5. Penalidades por Antecipação e Atraso

5.1. O Tempo Total de Antecipação e Atraso

Todas as funções objetivo apresentadas até o momento são medidas de desempenho regulares (ou seja, não-decrescentes em C_j para todo *j*). Na prática, pode ocorrer que, se o trabalho *j* for concluído antes de seu prazo de entrega d_j uma penalidade de antecipação é aplicada. A antecipação do trabalho *j* é definida como:

$$E_j = \max (d_j - C_j , 0) \tag{5.1}$$

Dessa forma, a função objetivo é uma generalização do conceito de atraso total, como colocado anteriormente, passando a ser visto como a soma dos custos de antecipação (*earliness)* com os custos dos atrasos (*tardiness)*, acompanhados das ponderações apropriadas, como será discutido a seguir

$$\text{Min } Z = \sum_{j=1}^{n} E_j + \sum_{j=1}^{n} T_j \tag{5.2}$$

Dado que este problema é mais difícil do que o problema do atraso total, faz sentido primeiramente ser feita uma análise de casos especiais que são tratáveis. Considere o caso especial com todos os trabalhos com o mesmo prazo de entrega, ou seja, $d_j = d$ para todo *j*.

Uma programação ideal para este caso especial tem uma série de propriedades úteis. Por exemplo, pode ser facilmente mostrado que, após o primeiro trabalho ser iniciado, os *n* trabalhos devem ser processados sem interrupção, isto é, não deve haver ociosidade forçada entre o processamento de quaisquer dois trabalhos consecutivos. No entanto, é possível que uma programação ideal não inicie imediatamente os processamentos dos trabalhos em

tempo de 0, ou seja, pode-se esperar algum tempo antes que o primeiro trabalho seja iniciado.

A segunda propriedade diz respeito à sequência real dos trabalhos. Qualquer sequência pode ser dividida em dois conjuntos disjuntos de trabalhos e, possivelmente, um trabalho adicional. Um conjunto contém os trabalhos que estão concluídos antecipadamente, ou seja, $C_j \leq d$, e o outro conjunto contém os trabalhos que são iniciados com atraso. O primeiro conjunto de trabalho, que contém os trabalhos iniciados antecipadamente, é chamado de J_1 e um segundo conjunto de trabalhos, contendo os trabalhos iniciados com atraso, é chamado de J_2. Além desses dois conjuntos de trabalhos, pode haver mais um trabalho que é iniciado antecipadamente e finalizado com atraso.

Em um sequenciamento ótimo, os trabalhos do conjunto J_1 estão programados até a data prometida de acordo com o Maior Tempo de Processamento (*Longest Processing Time* – LPT), ou seja, do trabalho com maior tempo de processaento para o trabalho com menor tempo de processamento e os trabalhos do conjunto J_2 são programados por após a data prometida, de acordo com o Menor Tempo de Processamento (*Shortest Processing Time* – SPT), isto é, do trabalho com menor tempo de processamento para o trabalho com maior tempo de processamento. De acordo com essa propriedade, é comumente dito que o sequenciamento ótimo possui um perfil "V", como a Figura 14.

Por muitos anos, as pesquisas de programação eram focadas em medidas de desempenho individuais, referidas como medidas regulares, que não são não decrescentes com os tempos de conclusão do trabalho. Como falado anteriormente, algumas medidas regulares de desempenho são: média de Tempo de Fluxo, média de antecipação, o percentual de trabalhos atrasados e com média de atraso. O critério do atraso médio, em particular, tem sido uma forma padrão para medir a conformidade com datas prometidas, embora ele ignore as consequências de trabalhos concluídos com antecipação.

