USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

Unha revisión de problemas de horarios e aplicacións

Sara Brea González

2024/2025

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Unha revisión de problemas de horarios e aplicacións

Sara Brea González

Xullo, 2025

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Traballo proposto

Área de Coñecemento: Estatística e Investigación Operativa
Título: Unha revisión de problemas de horarios e aplicacións
Breve descrición do contido
No ámbito da xestión de operacións e no campo da optimización combinatoria, os problemas de horarios supoñen un desafío fundamental e son de vital importancia. Este tipo de problemas tratan de asignar recursos limitados (como tempo, máquinas ou persoal) a un conxunto de tarefas de forma eficiente, atendendo a un certo criterio de optimalidade, e considerando certas restricións temporais adicionais. A súa relevancia é notable nunha gran variedade de problemas que xorden en contextos industriais, incluíndo procesos de manufactura, transporte, servizo e tecnoloxía, entre outros. O traballo comeza cunha introdución da notación destes problemas, seguido do estudo de propiedades e algoritmos de resolución de distintos tipos de problemas de horarios e as súas aplicacións na realidade.
Recomendacións
Outras observacións

Índice

Resumo

Os problemas de horarios consisten na asignación eficiente de recursos limitados a un conxunto de tarefas, tendo en conta determinadas restricións temporais e criterios de optimalidade. No primeiro capítulo comezamos detallando a notación que empregaremos ao longo do traballo. A continuación, no segundo e terceiro capítulo faremos unha revisión de diferentes versións nas que se poden presentar ditos problemas, presentando en cada caso o algoritmo correspondente para atopar unha solución óptima, xunto con certas propiedades teóricas. No cuarto capítulo inclúese a implementación dos algoritmos mediante código de R. Finalmente, no quinto capítulo recóllense as conclusións, onde se destaca a relevancia destes problemas tanto en contextos da vida cotiá como no ámbito profesional.

Abstract

Scheduling problems involve the efficient allocation of limited resources to a set of tasks, considering specific time constraints and optimality criteria. In the first chapter, notation used across the paper will be detailed. The second and third chapters review different variants of the problem, presenting in each case the corresponding algorithm to find the optimal solution, along with the discussion of certain associated theoretical properties. The fourth chapter includes the implementation of these algorithms in R code. Finally, in the fifth chapter, conclusions will be drawn, highlighting the relevance of these problems both in everyday live as well as professional contexts.

Introdución

No mundo no que vivimos hoxe en día, unha preocupación común para a maioría de persoas é o tempo, ese compañeiro de vida que pase o que pase non se detén e avanza a pasos axigantados. Trátase dun ben insustituible que a sociedade actual busca exprimir ao completo sen deixar pasar un só segundo, buscando aproveitar e darlle un sentido a cada instante.

Como consecuencia disto, organizar o tempo de forma eficiente converteuse nunha necesidade fundamental en diversos ámbitos da vida cotiá e profesional. No ámbito da xestión de operacións e no campo da optimización combinatoria, os problemas de horarios (coñecidos como scheduling problems, en inglés) supoñen un desafío fundamental e son de vital importancia. Os problemas de horarios consisten en asignar recursos limitados a un conxunto de tarefas de forma eficiente coa finalidade de optimizar un ou máis obxectivos. O resultado final destes problemas dará lugar a un horario óptimo que deberá cumprir certas restricións temporais de diferentes tipos.

Un horario consiste nun plan tanxible ou documento que aporta información sobre cando deben suceder certas cousas, como o horario dos buses ou un horario escolar. A pregunta "cando?", como se plantexa en [1], pode ser respondida con información temporal ou con termos de secuencia da información que recolle o horario.

Os problemas de horarios son de vital importancia en contextos industriais, incluíndo procesos de manufactura, transporte, servizos e tecnoloxía, entre outros. Dependendo da situación á que teñan que ser aplicados, os recursos a asignar poden ser tanto máquinas, como pistas dun aeroporto ou unidades de procesamento nun entorno informático. Pola súa banda, as tarefas poden ser operacións nun proceso de produción, despegues e aterrizaxes de diferentes avións e execucións dun certo programa no ordenador, como algúns exemplos. Os obxectivos a cumprir tamén poden ser igual de variados, como minimizar o tempo de completado da tarefa final ou minimizar o número de tarefas con retraso.

