exemplo 4.1.1. Considere a instância de $\boldsymbol{P}$ **| pmtn;** $\boldsymbol{r_i}$ **|** $\boldsymbol{L_{\max}}$ com 2 máquinas e 6 trabalhos, cujos dados são

i	1	2	3	4	5	6
p_i	2	5	6	3	4	3
r_i	1	2	1	0	2	3
d_i	8	9	15	12	13	11

Vamos resolver uma sequência de problemas de decisão. Precisamos fixar um valor para L e modificarmos as datas finais fazendo $d_i^L = L + d_i$, $i = 1, \ldots, 6$.

Fixando $L = -4$, é fácil ver que o problema de escalonamento é inviável, já que é impossível escalonar o trabalho 2 na sua janela de tempo $[r_2, d_2^L] = [2, 5]$, visto que seu tempo de processamento é 5 e que o problema não permite que um trabalho seja processado em mais de uma máquina simultaneamente. Por esse motivo, devemos aumentar o valor de L até conseguirmos um escalonamento viável, e como os dados do problema são todos inteiros (geralmente são, como dito na seção 2.2), o aumento de L será de uma unidade e o primeiro escalonamento encontrado será ótimo.

Quando $L = -3$ é possível ver que o problema de escalonamento é inviável. Para $L = -2$, temos $d^L = (6, 7, 13, 10, 11, 9)$ e então a união exclusiva dos conjuntos $\{r_i;\ i = 1, \ldots, n\}$ e $\{d_i^L;\ i = 1, \ldots, n\}$ resulta no conjunto

$$\{0, 1, 2, 3, 6, 7, 9, 10, 11, 13\},$$

onde definimos os seguintes intervalos: $I_1 = [0, 1],\ I_2 = [1, 2],\ I_3 = [2, 3],\ I_4 = [3, 6],$ $I_5 = [6, 7],\ I_6 = [7, 9],\ I_7 = [9, 10],\ I_8 = [10, 11],$ e $I_9 = [11, 13]$.

Lembrando ainda que os únicos arcos (i, I_K) existentes são aqueles em que $I_K = [t_K, t_{K+1}] \subset [r_i, d_i^L]$, a rede que se obtém desse problema é dada na Figura 16. Os intervalos são tais que (a partir daqui, d_i será usado para d_i^L)

$$\begin{aligned}
[r_1, d_1] &= [1, 6] = I_2 \cup I_3 \cup I_4 \\
[r_2, d_2] &= [1, 7] = I_2 \cup I_3 \cup I_4 \cup I_5 \\
[r_3, d_3] &= [1, 13] = I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8 \cup I_9 \\
[r_4, d_4] &= [0, 10] = I_1 \cup I_2 \cup I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \\
[r_5, d_5] &= [2, 11] = I_3 \cup I_4 \cup I_5 \cup I_6 \cup I_7 \cup I_8 \\
[r_6, d_6] &= [3, 9] = I_4 \cup I_5 \cup I_6
\end{aligned}$$

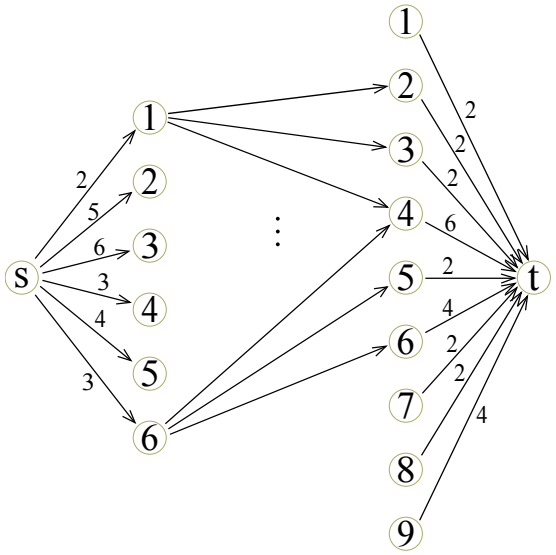


Figura 16: Rede do problema de fluxo do Exemplo 4.1.1. Omitimos alguns arcos.