No entanto, esta ênfase tem mudado com o interesse atual da filosofia Just-in-time (JIT), que defende a noção de que a antecipação, bem como o atraso, devem ser desencorajados. Em um ambiente de programação JIT, os trabalhos concluídos antecipadamente devem ser mantidos em estoque de produtos acabados até seu prazo de entrega, enquanto os trabalhos que são concluídos depois do prazo de entrega podem influenciar um cliente para que este encerre as

operações com a empresa. Portanto, uma programação ótima é aquela em que todos os trabalhos acabam exatamente em suas datas prometidas atribuídas. Isso pode ser traduzido para um objetivo de programação de muitas maneiras. O objetivo mais óbvio é o de minimizar o desvio de tempos de conclusão dos trabalhos em torno destas datas prometidas. No entanto, há outras formas que podem ser adequadas para medir o desempenho de uma programação. Obviamente, o JIT engloba um conjunto muito mais amplo de princípios do que apenas as relativas a datas prometidas, mas modelos de programação com penalidades por antecipação e atraso (*Earliness and Tardiness* – E/T) parecem captar a dimensão de programação de uma abordagem JIT.

Dessa forma, o conceito de penalização tanto da antecipação quanto do atraso gerou uma nova e rápida linha em pesquisa e desenvolvimento na área de programação. O uso de ambas as penalidades dão origem a uma medida de desempenho não regular, que levou a novas questões metodológicas na elaboração de procedimentos de solução.

5.2. O Modelo E/T

Praticamente toda a literatura sobre problemas de E/T lida com a programação estática. Em outras palavras, o conjunto de trabalhos a serem programados é conhecido de antemão e está simultaneamente disponível. A grande maioria dos artigos sobre problemas de E/T também lida com modelos em uma única máquina, apesar de que alguns resultados para uma única máquina foram estendidos para máquinas paralelas. Além disso, o critério quantitativo normalmente pode ser interpretado como a minimização do custo total de penalidade. No entanto, as penalidades podem ser medidas de diferentes formas. A maioria dos casos da relação E/T existentes na literatura decorre da generalidade das suposições feitas sobre datas prometidas e custos de penalidade.

Para descrever um modelo genérico E/T, toma-se um número de trabalhos n a ser sequenciado. O trabalho j é descrito por um tempo de processamento t_j e um prazo de entrega d_j. Conforme indicado, assume-se que os trabalhos estão disponíveis simultaneamente. Como resultado das decisões de programação, ao trabalho j será atribuído um tempo de conclusão, denotado C_j. Estejam E_j e T_j

representando a antecipação e o atraso, respectivamente, do trabalho *j*. Estas quantidades são definidas como

$$\mathrm{E_j} = \max(0, d_j - C_j) = \left(d_j - C_j\right)^+ \tag{5.3}$$

$$\mathrm{T_j} = \max(0, C_j - d_j) = \left(C_j - d_j\right)^+ \tag{5.4}$$

Associa-se a cada trabalho uma unidade de penalização por antecipação $\alpha_j > 0$ e uma unidade de penalização por atraso $\beta_j > 0$. Assumindo que as funções de penalização são lineares, a função objetivo E/T de programação de uma sequência *S* pode ser escrita como *f (S)*, onde

$$\mathrm{f(S)} = \sum_{\mathrm{j=1}}^{n} \left[\alpha_j \left(d_j - C_j\right)^+\right) + \beta_j \left(C_j - d_j\right)^+ \Big] \tag{5.5}$$

ou

$$\mathrm{f(S)} = \sum_{\mathrm{j=1}}^{n} (\alpha_j \mathrm{E_j} + \beta_j \mathrm{T_j}) \tag{5.6}$$

Em casos mais gerais, o trabalho *j* também incorre a penalidade pelo tempo de conclusão $\theta_j C_j$, o que proporciona um incentivo para mudar a ordem dos trabalhos rapidamente. O modelo contém uma compensação adicional, pois a penalidade do Tempo de Fluxo tende a induzir sequenciamento onde trabalhos com tempos de processamento menores estão nas primeiras posições da programação, enquanto o custo de antecipação induz a seqüência inversa, no início do sequenciamento. Outra penalidade dos casos mais gerais é a penalidade pela data prometida $\gamma_j d_j$, com a data prometida como variável de decisão. Em termos práticos, pode-se pensar em uma indústria que oferece uma data prometida para seu cliente durante a negociação, mas oferecem uma redução no preço caso a data prometida for definida muito tarde. Com isso, o conceito de E/T pode ser estendido para uma função objetivo mais geral

$$\mathrm{f(S)} = \sum_{\mathrm{j=1}}^{n} \left[\alpha_j \mathrm{E_j} + \beta_j \mathrm{T_j} + \theta_j C_j + \gamma_j d_j\right] \tag{5.7}$$