Neste traballo centrámonos no estudo de problemas de horarios deterministas cunha única máquina. Na literatura, como xa comentamos, tamén se definen outros tipos de problemas de horarios con máis complexidade, xa sexa con varias máquinas, con máquinas en paralelo ou con incerteza nos diferentes parámetros do problema. Apoiarémonos para realizar dito estudo en [1],

[2], [3], [4] e [5], mais empregando como referencia principal [4].

No Capítulo 1 comezaremos cunha breve introdución dos elementos e parámetros que poden aparecer nestes problemas. Logo, faremos un repaso por distintos obxectivos a optimizar. No Capítulo 2, falaremos do tempo total de finalización ponderado e presentaremos dous algoritmos para dúas variantes desta casuística. No Capítulo 3 introduciremos as chamadas funcións de custo, dando lugar a novos obxectivos que permiten modelar a realidade de mellor maneira. Neste capítulo temos tres seccións que se corresponden con tres tipos de función de custo, para as que presentaremos o seu correspondente algoritmo. No Capítulo 4 recóllese o código en R das implementacións dos algoritmos estudados, seguido do Capítulo 5, que dá peche ao traballo recollendo as conclusións finais de todo o estudo.

Para aplicar o marco teórico de forma práctica, en cada sección atoparemos un exemplo cun contexto real no que se ilustrará o estudado nese apartado. Todos os exemplos representan unha situación dentro do servizo sanitario, un contexto no que os problemas de horarios teñen gran importancia. Dado que moitas veces neste ámbito os recursos son limitados, e as tarefas, neste caso os pacientes, son pola contra, moi abundantes, é imprescindible buscar a mellor forma de asignar ao maior número de tarefas, é dicir, o que sería tratar ao maior número de persoas, tendo en conta as condicións e estado de cada un.

Xa falábamos ao comezo de que o tempo era un ben moi valioso, e se o é para a maioría de persoas, canto máis o será para alguén que teña que estar agardando nas salas de espera desas consultas? Damos paso a un estudo sobre diferentes problemas de horarios que poden servir de axuda para as situacións que vimos de tratar.

Capítulo 1

Introdución aos problemas de horarios

Os **problemas de horarios** xestionan e organizan a execución dun conxunto de tarefas a realizar nunha ou en varias máquinas. Neste traballo só imos considerar problemas de horarios deterministas cunha única máquina. O propósito dos problemas de horarios nos que nos centraremos será asociar o horario correspondente a cada tarefa tendo en conta unha serie de restricións como ordes de preferencia, ventás de tempo, o límite de recursos dos que dispoñemos, etc. Ademais, como veremos, á hora de deseñar dita planificación pódense considerar distintos obxectivos a minimizar como por exemplo, minimizar o tempo máximo no que se realizan todas as tarefas.

Comezamos cunha breve introdución da notación que empregaremos ao longo do traballo. Nos problemas de horarios (coñecidos como *scheduling problems* en inglés) teranse en conta o número de tarefas a realizar, denotadas por $\boldsymbol{j}$, sendo $\boldsymbol{n}$ o número total. Dispoñemos da seguinte información asociada a cada tarefa:

p_j denota o tempo de procesamento de dita tarefa.

r_j data de inicio de j.

d_j data límite para completar a tarefa. En caso de que este data sexa de obrigado cumprimento denotarase por $\bar{d}_j$.

w_j identifica a importancia de j.

Os problemas de horarios represéntanse mediante a terna $\boldsymbol{\alpha} \mid \boldsymbol{\beta} \mid \boldsymbol{\gamma}$, podendo omitir algunha delas no caso de non ser necesaria. O parámetro $\boldsymbol{\alpha}$ representa o entorno asociado ás máquinas. Existen diferentes posibilidades como (1), se temos unha única máquina; P_m, temos m máquinas idénticas en paralelo; Q_m, contamos con m máquinas en paralelo con diferentes velocidades... Este parámetro non é prescindible e só presenta unha única entrada, pois as distintas opcións posibles para as máquinas non son compatibles. No caso dos problemas que imos estudar, α

tomará o valor (1), describindo o que chamaremos *Single Machine Models*.

Por outro lado, $\boldsymbol{\beta}$ indica os detalles do proceso. Este parámetro pode quedar baleiro, tomar un valor ou varios, algúns exemplos son:

- r_j, xa definido anteriormente, de non aparecer, a tarefa j pode comezar en calquer momento.
- *prmp*, permítese interrumpir unha tarefa con outra diferente antes de que a primeira remate, de non aparecer este valor, non sería posible a interrupción.
- *prec*, require que un traballo remate antes de comezar co seguinte.