O problema de fluxo máximo associado consiste em encontrar uma solução do sistema

$$\left.\begin{aligned} x_{1,4}+x_{1,3}+x_{1,2} &= 2\\ x_{2,5}+x_{2,4}+x_{2,3} &= 5\\ x_{3,9}+x_{3,8}+x_{3,7}+x_{3,6}+x_{3,5}+x_{3,4}+x_{3,3}+x_{3,2} &= 6\\ x_{4,7}+x_{4,6}+x_{4,5}+x_{4,4}+x_{4,3}+x_{4,2}+x_{4,1} &= 3\\ x_{5,8}+x_{5,7}+x_{5,6}+x_{5,5}+x_{5,4}+x_{5,3} &= 4\\ x_{6,6}+x_{6,5}+x_{6,4} &= 3 \end{aligned}\right\} \quad (4.1)$$

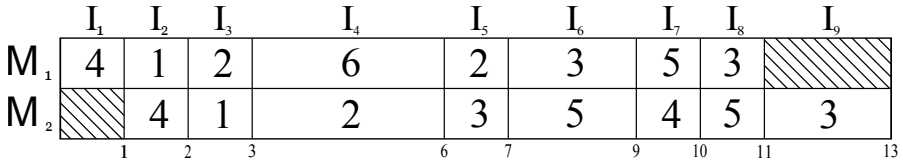
$$\left.\begin{aligned} x_{4,1} &\leq 2\\ x_{4,2}+x_{3,2}+x_{1,2} &\leq 2\\ x_{5,3}+x_{4,3}+x_{3,3}+x_{2,3}+x_{1,3} &\leq 2\\ x_{6,4}+x_{5,4}+x_{4,4}+x_{3,4}+x_{2,4}+x_{1,4} &\leq 6\\ x_{6,5}+x_{5,5}+x_{4,5}+x_{3,5}+x_{2,5} &\leq 2\\ x_{6,6}+x_{5,6}+x_{4,6}+x_{3,6} &\leq 4\\ x_{5,7}+x_{4,7}+x_{3,7} &\leq 2\\ x_{5,8}+x_{3,8} &\leq 2\\ x_{3,9} &\leq 4\\ x_{1,2} &\leq 1 \end{aligned}\right\} \quad (4.2)$$

$$\left.\begin{aligned} x_{1,3},\ x_{2,3},\ x_{2,5},\ x_{3,2},\ x_{3,3},\ x_{3,5},\ x_{3,7},\ x_{3,8},\ x_{4,1},&\\ x_{4,2},\ x_{4,3},\ x_{4,5},\ x_{4,7},\ x_{5,3},\ x_{5,5},\ x_{5,7},\ x_{5,8},\ x_{6,5} &\leq 1\\ x_{1,4},\ x_{2,4},\ x_{3,4},\ x_{4,4},\ x_{5,4},\ x_{6,4} &\leq 3\\ x_{3,6},\ x_{3,9},\ x_{4,6},\ x_{5,6},\ x_{6,6} &\leq 2 \end{aligned}\right\} \quad (4.3)$$

Resolvendo-o encontramos a seguinte matriz solução:

$$x = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 1 & 2 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 1 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

que nos diz que no intervalo I_j, há uma porção de tamanho x_{ij} do trabalho i sendo processada. Com a solução x em mãos, temos infinitos possíveis escalonamentos, todos ótimos, bastando apenas obedecer às porções de cada trabalho em cada intervalo. Um desses escalonamentos é



□

4.2 $\boldsymbol{P \mid p_i = 1;\, r_i \mid L_{\max}}$

É conveniente assumir que os trabalhos estejam indexados de modo que

$$r_1 \leq r_2 \leq \cdots \leq r_n.$$

É fácil obter uma solução se todos os tempos de liberação r_i forem inteiros, escalonando trabalhos disponíveis em ordem não decrescente dos prazos de conclusão. Mais especificamente, se no tempo corrente t existe alguma máquina disponível, e existe um trabalho ainda não escalonado i com $r_i \leq t$, escalona-se o trabalho com menor prazo. A seguir exibimos o algoritmo para tal problema, onde K é o número de máquinas ocupadas no tempo t, m é o número de máquinas, M é o conjunto de trabalhos que podem ser escalonados no tempo t mas que ainda não foram escalonados, e j é um contador do número de trabalhos já escalonados durante todo o processo.