Em algumas formulações do problema E/T, a data prometida é conhecida, enquanto em outras o problema é otimizar a data prometida e a sequência dos trabalhos simultaneamente. Alguns dos resultados mais simples de problemas E/T

foram derivadas de modelos em que todos os trabalhos têm uma data prometida comum (ou seja, $d_j = d$). Um modelo mais geral permite diferentes datas prometidas, mas, como será visto, soluções para problemas com diferentes datas prometidas parecem estar intrinsecamente diferentes de soluções para os problemas com uma data comum. Na mesma linha, alguns modelos prescrevem penalidades comuns, enquanto outros permitem diferenças entre os trabalhos ou a diferenças entre as penalidades por antecipação e atraso.

Com tantas variações do problema E/T na literatura, pode-se perguntar quais são de importância prática. Acredita-se que cada um possua algum mérito. Tratar datas prometidas como variáveis de decisão reflete a prática em algumas oficinas em fixarem as datas de entrega internamente, como meta para orientar o andamento das atividades do chão de fábrica. Ao prescrever-se uma data prometida comum, pode-se representar uma situação onde vários itens constituem a ordem de um único cliente, ou pode refletir um ambiente de montagem em que os componentes devem estar todos prontos ao mesmo tempo, para evitar atrasos de montagem. Pode-se avaliar as penalidades, tanto para antecipação quanto para atrasos, de diferentes maneiras. Essas diversas formas de impor as penalidades para antecipação e para os atrasos sugere que os componentes dos custos associados para cada penalidade (como, por exemplo, aumentar cargas para antecipação e incentivar atraso) são claramente diferentes. Deve-se ter em mente, contudo, que o papel principal das funções de penalidade é encaminhar soluções para o objetivo de atender todas as datas prometidas exatamente no prazo.

Uma programação ideal, em que todas as datas prometidas sejam cumpridas exatamente, não é difícil de identificar. No entanto, não é tão óbvio como comparar programações não ideais. Diferentes funções de penalização podem ser vistas como sugestões para medir o desempenho de programações subótimas, quando apenas a ideal foi bem especificada.

5.3. Minimizando o Desvio Total Absoluto de uma Data Prometida Comum

Um caso de especial importância na família de problemas de E/T envolve a minimização da soma dos desvios absolutos dos tempos de conclusão do trabalho

a partir de uma data comum. Em particular, a função objetivo pode ser escrita como

$$f(S) = \sum_{j=1}^{n} \left| C_j - d \right| = \sum_{j=1}^{n} (E_j + T_j) \tag{5.8}$$

com o entendimento que $d_j = d$. Quando escreve-se a função objetivo nessa forma, fica claro que a antecipação e o atraso são penalizados a uma mesma razão para todos os trabalhos.

De forma um pouco simplificada, pode-se descrever a solução qualitativamente. Para construir a programação, é desejável que a data prometida esteja, de alguma forma, no meio da sequência dos trabalhos. Se *d* for muito apertado, então não vai ser possível ajustar os trabalho antes de *d*, devido à restrição de que nenhum trabalho pode começar antes do tempo zero. Assim, para um determinado conjunto de trabalhos onde *d* é muito apertado, é criada uma versão *restrita* do problema de sequenciamento.

É possível mostrar que existe uma solução ótima para o problema irrestrito com as seguintes propriedades

I. Não existe tempo ocioso inserido na programação. (Se o trabalho *j* começa imediatamente após o trabalho *i* na programação, então $C_j = C_i + t_j$.)