Por último, o parámetro $\boldsymbol{\gamma}$ determina o obxectivo a minimizar, como xa mencionamos ao inicio desta sección. Este obxectivo sempre será unha función do tempo de finalización da tarefa j, o cal denotaremos por $\boldsymbol{C_j}$. Este parámetro depende do horario asignado a cada tarefa (como se vai procesando cada tarefa e a que hora). Estas funcións denomínanse medidas regulares e caracterízanse por ser non decrecentes en $C_1, .., C_n$. Nos seguintes capítulos presentaremos distintas funcións obxectivo para este parámetro mais non poderá presentar máis dunha á vez.

Para referirnos a un certo horario concreto denotarémolo por $\mathcal{S}$, e en caso de ter que manexar distintos horarios nun mesmo contexto denotaremos os seguintes por $\mathcal{S}'$, engadindo as $'$ que sexan necesarias segundo o número de horarios que teñamos.

Como xa dixemos, vamos centrar o noso estudo nos problemas cunha única máquina. Os resultados obtidos para estes modelos son aplicables para situacións máis complexas como, por exemplo, un entorno de traballo con varias máquinas situadas en paralelo. Para comprender mellor os termos introducidos ata o de agora, vexámolos no seguinte exemplo.

Exemplo 1.1. Problema de horarios para o tratamento de radioterapia nunha única máquina. Nun hospital contan cunha única máquina de radioterapia para tratar aos pacientes con cancro. Cada paciente require dunha sesión de tratamento cun tempo específico de duración e cada sesión ten unha franxa horaria máis favorable debido a razóns médicas ou dispoñibilidade do paciente. O obxectivo é minimizar o retraso total na administración dos tratamentos. En adición, débese cumprir que un traballo remate antes de comezar co seguinte (para o parámetro β non teremos o valor *prmp*). A modo de exemplo temos no Cadro 1.1 información relativa a 4 pacientes.

Neste contexto cada paciente denota unha tarefa j, a tarefa sería darlle o seu tratamento, tendo $n = 4$ como número total. Os tempos de procesamento corresponden á segunda columna, duración do tratamento de cada paciente, é dicir, o tempo durante o cal a máquina de radioterapia estará ocupada: $p_1 = 30$, $p_2 = 20$, $p_3 = 40$, $p_4 = 25$. Por outra banda, a columna do horario desexado indica o momento límite no que o tratamento de cada paciente debe estar rematado. Se

consideramos que os tratamentos comezan a partir das 9:00 da mañá, estes valores representan os minutos transcorridos dende ese momento: $d_1 = 60$, $d_2 = 150$, $d_3 = 120$, $d_4 = 90$.

Paciente	Tempo de tratamento (min)	Horario desexado (minuto do día)
1	30	60
2	20	110
3	40	100
4	25	90

Cadro 1.1: Datos dos pacientes

O retraso dun tratamento calcúlase como $L_j = C_j - d_j$, sendo C_j o momento do día no que finaliza o tratamento do paciente j. Para o parámetro γ, a función a minimizar será $\sum_{j=1}^{n} L_j$. En canto aos detalles do proceso sabemos que un traballo debe rematar antes de que comece o seguinte, é dicir, non pode haber interrupcións. Logo, para o parámetro β temos o valor *prec*. En síntese, este problema podemos describilo coa terna $1|\ prec\ |\sum_{j=1}^{n} L_j$.

Unha posible solución podería ser a secuencia de pacientes do Cadro 1.2. Observamos que o retraso total na administración dos tratamentos nese caso será de tan só 5 minutos.

Orde dos tratamentos	1	4	3	2
p_j	30	25	40	20
C_j	30	55	95	115
d_j	60	90	100	110
L_j	0	0	0	5

Cadro 1.2: Solución posible

Merece a pena mencionar que o problema está plantexado de maneira moi simplificada. Na realidade a xestión dos tratamentos de radioterapia é moito máis complexa entrando en xogo moitos outros factores e variables. Este exemplo serviranos de apoio para a explicación das seguintes seccións modificándoo segundo sexa necesario.