Algoritmo 7 : $\boldsymbol{P \mid p_i = 1;\ r_i}$ **inteiro** $\boldsymbol{\mid L_{max}}$

1: $j := 1$;
2: **Enquanto** $j \leq n$ **faça**
3: **Início**
4: $t := r_j$; $M := \{i;\ i$ não foi escalonado e $r_i \leq t\}$; $K := 1$;
5: **Enquanto** $M \neq \emptyset$ **faça**
6: **Início**
7: Encontre um trabalho i em M com menor prazo;
8: $M := M - \{i\}$;
9: Escalone i no tempo t;
10: $j := j + 1$;
11: **Se** $K + 1 \leq m$ **então** $K := K + 1$;
12: **Caso contrário**
13: **Início**
14: $t := t + 1$;
15: $K := 1$;
16: $M := M \cup \{i;\ i$ não foi escalonado e $r_i \leq t\}$;
17: **Fim**
18: **Fim**
19: **Fim**

No laço das linhas 5 a 18, são criados blocos de trabalhos que são processados sem tempo ocioso entre eles. Após terminar um bloco, o valor r_j corrente é o tempo de início do próximo bloco.

A linha 11 testa se há máquinas disponíveis no tempo t. Se houverem, incrementa o contador K de máquinas ocupadas. Caso contrário, avança ao próximo tempo $t + 1$ e refaz o conjunto das possíveis máquinas a serem escalonadas. Note que K passa a ser 1 novamente, indicando que todas as máquinas estão disponíveis em $t + 1$.

Na linha 9, podemos escalonar o trabalho i, no tempo t, em qualquer máquina disponível, já que todas são idênticas. Particularmente, podemos escalonar i na máquina K. Desta forma, K pode ser usado para indicar em qual máquina escalonaremos cada trabalho escolhido.

Teorema 4.2.1. *O Algoritmo* $\boldsymbol{P \mid p_i = 1;\ r_i}$ **inteiro** $\boldsymbol{\mid L_{max}}$ *constrói um escalonamento ótimo.*

Demonstração. Seja S um escalonamento construído pelo algoritmo. Reindexando os trabalhos se necessário, supomos que o algoritmo escalone os trabalhos $1, 2, \ldots, n$, nessa ordem. Seja S^* um escalonamento ótimo com as seguintes propriedades:

- os primeiros $r-1$ trabalhos escalonados em S são escalonados no mesmo tempo em ambos os escalonamentos S e S^*. Ou seja, a menos da ordem das máquinas, os primeiros $r-1$ trabalhos de S e S^* são os mesmos, escalonados nos mesmos tempos;
- $r-1$ é máximo.

Assim, o trabalho r é escalonado em algum tempo t em S. Como o algoritmo preenche as máquinas sempre que possível em ordem crescente no tempo, r é escalonado em algum tempo posterior a t em S^*, digamos s. Consideremos dois casos:

Caso 1: alguma máquina em S^* está ociosa no tempo t. Nesse caso, o trabalho r em S^* pode ser movido para tal máquina e processado no tempo t. Seja $\overline{S}$ o escalonamento obtido. Sendo $\overline{L}_r = \overline{C}_r - d_r$ e $L_r^* = C_r^* - d_r$ os atrasos do trabalho r nos escalonamentos $\overline{S}$ e S^*, respectivamente, e $L^*_{\max}$ o valor ótimo do problema, temos

$$L^*_{max} \geq L^*_r = C^*_r - d_r = s + 1 - d_r > t + 1 - d_r = \overline{C}_r - d_r = \overline{L}_r.$$