II. A programação ótima tem uma forma em V, como mostrado na Figura 14. (Trabalhos para os quais $C_j \leq d$ são sequenciados em ordem não crescente de tempo de processamento; trabalhos para os quais $C_j > d$ são sequenciados em ordem não decrescente de tempo de processamento)

III. Um trabalho é concluído precisamente na data prometida. ($C_j = d$ para algum *j*.)

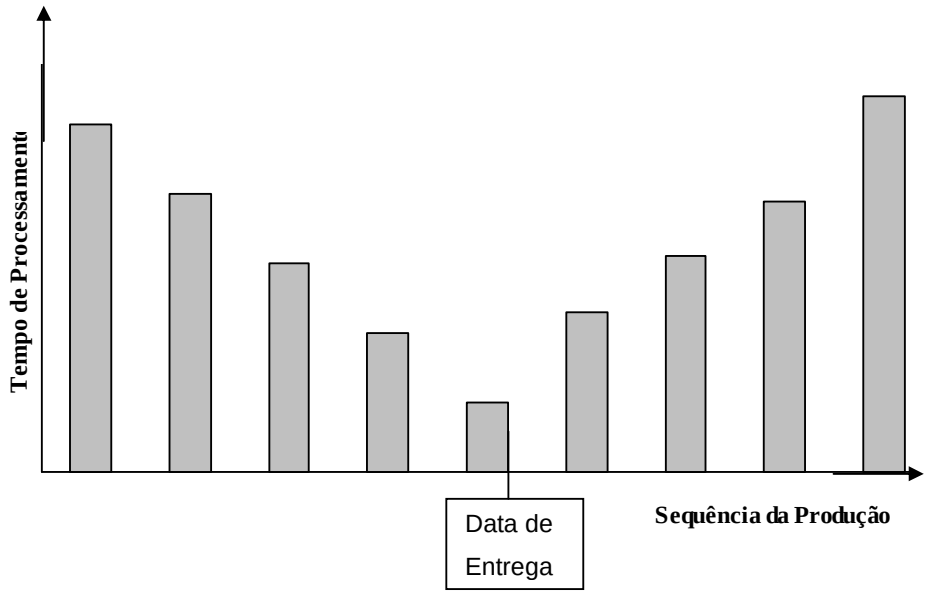

Figura 14 – Perfil "V" do sequenciamento ótimo
Fonte: Autor

A Propriedade I implica que uma sequência de trabalhos, e um tempo de início para essa sequência, são suficientes para determinar uma programação: Isso significa que a busca por um ótimo deve considerar $n!$ sequências diferentes. A Propriedade II implica que uma vez que a quantidade de itens nos dois conjuntos (J_1 e J_2) é conhecida, a sequência dos trabalhos dentro de cada conjunto pode ser determinada imediatamente: Ou seja, a busca de um ótimo deve considerar apenas as 2^n maneiras de formar os conjuntos, em vez de todas as $n!$ sequências. Mesmo sabendo a sequência de trabalho ótima, no entanto, ainda tem-se um número infinito de programações para avaliar, porque o tempo de início não está definido. A Propriedade III limita o conjunto de possibilidades para aquelas programações em que alguns tempos de conclusão dos trabalhos caem justamente na data prometida. Como consequência, cada trabalho será processado ou inteiramente antes da data prometida ou totalmente depois dele: Isso significa que uma solução pode ser particionada em dois conjuntos de trabalhos, um conjunto de trabalhos antecipados, J_1, que inclui um trabalho precisamente *on-time* e um conjunto de trabalhos atrasados, J_2. Existem, portanto, 2^n programações para serem avaliadas na busca de um ótimo. Como será apresentado, essas três propriedades podem ser generalizadas para problemas mais complexos.

A análise deste problema demonstra que existem muitas soluções ótimas. Seja um conjunto de trabalhos representado por $J_1 = B$ que termina na data prometida, ou antes, e seja $b = |B|$, a quantidade de trabalhos em B. Analogamente, seja um conjunto de trabalhos representado por $J_2 = A$ que termina após a data prometida e seja $a = |A|$. Além disso, seja B_i o índice do i-ésimo trabalho em B e A_i denota o i-ésimo trabalho em A. A penalidade por antecipação para o trabalho B_i é a soma dos tempos de processamento de todos os trabalhos em B concluídos posteriormente a B_i. Com algumas manipulações algébricas, o total das penalidades para os trabalhos em B podem ser escritos

$$C_B = 0t_{B_1} + 1t_{B_2} + ... + (b-2)t_{B_{(b-1)}} + (b-1)t_{B_b} \tag{5.9}$$