Capítulo 2

Tempo total de finalización ponderado

O primeiro tipo de problema que trataremos ten como obxectivo (parámetro γ), **minimizar o tempo total de finalización ponderado**, representado pola función $\sum_j w_j C_j$. Comezaremos introducindo uns resultados teóricos previos á presentación do primeiro algoritmo para resolver os problemas de horarios neste contexto. No primeiro teorema recóllese que a regra $Weighted\ Shortest\ Processing\ Time\ first$ ($WSPT$), mediante a que se ordenan as tarefas a realizar por orde decrecente do cociente w_j/p_j, é a mellor forma de afrontar os problemas desta sección. A continuación, presentaremos dous lemas moi útiles á hora de asignar o horario ao aplicar o algoritmo. De seguido, empregaremos unha modificación do exemplo do capítulo anterior para visualizar de maneira aplicada o algoritmo do que imos a falar. Na segunda e última sección, introduciremos as datas de inicio como parámetro para β, ademais de permitir a interrupción entre tarefas. Estudaremos se a regra $WSPT$ segue a ser óptima para estes novos problemas empregando para iso un exemplo para ver claramente os resultados.

Teorema 2.1. *A regra WSPT é óptima para o problema* $1//\sum_j w_j C_j$.

Demostración. Vexámolo por redución ao absurdo. Supoñamos que existe un horario $\mathcal{S}$ óptimo que non segue a regra $WSPT$ de maneira que sexa óptimo. Nel temos dúas tarefas consecutivas, a j e logo a k, verificando

$$\frac{w_j}{p_j} < \frac{w_k}{p_k}$$

é dicir, non seguen unha orde decrecente do cociente w_j/p_j, polo que non se cumpre a regra $WSPT$. Asumimos que o tempo de inicio da tarefa j é t, e unha vez rematada comeza k.

Realizamos un intercambio de tarefas e definimos un novo schedule chamado $\mathcal{S}'$. Agora o traballo k dará comezo no tempo t e a continuación realízase j. O resto de tarefas mantéñense nas mesmas posicións. O tempo total de finalización ponderado dos traballos anteriores a j e a

k non se ve afectado polo intercambio, mais si o fará o dos posteriores. Esta modificación está determinada polos traballos j e k.

Baixo $\mathcal{S}$ o tempo total de finalización ponderado dos traballos j e k é

$$(t + p_j)w_j + (t + p_j + p_k)w_k \tag{2.1}$$

e baixo $\mathcal{S}'$ será

$$(t + p_k)w_k + (t + p_k + p_j)w_j \tag{2.2}$$

Desenvolvendo as expresións anteriores obtemos para (2.1) e (2.2)

$$\underline{tw_j} + \underline{p_j w_j} + \underline{tw_k} + p_j w_k + \underline{p_k w_k} \qquad \underline{tw_k} + \underline{p_k w_k} + \underline{tw_j} + p_k w_j + \underline{p_j w_j},$$

respectivamente. Comparando ambas ecuacións observamos que os termos subliñados están en ambas expresións.

Se $w_j/p_j < w_k/p_k$, é dicir, $w_j p_k < w_k p_j$, obtemos que o tempo total de finalización ponderado deses traballos baixo $\mathcal{S}'$ é estrictamente menor que o correspondente a $\mathcal{S}$, chegando así a unha contradicción co feito de que $\mathcal{S}$ sexa óptimo. Esta demostración é unha ampliación da recollida en [4]. □

2.1. Restricións de precedencia

Agora vexamos como afecta no tempo de finalización a introdución de restricións de precedencia para as tarefas. Definimos o novo problema como $1|\ prec\ |\ \sum_j w_j C_j$, onde o parámetro β toma o valor *prec*, o que indica que un traballo ten que estar rematado antes de comezar co seguinte.

Consideraremos unha das situacións máis simples. Tomamos dúas cadeas de traballos paralelas: a cadea I está formada polas tarefas 1,...,k e a cadea II polas restantes, k+1,...,n. Como apoio para o algoritmo que resolve o problema de mininización, presentamos os seguintes resultados.

Lema 2.2. *Se*

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > (<) \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$$

logo, é óptimo comezar co procesamento da cadea I antes (despois) que coa cadea II.

Demostración. Vexámolo por redución ao absurdo.