Caso 2: todas as máquinas em S^* estão ocupadas no tempo t. Nesse caso existe algum trabalho k que é processado no tempo t em S^* mas em S não é processado no tempo t. Da linha 7 do algoritmo, devemos ter $d_r \leq d_k$. Podemos então trocar k e r de posição em S^*, obtendo um novo escalonamento $\overline{S}$. Nesse caso $\overline{C}_k = s + 1$ e $\overline{C}_r = t + 1$, fornecendo

$$L^*_{max} \geq L^*_r = C^*_r - d_r = s + 1 - d_r \geq s + 1 - d_k = \overline{C}_k - d_k = \overline{L}_k$$

e

$$L^*_{max} \geq L^*_r = C^*_r - d_r = s + 1 - d_r > t + 1 - d_r = \overline{C}_r - d_r = \overline{L}_r.$$

Das desigualdades obtidas, em ambos os casos o escalonamento modificado $\overline{S}$ é ótimo. No entanto, $\overline{S}$ e S têm os $r > r-1$ primeiros trabalhos escalonados no mesmo tempo, contrariando a maximalidade de $r-1$. Portanto o algoritmo está correto. □

Por fim, é possível adaptar o Algoritmo 7 para tempos de liberação reais não inteiros (BRUCKER, 2006).

4.3 P | intree; $p_i = 1$ | $L_{\max}$

4.3.1 Preliminares

Consideraremos agora os problemas de n trabalhos com relações de precedência. Lembramos que escrevemos $i \to j$ se j é um sucessor imediato de i, ou equivalentemente, se i é um antecessor imediato de j. Os conjuntos $IP(i)$ e $IS(i)$ de todos os antecessores imediatos e sucessores imediatos, respectivamente, do trabalho i, serão denotados por

$$IP(i) = \{j;\ j \to i\} \ \text{ e } \ IS(i) = \{j;\ i \to j\}.$$

Quando as relações de precedência são dadas por uma *intree*, $IS(i)$ contém no máximo um trabalho, e quando são dadas por uma *outtree*, $IP(i)$ tem tal propriedade. Nesta seção consideramos apenas um problema sobre uma *intree*. Para simplificar a notação, denotaremos por $s(i)$ o único sucessor de i na *intree*, caso exista. Caso não exista, escrevemos $s(i) = 0$ (isto é, $IS(i) = \emptyset$).

As datas finais de conclusão d_i são ditas **consistentes** em relação às relações de precedência se $d_i \leq d_j - p_j$ sempre que $i \to j$.

4.3.2 Resolução para P | intree; $p_i = 1$ | $L_{\max}$

O procedimento que resolve esse problema tem dois passos. No primeiro, as datas finais de conclusão dos trabalhos são modificadas de modo que fiquem consistentes em relação às relações de precedência. No segundo, os trabalhos são escalonados em ordem não-decrescente das datas finais modificadas.

Os procedimentos aqui exibidos aplicam-se para o caso em que uma ou mais *intree*'s são consideradas nas relações de precedência. Dessa forma, tais procedimentos funcionam tanto para uma árvore quanto para uma floresta (conjunto de árvores). Assim, podem haver várias raízes, uma para cada *intree*. Consideramos o conjunto T de todas as raízes, ou seja, os vértices sem sucessores ($s(i) = 0$).

A ideia do algoritmo de modificação das datas finais é trocar d_i por $\min\{d_i, d_j - 1\}$ sempre que $i \to j$. Isso é feito sistematicamente das raízes para as folhas das *intree*'s, ou seja, das raízes para os vértices sem antecessores ($IP(i) = \emptyset$).

Algoritmo 8 : Modificar d_j

```
1: T := {i; i não tem sucessor};
2: Enquanto T ≠ ∅ faça
3: Início
4:     Escolha um trabalho i ∈ T;
5:     Para todo j ∈ IP(i) faça
6:     Início
7:         d_j := min{d_j, d_i − 1};
8:         T := T ∪ {j};
9:     Fim
10:    T := T − {i};
11: Fim
```

Note que na linha 5, $j \to i$, ou seja, $s(j) = i$. Assim, na linha 7, d_j passa a ser o mínimo entre o próprio d_j e a data final de seu sucessor menos 1.

Denotaremos as datas finais modificadas por d'_i, o que resulta em $d'_i < d'_j$ sempre que $i \to j$.

Lema 4.3.1. *Um escalonamento não tem trabalhos atrasados com respeito às datas finais de conclusão originais se, e somente se, não tem trabalhos atrasados com respeito às datas finais modificadas.*

Demonstração. A volta é trivial pois $d'_i \leq d_i, \forall i \in \{1, \ldots, n\}$.