Analogamente, a penalidade total para os trabalhos em A é

$$C_A = at_{A_1} + (a-1)t_{A_2} + ... + 2t_{A_{(a-1)}} + 1t_{A_a} \tag{5.10}$$

A função objetivo é a soma de C_B e C_A, e os tempos de processamento são conhecidos. Com a e b conhecidos, essa soma seria minimizada combinando o menor coeficiente na soma com o maior tempo de processamento, o segundo menor coeficiente com o próximo maior tempo de processamento, e assim por diante, e se houver algum coeficiente com valor igual, escolhe-se o coeficiente aleatoriamente. Dessa forma, se n for par, então deve-se definir $b = a$, se n for ímpar, então $b = a + 1$. Dito de outra forma, tem-se a seguinte propriedade.

IV. Em uma programação ótima, o *b-ésimo* trabalho na sequência é concluído no tempo d, onde b é o menor inteiro maior ou igual a $n/2$.

Em outras palavras, $b = n/2$ se n é par, e $b = (n + 1)/2$ se n é impar. Mais a frente, essa propriedade será também generalizada.

Com a adequação dos coeficientes para os Tempos de Processamento em (5.9) e (5.10), constata-se que o menor coeficiente é zero e só aparece no C_B. Portanto, o trabalho mais longo é atribuído a B e, à luz da Propriedade II, aparece em primeiro lugar na sequência. O próximo menor coeficiente é *um*, aparecendo

em ambos os C_B e C_A. Isto significa que um dos próximos dois maiores trabalhos é atribuído a *A*, como seu último trabalho, e o outro para *B*, como o seu segundo trabalho. Continuando desta forma, finalmente descobre-se que o trabalho mais curto ou é o último trabalho de *B* ou o primeiro trabalho de *A*. Existem duas formas para atribuir cada par de trabalho que deve ser dividido entre os dois conjuntos *A* e *B*: se *n* é par, cria-se um trabalho fictício adicional com tempo de processamento zero para completar o último par. Assim, como observado por Bagchi, Sullivan e Chang (1986), o número total de programações ótimas é 2^r, onde

$$r = (n-1)/2, \text{ se } n \text{ é impar} \tag{5.11}$$

$$r = n/2, \text{se } n \text{ é par} \tag{5.12}$$

Na verdade, esta observação pressupõe que os Tempos de Processamento são únicos, isto é, não existem valores para os Tempos de Processamento iguais e caso isto ocorra, o número de programações ótimas é ainda maior.

Emmons (1987) considera critérios secundários como um meio de escolher entre as muitas programações ideais. Bagchi, Chang e Sullivan (1987) concentram-se especificamente sobre o critério secundário de minimizar o tempo total de processamento no conjunto *B*, que é realizado através da atribuição do menor trabalho de cada par atribuído a *B* e, se *n* for par, atribuindo o trabalho mais curto para *A*. (Define-se essa implementação como o Algoritmo 1.) Como resultado, o comprimento de B é

$$\Delta = t_n + t_{n-2} + t_{n-4} + \ldots \tag{5.13}$$

quando os trabalhos são indexados em ordem não decrescente de tempo de processamento, ou seja, t_n é o trabalho mais longo. Esta solução produz o menor valor de *d* coerente com o problema que permanece irrestrito. Em outras palavras, o problema é livre para $d \geq \Delta$ e restrito, caso contrário. Assim, se *d* era uma variável de decisão, a solução é configurar $d = \Delta$ e então utilizar o Algoritmo1.

Nota-se que o Algoritmo 1 é polinomial e pode ser implementado para ser executado em tempo O (n log n).

A versão restrita do problema ocorre quando $d < \Delta$. Nesse caso, as propriedades I e II ainda são mantidas (Raghavachari (1986) prova essa última propriedade.). Todavia, a propriedade III não se sustenta: A solução ótima pode conter um trabalho abrangente, que começa antes de d e termina depois de d. Hall, Kubiak e Sethi (1989) demonstraram que a versão restrita do problema é NP-completo.