Para a secuencia 1,..., k, k+1,..., n, o tempo total de finalización ponderado é

$$w_1 p_1 + ... + w_k \sum_{j=1}^{k} p_j + w_{k+1} \sum_{j=1}^{k+1} p_j + ... + w_n \sum_{j=1}^{n} p_j \tag{2.3}$$

mentras que para a secuencia k+1,..., n, 1,..., k, será

$$w_{k+1}p_{k+1} + ... + w_n \sum_{j=k+1}^{n} p_j + w_1(\sum_{j=k+1}^{n} p_j + p_1) + ... + w_k \sum_{j=1}^{n} p_j \tag{2.4}$$

Desenvolvemos a expresión (2.3)

$$w_1p_1 + w_2\sum_{j=1}^{2} p_j + w_3\sum_{j=1}^{3} p_j + ... + w_k\sum_{j=1}^{k} p_j +$$
$$(w_{k+1}\sum_{j=1}^{k} p_j + w_{k+1}p_{k+1}) + (w_{k+2}\sum_{j=1}^{k} p_j + w_{k+2}\sum_{j=k+1}^{k+2} p_j) + ... + (w_n\sum_{j=1}^{k} p_j + w_n\sum_{j=k+1}^{n} p_j)$$

Facemos o mesmo coa expresión (2.4)

$$w_{k+1}p_{k+1} + w_{k+2}\sum_{j=k+1}^{k+2} p_j + ... + w_n\sum_{j=k+1}^{n} p_j +$$
$$(w_1\sum_{j=k+1}^{n} p_j + w_1p_1) + (w_2\sum_{j=k+1}^{n} p_j + w_2\sum_{j=1}^{2} p_j) + (w_3\sum_{j=k+1}^{n} p_j + w_3\sum_{j=1}^{3} p_j) + ... + (w_k\sum_{j=k+1}^{n} p_j + w_k\sum_{j=1}^{k} p_j)$$

É evidente que existen varios termos que se repiten nas dúas expresión. Comparándoas, obtemos para (2.3) e (2.4)

$$w_{k+1}\sum_{j=1}^{k} p_j + w_{k+2}\sum_{j=1}^{k} p_j + ... + w_n\sum_{j=1}^{k} p_j = \sum_{j=k+1}^{n} w_j\sum_{j=1}^{k} p_j$$
$$w_1\sum_{j=k+1}^{n} p_j + w_2\sum_{j=k+1}^{n} p_j + w_3\sum_{j=k+1}^{n} p_j + ... + w_k\sum_{j=k+1}^{n} p_j = \sum_{j=1}^{k} w_j\sum_{j=k+1}^{n} p_j$$

se ademais temos en conta que

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$$

obtemos

$$\sum_{j=1}^{k} w_j \sum_{j=k+1}^{n} p_j > \sum_{j=k+1}^{n} w_j \sum_{j=1}^{k} p_j$$

polo que concluimos que o proceso será óptimo se comezamos coa cadea I. Esta demostración é unha ampliación da recollida en [4]. □

Por outra banda, existe a posibilidade de que, habendo dúas cadeas, non sexa necesario rematar todas as tarefas da primeira antes de comezar coa segunda. Antes de presentar o seguinte resultado, introducimos a seguinte definición.

Definición 2.3. Consideramos a cadea 1,..,k e sexa $\boldsymbol{l^*}$ o elemento que satisfai a seguinte igualdade

$$\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} = \max_{1\leq l\leq k} (\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j})$$

O termo da esquerda denótase por $\rho(1,...,k) = \frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j}$ e defínese como o **$\boldsymbol{\rho}$-factor** da cadea 1,...,k. Por conseguinte, definimos $\boldsymbol{l^*}$ como o traballo que determina o ρ-factor da cadea 1,...,k.

Lema 2.4. *Se l^* determina $\rho(1,...,k)$, existe unha secuencia óptima que procesa de seguido as tarefas 1,...,l^*, é dicir, sen que existan interrupcións de tarefas doutras cadeas entre elas.*

Demostración. Procedamos á demostración do resultado por redución ao absurdo. Supoñamos que baixo a secuencia óptima de procesamento, a subsecuencia 1,...,l^* está interrompida pola tarefa v pertencente a outra cadea diferente. Desta forma, a secuencia óptima contén á subsecuencia $\mathcal{S}$ definida como:

$$\mathcal{S} \quad \rightarrow 1,...,u,v,(u+1),...,l^*$$

Para completar o resultado vexamos que o tempo total de procesado será menor coa subsecuencia

$$\mathcal{S}' \rightarrow v,1,...,l^* \quad \text{ou} \quad \mathcal{S}'' \rightarrow 1,...,l^*,v$$

con respecto á subsecuencia $\mathcal{S}$, é dicir, situando a tarefa v ao inicio ou ao final da subsecuencia, sen interromper á considerada inicialmente.