Para a ida, assumimos sem perda de generalidade que $n, n-1, \ldots, 1$ é a ordem na qual os trabalhos são analisados pelo Algoritmo 8 (linhas 4 e 5), incluindo as raízes. Por exemplo, n é raiz, escolhido na primeira execução da linha 4 e os nós posteriores são seus antecessores, aqueles em $IP(n)$.

Considere um escalonamento sem atrasos com respeito às datas finais de conclusão originais, ou seja, com $C_i \leq d_i,\ \forall i \in \{1, \ldots, n\}$. Suponha que $r > 1$ seja um trabalho tal que $C_i \leq d'_i,\ \forall i \in \{r, \ldots, n\}$. Se existe algum trabalho $j \in \{r, \ldots, n\}$ tal que $r-1 \in IP(j)$, ou equivalentemente, $s(r-1) = j$, então $C_{r-1} \leq C_j - 1 \leq d'_j - 1$. Como também $C_{r-1} \leq d_{r-1}$, então

$$C_{r-1} \leq \min\{d_{r-1}, d_j - 1\} = d'_{r-1}.$$

Se para todo $j \in \{r, \ldots, n\}$ tivermos $r - 1 \notin IP(j)$, então pela ordem de execução do Algoritmo 8, o trabalho $r - 1$ é raiz e a data final d_{r-1} não é alterada. Assim

$d'_{r-1} = d_{r-1} \geq C_{r-1}$. Isto é, se os trabalhos $r, \ldots, n$ não têm atraso com respeito às datas finais modificadas, então os trabalhos $r-1, r, \ldots, n$ também não sofrem atraso. Prosseguindo, obtemos $C_i \leq d'_i,\ \forall i \in \{1, \ldots, n\}$.

Para finalizar falta argumentarmos que tal r existe. De fato, basta tomarmos $r = n$ pois como n é raiz, sua data final não é alterada, e logo $C_n \leq d_n = d'_n$. □

No segundo passo, os trabalhos são escalonados em ordem não-decrescente das datas finais modificadas. Isto é feito escalonando cada trabalho no tempo de início mais cedo disponível, isto é, o tempo mais cedo no qual menos de m tarefas são escalonadas para começar e todos os seus antecessores já tenham sido completados.

No algoritmo, assumimos que os trabalhos estejam numerados de modo que

$$d'_1 \leq d'_2 \leq \ldots \leq d'_n.$$

Mais ainda, F denota o tempo mais cedo no qual alguma máquina está disponível, $r(i)$ o tempo de finalização mais recente de um antecessor de i, $n(t)$ conta o número de trabalhos escalonados no tempo t e $x(i)$ é o tempo de início do trabalho i.

Algoritmo 9 : $\boldsymbol{P}$ **| intree;** $\boldsymbol{p_i = 1}$ **|** $\boldsymbol{L_{\max}}$

```
1:  F := 0;
2:  Para i = 1 até n faça r(i) := 0;
3:  Para t = 1 até n faça n(t) := 0;
4:  Para i = 1 até n faça
5:  Início
6:      t := max{r(i), F};
7:      x(i) := t;
8:      n(t) := n(t) + 1;
9:      Se n(t) = m então F := t + 1;
10:     j := s(i);
11:     r(j) := max{r(j), t + 1};
12: Fim
```

O escalonamento construído por esse algoritmo tem a importante propriedade de que o número de trabalhos escalonados em qualquer tempo é não menor que o número de trabalhos escalonados em tempos posteriores. De fato, suponha por absurdo que $k < m$ trabalhos sejam escalonados para iniciar no tempo t e pelo menos $k+1$ trabalhos são escalonados para iniciar no tempo $t+1$. Como o procedimento escalona trabalhos no

tempo de início mais cedo disponível e menos de m trabalhos são escalonados no tempo t, os $k+1$ trabalhos escalonados no tempo $t+1$ devem ter, cada um, um antecessor imediato escalonado para começar no tempo t. Isso é impossível pela estrutura de *intree*, visto que cada trabalho iniciando no tempo t tem no máximo um sucessor, e logo são possíveis no máximo k sucessores para o tempo $t+1$.