Bagchi, Sullivan e Chang (1986) descrevem um algoritmo para resolver a versão restrita. No entanto, o respectivo procedimento pressupõe implicitamente que o tempo de início da programação é zero. Szwarc (1989) aponta que o horário de início ótimo pode ser diferente de zero, de modo que o algoritmo de Bagchi, Sullivan e Chang não garantem otimalidade. Szwarc passa a dar uma condição suficiente para a programação ótima para iniciar no tempo zero

$$t_1 + t_2 + ... + t_r \geq \mathrm{d} \tag{5.14}$$

com r como foi definido em (5.11) e (5.12). Se esta condição é verdadeira, então pode-se assumir zero como hora de início. Mesmo assim, o problema pode não ser fácil de resolver. (Os procedimentos de solução propostos por Szwarc, e Bagchi, Sullivan e Chang são de carácter enumerativos.) Sundararaghavan e Ahmed (1984) apresentam um procedimento de solução heurística que parece funcionar efetivamente quando o horário de início é zero; Baker e Chadowitz (1989) relaxam esta condição e generalizam o algoritmo.

Como a essência do problema E/T reside na sua medida de desempenho não regular, os algoritmos projetados para variações em que o horário de início é obrigado a ser igual a zero acabam sendo de interesse limitado. Acredita-se que quando um critério E/T é utilizado, uma solução deve ser procurada em um domínio que permita o início da programação com atraso. A imposição de uma obrigação de hora de início igual a zero diminui a importância da função objetivo. Em termos práticos: não se pode imaginar que um programador imponha uma restrição arbitrária que aumente o custo de uma programação.

Em resumo, o problema sem restrições pode ser resolvido pelo Algoritmo1, que classifica os trabalhos e atribui a esses trabalhos uma posição na sequência.

Por outro lado, o problema restrito não foi resolvido, em geral, por qualquer outro método do que um algoritmo enumerativo. No pior dos casos, esta aproximação requer uma comparação de 2^n programações.

5.4. Penalidades Diferentes para Antecipação e Atraso

Uma generalização do modelo básico deriva-se da noção de que a antecipação e o atraso devem ser penalizados com taxas diferentes. Como observado anteriormente, α pode representar um custo de manutenção do trabalho na linha de produção, enquanto β representa uma penalidade por atraso. Estas taxas são provavelmente diferentes, especialmente porque α tende a ser endógeno, enquanto β tende a ser exógeno. Em particular, seja

$$f(S) = \sum_{j=1}^{n} (\alpha E_j + \beta T_j) \tag{5.15}$$

Esse problema, envolvendo a soma ponderada dos desvios absolutos, foi analisado por Bagchi, Chang e Sullivan (1987) e resumida por Emmons (1987). Novamente, há uma versão restrita do problema, bem como sua versão sem restrições, assim como definido em (5.13). Na versão sem restrições, uma solução ótima tem as seguintes propriedades:

I. Não há nenhum tempo ocioso inserido.
II. A programação ideal é em forma de V.
III. Um trabalho é concluído no tempo *d*.

Estes resultados são novamente provados simplesmente por contradição. Em seguida, os componentes da função objetivo, analogamente como em (5.9) e (5.10), são a penalização total para *B*

$$C_B = 0t_{B_1} + \alpha t_{B_2} + ... + (b-2)\alpha t_{B_{(b-1)}} + (b-1)\alpha t_{B_B} \tag{5.16}$$

E a penalidade total para *A*

$$C_A = a\beta t_{A_1} + (a-1)\beta t_{A_2} + ... + 2\beta t_{A_{(a-1)}} + 1\beta t_{A_a} \tag{5.17}$$

A solução pode novamente ser determinada por uma correspondência entre coeficientes e os tempos de processamento. A generalização da propriedade IV é

IV. Em uma programação ótima, o *b-ésimo* trabalho na sequência é concluído no tempo d, onde b é o menor inteiro maior ou igual a $n\beta/(\alpha + \beta)$.