Buscamos aplicar o Lema 2.2 para as subsecuencias $\mathcal{S}$ e $\mathcal{S}'$. Seguindo a terminoloxía da demostración anterior, tomamos:

$$\begin{aligned} \text{Cadea I} &\rightarrow 1,\dots,u \\ \text{Cadea II} &\rightarrow v \end{aligned}$$

O enunciado do Lema 2.2 quedaría do seguinte xeito:

Se

$$(\frac{w_1+w_2+...+w_u}{p_1+p_2+...+p_u}) = \frac{\sum_{j=1}^{u} w_j}{\sum_{j=1}^{u} p_j} > \frac{w_v}{p_v} \tag{2.5}$$

logo é óptimo comezar co procesamento da cadea I antes que coa cadea II (é dicir, $\mathcal{S}$ terá menor tempo total de procesado que $\mathcal{S}'$).

Aplicamos de novo o Lema 2.2 para as subsecuencias $\mathcal{S}$ e $\mathcal{S}''$ tomando:

$$\begin{aligned} \text{Cadea I}' &\rightarrow v \\ \text{Cadea II}' &\rightarrow u+1,\dots,l^* \end{aligned}$$

Reescribimos o enunciado do Lema 2.2 quedando da seguinte forma:

Se

$$\frac{w_v}{p_v} > \frac{\sum_{j=u+1}^{l^*} w_j}{\sum_{j=u+1}^{l^*} p_j} (= \frac{w_{u+1} + w_{u+2} + ... + w_{l^*}}{p_{u+1} + p_{u+2} + ... + p_{l^*}}) \tag{2.6}$$

logo é óptimo comezar co procesamento da cadea I′ antes que coa cadea II′ (é dicir, $\mathcal{S}$ terá menor tempo total de procesado que $\mathcal{S}''$).

Se a tarefa l^* é a que determina o ρ-factor da cadea $1, ..., k$ temos o seguinte:

$$\frac{w_{u+1} + w_{u+2} + ... + w_{l^*}}{p_{u+1} + p_{u+2} + ... + p_{l^*}} > \frac{w_1 + w_2 + ... + w_u}{p_1 + p_2 + ... + p_u} \tag{2.7}$$

Chegamos a esta desigualdade pois, por definición, l^* é o traballo que determina o ρ-factor. Tendo isto en conta, a secuencia $u+1, .., l^*$ debe ter un cociente maior que o da secuencia $1, ..., u$, xa que esta última non contén ao punto que maximiza dito cociente. Se $1, ..., u$ tivese un cociente maior, o máximo non estaría en l^*, contradecindo así a definición de l^*.

Entón se $\mathcal{S}$ é mellor que $\mathcal{S}''$, tendo en conta (2.6) e logo (2.7) chegamos ao seguinte:

$$\frac{w_v}{p_v} > \frac{w_{u+1} + w_{u+2} + ... + w_{l^*}}{p_{u+1} + p_{u+2} + ... + p_{l^*}} > \frac{w_1 + w_2 + ... + w_u}{p_1 + p_2 + ... + p_u}$$

Obtendo así unha contradición coa desigualdade (2.5), o que nos leva a que $\mathcal{S}'$ é mellor que $\mathcal{S}$. Esta demostración é unha ampliación da recollida en [4].

□

Estamos en condicións de presentar o seguinte algoritmo. No contexto no que se consideran dúas cadeas, resultará moi sinxelo de aplicar.

Algoritmo 2.5. ***Tempo total de finalización ponderado e cadeas.*** *No momento no que a máquina estea libre, escollemos a cadea co ρ-factor máis alto e procesámola sen interrupcións ata chegar á tarefa que determina o ρ-factor (incluíndo dita tarefa). Este proceso repítese coas cadeas que vaian quedando ata que todas as tarefas estean procesadas.*

Para ver claramente como funciona o algoritmo anterior vexamos un exemplo.

Exemplo 2.6. Problema de horarios para o tratamento de radioterapia nunha única máquina con prioridades entre os pacientes. Seguimos no mesmo contexto que no Exemplo 1.1 respectando os termos xa definidos. Recollemos a información necesaria no Cadro 2.1. Como novidade, temos 6 pacientes ($n = 6$) e tense en conta a súa prioridade médica en función do avanzada que estea a súa enfermidade. A importancia do paciente j vén recollida na última

columna, correspondente á prioridade médica de cada paciente, canto maior sexa este valor, maior prioridade terá: $w_1 = 3, w_2 = 1, ..., w_6 = 2$. Recordemos que no hospital existe unha única máquina e os tratamentos aplícanse de forma continua a cada paciente de maneira que ata que un remate, non pode comezar o seguinte.