Lema 4.3.2. *Se existe um escalonamento no qual nenhum trabalho é atrasado, então o escalonamento construído pelo Algoritmo 9 tem tal propriedade.*

Demonstração. Assumamos que exista um escalonamento sem atrasos, e suponha por absurdo que há algum trabalho atrasado no escalonamento $x(1), \ldots, x(n)$ construído pelo Algoritmo 9. Então pelo Lema 4.3.1, também existe nesse escalonamento algum trabalho atrasado com respeito às datas finais modificadas. Considere o menor i tal que $x(i)+1 > d'_i$. Seja $t < d'_i$ o maior inteiro com a propriedade de que $\left|\{j;\ x(j) = t,\ d'_j \leq d'_i\}\right| < m$, ou seja, o maior instante com alguma janela ociosa.

Existe t com a propriedade descrita pois caso contrário teríamos md'_i trabalhos j com $d'_j \leq d'_i$ escalonados antes de d'_i. O trabalho i não pertence a esse conjunto pois $x(i)+1 > d'_i$, o que significa que, se quisermos um escalonamento sem trabalhos atrasados, o trabalho i deveria ser escalonado antes de d'_i e logo teríamos $md'_i + 1$ trabalhos a serem escalonados no intervalo $[0, d'_i]$, o que é uma contradição.

Cada trabalho j com $d'_j \leq d'_i$ e $x(j) > t$ deve ter antecessores, não necessariamente imediatos, iniciando no tempo t, pois caso contrário j poderia ser escalonado na janela ociosa no tempo t. Agora consideramos dois casos:

Caso 1: $t = d'_i - 1$. Temos $x(i) > d'_i - 1 = t$. Então um antecessor k de i deve iniciar no tempo t (observação anterior) e terminar no tempo d'_i. Observe que tal k existe pois i satisfaz $d'_i \leq d'_i$ e $x(i) > t$. Como $d'_k < d'_i$ (pelo Algoritmo 8 de modificação das datas finais) e ainda $d'_i = x(k)+1$, então o trabalho k está atrasado também ($d'_k < d'_i = x(k)+1$), o que contradiz a minimalidade de i, já que $d'_k < d'_i \Rightarrow k < i$ (lembre-se que assumimos a ordem $d'_1 \leq \cdots \leq d'_n$ para execução do Algoritmo 9).

Caso 2: $t < d'_i - 1$. Exatamente m trabalhos j com $d'_j \leq d'_i$ iniciam no tempo $t+1$, cada um tendo um antecessor começando no tempo t, pelo mesmo motivo do caso anterior. Pela estrutura de *intree*, todos esses antecessores devem ser diferentes pois se dois trabalhos distintos u e v tivessem o mesmo antecessor, este mesmo antecessor teria u e v como sucessores. Assim, se k é um antecessor de j, pelo Algoritmo 8 de modificação das datas finais, $d'_k < d'_j \leq d'_i$. Logo, no tempo t existem m trabalhos k com $d'_j \leq d'_i$, o que contradiz

a definição de t. □

Teorema 4.3.1. *O Algoritmo* $\boldsymbol{P}$ **| intree;** $\boldsymbol{p_i = 1}$ **|** $\boldsymbol{L_{\max}}$ *constrói um escalonamento ótimo.*

Demonstração. Seja $L^*_{\max}$ o valor ótimo. Então existe um escalonamento satisfazendo

$$\max_{i=1}^{n}\{C_i - d_i\} \leq L^*_{\max} \tag{4.4}$$

que é equivalente a

$$C_i \leq d_i + L^*_{\max}, \quad i = 1, \ldots, n. \tag{4.5}$$

Tal escalonamento não apresenta atrasos em relação às datas finais $d_i + L^*_{\max}$. De acordo com o Lema 4.3.2, um escalonamento S construído pelo Algoritmo 9 para as datas finais $d_i + L^*_{\max}$ satisfaz (4.4), ou equivalentemente, (4.5). Assim, o escalonamento S é ótimo. No entanto, o algoritmo fornecerá o mesmo escalonamento S para as datas finais originais d_i's. De fato, se $j \to i$ então

$$(d_i + L^*_{\max})' = \min\{d_i + L^*_{\max}, d_j + L^*_{\max} - 1\} = \min\{d_i, d_j - 1\} + L^*_{\max} = d'_i + L^*_{\max}.$$