Note que quando $\alpha = \beta$, esta afirmação da propriedade se reduz à propriedade IV dada anteriormente.

Bagchi, Chang e Sullivan (1987) apresentam uma alternativa ao processo de correspondência, que se define como Algoritmo 2. Ele começa com os conjuntos A e B vazios, e atribui os trabalhos em cada conjunto utilizando a seguinte regra de decisão:

- Considere os trabalhos em ordem não crescente de tempo de processamento.
- Se $\alpha \mid B \mid < \beta (1 + \mid A \mid)$, então atribua o próximo trabalho ao conjunto A, caso contrário, o atribua ao conjunto B.
- Continue até que todos os trabalhos estejam em sequência.

Dependendo dos parâmetros do problema, pode novamente existir uma alternativa ótima. Se essa alternativa existir, então o algoritmo de correspondência pode ser implementado de uma maneira que se assegure que o trabalho em B é o mais curto possível. (Este recurso está implícito no Algoritmo 2). Uma vez que isso seja feito, tem-se

$$\Delta = \sum_{i=1}^{b} t_{Bi} \tag{5.18}$$

Como antes, o problema é dito sem restrições para $d \geq \Delta$. Note que em um problema mais simples, com $\alpha = \beta$, é possível calcular Δ em (5.13) diretamente pelos parâmetros dados. No caso de penalidades por antecipação e atraso

diferentes, no entanto, é necessário resolver a versão irrestrita do problema antes de utilizar (5.18).

Para a versão restrita do problema, que é um NP-completo, Bagchi, Chang e Sullivan (1987) mostram que as Propriedades I e II são mantidas. No entanto, nenhum procedimento de otimização foi desenvolvido para versões restritas, exceto quando assume-se que os tempos de início são zero. Baker and Chadowitz (1989) apresentam uma aproximação heurística da solução para essa versão.

Considere, a seguir, o mesmo modelo com a data prometida como uma variável de decisão. A solução ótima é definir $d = \Delta$ a partir de (5.18) e, em seguida, usar a solução do problema sem restrições. Este resultado é originalmente feito por Panwalkar, Smith e Seidmann (1982), que incorporam esse resultado na análise de um caso mais geral.

5.5. Penalidades por Antecipação e Atraso Dependentes do Trabalho

Uma direção óbvia para potencial generalização do modelo é permitir que cada trabalho tenha a sua própria penalização α_j e β_j.

Especificamente, a função objetivo assume a forma

$$f(S) = \sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j) \qquad (5.19)$$

Quando $\alpha_j = \beta_j$, a penalidade por atraso corresponde à penalidade de antecipação para qualquer trabalho em particular, mas as penalidades podem variar entre os trabalhos. A versão sem restrições deste problema foi examinada por Bagchi (1985), Cheng (1987), Quaddus (1987), Bector, Gupta e Gupta (1988) e Hall e Posner (1989). Cheng apresenta uma variação da Propriedade IV para o problema básico. Bagchi considera o caso em que $\alpha_j = \alpha.t_j$. (Essa proporcionalidade modela uma situação onde o custo unitário da antecipação reflete o custo de valor agregado, que tende a ser proporcional ao tempo de processamento.) Ele prova algumas propriedades de dominância que podem acelerar um procedimento de solução. Bector, Gupta e Gupta apresentam uma perspectiva de programação linear sobre esses mesmos resultados. Hall e Posner

provam as variações relevantes das propriedades I e II e também algumas propriedades de dominância que proporcionem condições necessárias para uma sequência ótima. Seu resultado mais significativo é a prova de que a versão sem restrições do problema é NP-completo. Eles desenvolveram um algoritmo de programação dinâmica, onde mostram ser pseudopolinomial. Além disso, eles demonstram a eficácia de seu algoritmo computacional, atacando os problemas que contêm centenas de trabalhos e obtendo soluções ótimas com pequenos tempos de execução.