Paciente	Tempo de tratamento (min)	Horario desexado (minuto do día)	Prioridade
1	30	60	3
2	20	110	1
3	40	100	2
4	25	90	4
5	35	180	5
6	50	210	2

Cadro 2.1: Datos dos pacientes

Neste problema a función obxectivo a minimizar será o tempo total de finalización ponderado, é dicir, $\sum_{j=1}^{n} w_j C_j$. Polo tanto, este problema de horarios podemos describilo coa terna $1|\ prec\ |\sum_{j=1}^{n} w_j C_j$.

Consideremos as seguintes cadeas entre os 6 pacientes:

$$\begin{aligned} \text{Cadea I} &\rightarrow 1, 2, 3, 4 \\ \text{Cadea II} &\rightarrow 5, 6 \end{aligned}$$

Centrarémonos na importancia e os tempos de procesamento dos pacientes.

Pacientes (j)	1	2	3	4	5	6
w_j	3	1	2	4	5	2
p_j	30	20	40	25	35	50

Para comezar a aplicar o Algoritmo 2.5 recordemos a Definición 2.1 e calculamos o ρ-factor de cada cadea.

A fórmula para calculalo é a seguinte:

$$\rho(1, ..., k) = \frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} = \max_{1 \leq l \leq k} \left(\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j}\right)$$

1. Comezamos calculando o ρ-factor das dúas cadeas.

Para a Cadea I:

$$\left.\begin{array}{rcl} \frac{\sum_{j=1}^{1} w_j}{\sum_{j=1}^{1} p_j} & = & \frac{3}{30} = 0.1 \\ \frac{\sum_{j=1}^{2} w_j}{\sum_{j=1}^{2} p_j} & = & \frac{3+1}{30+20} = 0.08 \\ \frac{\sum_{j=1}^{3} w_j}{\sum_{j=1}^{3} p_j} & = & \frac{3+1+2}{30+20+40} = 0,0\overline{6} \\ \frac{\sum_{j=1}^{4} w_j}{\sum_{j=1}^{4} p_j} & = & \frac{3+1+2+4}{30+20+40+25} = 0.087 \end{array}\right\} \Rightarrow \rho(1,...,4) = 0.1 \text{ determinado polo paciente 1.}$$

Para a Cadea II:

$$\left.\begin{array}{rcl} \frac{\sum_{j=5}^{5} w_j}{\sum_{j=5}^{5} p_j} & = & \frac{5}{35} = 0.14 \\ \frac{\sum_{j=5}^{6} w_j}{\sum_{j=5}^{6} p_j} & = & \frac{5+2}{35+50} = 0.082 \end{array}\right\} \Rightarrow \rho(5,6) = 0.14 \text{ determinado polo paciente 5.}$$

Como $\frac{5}{35} > \frac{3}{30}$ comezamos polo paciente 5 e como este é o paciente que determina o ρ-factor detémonos.

2. De acordo co algoritmo, agora debemos repetir o proceso coas tarefas non procesadas das cadeas restantes. O ρ-factor da cadea I non varía, e xa sabemos que é $\frac{3}{30}$. O ρ-factor da parte restante da cadea II será:

$$\rho(6) = \frac{\sum_{j=6}^{6} w_j}{\sum_{j=6}^{6} p_j} = \frac{2}{50} = 0.04 \text{ determinado polo paciente 6.}$$

Como $\frac{3}{30} > \frac{2}{50}$ o seguinte en recibir o tratamento será o paciente 1.

3. Continuamos aplicando o algoritmo. Neste caso, o ρ-factor da cadea II non varía e segue sendo $\frac{2}{50}$. Recalculamos o ρ-factor do que queda da Cadea I:

$$\left.\begin{array}{rcl} \frac{\sum_{j=2}^{2} w_j}{\sum_{j=2}^{2} p_j} & = & \frac{1}{20} = 0.05 \\ \frac{\sum_{j=2}^{3} w_j}{\sum_{j=2}^{3} p_j} & = & \frac{1+2}{20+40} = 0.05 \\ \frac{\sum_{j=2}^{4} w_j}{\sum_{j=2}^{4} p_j} & = & \frac{1+2+4}{20+40+25} = 0.082 \end{array}\right\} \Rightarrow \rho(2,3,4) = 0.082 \text{ determinado polo paciente 4.}$$

Seguindo o algoritmo, como $\frac{7}{85} > \frac{2}{50}$, escollemos a Cadea I e procesámola ata o paciente que determina o ρ-factor, é dicir, daráselle o tratamento ao paciente 2, 3 e 4, seguindo esa orde. Logo, o último paciente en recibir o tratamento será o número 6.