Se i é raiz então

$$(d_i + L^*_{\max})' = d_i + L^*_{\max} = d'_i + L^*_{\max}.$$

De qualquer modo, temos

$$(d_i + L^*_{\max})' = d'_i + L^*_{\max}, \quad i = 1, \ldots, n.$$

Portanto a ordem dos trabalhos estabelecida por

$$d'_1 \leq \cdots \leq d'_n$$

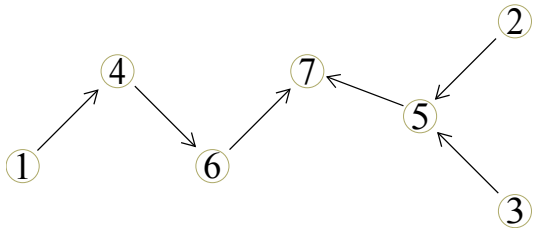
é a mesma que a definida por

$$(d_1 + L^*_{\max})' \leq \cdots \leq (d_n + L^*_{\max})'.$$

Além disso, o Algoritmo 9 não utiliza datas finais em seus cálculos. Resumindo, a aplicação do algoritmo ao problema com datas originais resulta em um escalonamento ótimo, como queríamos. □

4.3.3 Exemplo

Considere o problema $\boldsymbol{P3}$ **| intree;** $\boldsymbol{p_i = 1}$ **|** $\boldsymbol{L_{\max}}$, com 3 máquinas. Queremos escalonar 7 trabalhos com datas finais de conclusão $d = (2, 3, 3, 4, 2, 5, 6)$ e relações de precedência dadas pela *intree* de raiz 7



Primeiramente, aplicamos o Algoritmo 8 de modificação das datas finais obtendo $d' = (2, 1, 1, 4, 2, 5, 6)$. Agora, para aplicar o algoritmo $\boldsymbol{P}$ **| intree;** $\boldsymbol{p_i = 1}$ **|** $\boldsymbol{L_{\max}}$, devemos colocar as datas finais modificadas em ordem não-decrescente e ordenar os trabalhos de acordo com a sequência obtida. Desta forma obtemos o vetor $x = (0, 0, 0, 1, 1, 2, 3)$ que nos diz o tempo de início do processamento de cada trabalho, e assim, podemos construir o seguinte escalonamento:

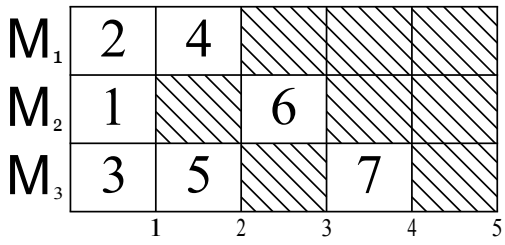


Figura 17: Escalonamento ótimo para o problema $\boldsymbol{P3}$ **| intree;** $\boldsymbol{p_i = 1}$ **|** $\boldsymbol{L_{\max}}$.

Evidentemente, outros escalonamentos ótimos podem ser obtidos a partir de x. Por exemplo, podemos trocar os trabalhos 1, 2 e 3 de posição, ou ainda trocar 6 e 7 de máquina.

Referências

BAZARAA, M. S.; JARVIS, J. J.; SHERALI, H. D. *Linear programming and network flows*. 4. ed. [S.l.]: John Wiley & Sons, 2010.

BRUCKER, P. *Scheduling Algorithms*. 5. ed. [S.l.]: Springer, 2006.

LAWER, E. L. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, v. 19, p. 544–546, 1973.

MULUK, A.; AKPOLAT, H.; XU, J. Scheduling problems: an overview. *Journal of Systems Science and Systems Engineering*, v. 12, n. 4, p. 481–492, 2003.

PINEDO, M. L. *Scheduling. Theory, Algorithms, and Systems*. 3. ed. [S.l.]: Springer, 2008.