Quaddus (1987) considera o caso geral em que $\alpha_j \neq \beta_j$ (e inclui também uma penalidade data prometida, γ_j), mas lida apenas com a seleção de uma data prometida. Baker e Scudder (1989) assinalam que Quaddus negligencia os aspectos de sequenciamento do problema. No entanto, é fácil mostrar que a Propriedade I ainda é mantida e a Propriedade II toma a forma

II. A programação ótima é em forma de V: os trabalhos em *B* estão sequenciados em ordem não crescente na razão t_j/α_j; os trabalhos em *A* são sequenciados em ordem não decrescente do razão na t_j/β_j.

Nota-se que quando $\alpha_j = \alpha$ e $\beta_j = \beta$, esta afirmação da propriedade é similar à propriedade II apresentada anteriormente.

Além disso, a Propriedade III é mantida e a forma geral da Propriedade IV especifica uma condição necessária para *b* como

IV. Em uma programação ótima, o *b-ésimo* trabalho na sequência é concluído no tempo *d*, onde *b* é o menor inteiro que satisfaça a inequação

$$\sum_{j=1}^{b}(\alpha_j + \beta_j) \geq \sum_{j=1}^{n}(\beta_j - \gamma_j) \tag{5.20}$$

Onde *j* denota o *j-ésimo* trabalho na sequência. (Essa expressão simplificada se reduz às formas da Propriedade IV dada anteriormente para casos especiais). No entanto, a busca por uma programação ótima deve enumerar e comparar as

diferentes programações em forma de V que são consistentes com a Propriedade IV.

A versão com restrição desses problemas não foi resolvida.

5.6. Datas Prometidas Distintas

O modelo E / T geralmente tem diferentes datas prometidas para o conjunto de trabalhos analisado. Esta característica tende a tornar mais difícil em determinar o custo mínimo de uma programação do que nos problemas discutidos até agora. No entanto, se as datas prometidas forem tratadas como variáveis de decisão, o problema acaba por ser relativamente simples.

O modelo desenvolvido nesta dissertação assume que as datas prometidas são dadas e distintas, e a função objetivo é a mesma que em (5.19).

Garey, Tarjan e Wilfong (1988) foram os primeiros a mostrar que este problema é NP-completo. Neste modelo, as propriedades I e II não se aplicam: A seqüência ideal pode não ser em forma de V e a inserção de tempo ocioso pode ser desejável. A busca por uma programação ótima pode, no entanto, ser decomposto em dois subproblemas: encontrar uma seqüência ótima de trabalhos e programar a inserção de tempo ocioso.

Abdul-Razaq e Potts (1988) resolvem este tipo de problema, com uma penalidade para a data prometida incluída, mas eles consideram somente as programações sem tempo ocioso. O seu método de solução é um esquema de branch-and-bound, e eles usam um processo de programação com relaxação dinâmica para obter bons limites. Seus resultados computacionais sugerem que os problemas com mais de 20 trabalhos podem levar a tempos de solução excessivos.

Embora existam outros trabalhos que não levam em consideração os tempos ociosos, o modelo que permita a utilização desses tempo é mais realista para a minimização de F(S).

Dada uma seqüência de trabalho, a sequência ótima para os tempos ociosos pode ser resolvida através da resolução de um problema de programação linear. No entanto, Garey et. all. (1988) fornecem detalhes sobre um procedimento mais simples que pode ser implementado para ser executado em tempo O (n log n).

Dado que o tempo ocioso pode ser facilmente otimizado para uma seqüência de trabalho específica, a tarefa restante é para alocar a melhor seqüência. Fry, Armstrong e Blackstone (1987) descrevem um procedimento de solução que se baseia na busca de troca de pares adjacentes dentro de uma vizinhança até que um ótimo local é descoberto. Os testes computacionais indicam que esta heurística produz soluções que variam em média de 2% da solução ótima. Fry, Darby-Dowman e Armstrong (1988) descrevem um procedimento branch-and-bound e dão resultados computacionais em detalhe. Eles indicam que o algoritmo é executado com dificuldade para problemas maiores do que n = 20. Yano e Kim (1986) lidam com o caso especial em que $\beta_j = \alpha_j = 1$. Eles resolvem problemas de até 30 trabalhos usando um esquema de branch-and-bound e relatam que uma heurística de troca de pares de trabalhos freqüentemente encontra uma solução ótima.