En síntese, a secuencia a seguir para aplicar os tratamentos será: 5 - 1 - 2 - 3 - 4 - 6. En base a dita secuencia presentamos no Cadro 2.2 a información inicial de cada paciente xunto co momento do día no que finaliza o seu tratamento.

Logo, o tempo total ponderado, calculado coa fórmula $\sum_{j=1}^{n} w_j C_j$, será:

$$3 \cdot 65 + 1 \cdot 85 + 2 \cdot 125 + 4 \cdot 150 + 5 \cdot 35 + 2 \cdot 200 = 1705 \text{ minutos.}$$

Orde dos tratamentos	5	1	2	3	4	6
p_j	35	30	20	40	25	50
w_j	5	3	1	2	4	2
C_j	35	65	85	125	150	200

Cadro 2.2: Solución

2.2. Datas de inicio

Ata o momento estivemos considerando tarefas sen unha data concreta de inicio, é dicir, todas elas estaban dispoñibles para comezar en tempo igual a 0. Consideremos agora unha nova situación na que as tarefas teñen unha data de inicio diferente e que ademais, poden ser interrompidas, o que denotamos por r_j e por *prmp*, respectivamente. Definimos o novo problema coa terna $1 \mid r_j, prmp \mid \sum_j w_j C_j$.

A primeira cuestión a contemplar é se a regra $WSPT$ na súa versión con dereitos de preferencia será óptima para estes problemas. Esta nova versión permítenos presentar o seguinte algortimo.

Algoritmo 2.7. ***Tempo total de finalización ponderado e datas de inicio.*** *As tarefas ordénanse en función de cal teña a maior relación entre a súa importancia (w_j) e o seu tempo de procesamento restante, tendo en conta que, como temos o valor prmp para o parámetro β, calquera tarefa que estea a ser procesada pode ser interrompida.*

Dado que o tempo de procesamento restante é un valor que vai diminuíndo co tempo e tendo en conta que na relación que acabamos de definir está situado no denominador, como o numerador (w_j) é constante, esta relación irá aumentando a medida que avance a tarefa. Isto é equivalente a que o nivel de prioridade dunha tarefa que está a ser procesada vai ir aumentando. Polo que no momento no que se seleccionou unha tarefa entre as que estaban dispoñibles, esta non vai ser interrompida por ningunha das restantes, xa que nas que non están a ser procesadas, a relación non varía o seu valor pois o tempo de procesamento restante non está sendo modificado.

Porén, como cada tarefa ten a súa data de inicio r_j, se agora chega unha nova tarefa que antes non estaba dispoñible e presenta un maior nivel de prioridade, comezará a ser procesada interrumpindo á que estaba ata o momento.

Ao aplicar o algoritmo que vimos de describir, non temos asegurado obter o horario óptimo pois estes problemas pertencen á categoría dos *NP-hard*, é dicir, á categoría dos problemas máis complicados de resolver.

Vexamos como funciona esta regra cun exemplo concreto.

Exemplo 2.8. Problema de horarios para o tratamento de pacientes cun único recurso dispoñible tendo en conta as súas prioridades, datas de chegada e permitindo que sexan interrompidos. Neste exemplo abordamos un problema de planificación nun contexto hospitalario, no que se asignan sesións a pacientes nun único recurso dispoñible (por exemplo, unha sala especializada ou unha máquina compartida). A diferenza dos tratamentos de radioterapia reais, que requiren que cada sesión se complete sen interrupcións, aquí consideramos un escenario alternativo no que **as sesións poden ser interrompidas e retomadas máis adiante**, sen que isto supoña un prexuízo para o paciente nin para a calidade do servizo.

Este tipo de situación podería darse en contextos como:

- Sesións de *fisioterapia ou rehabilitación*, nas que o terapeuta pode alternar entre pacientes.
- Tratamentos *ambulatorios lixeiros*, como logopedia ou apoio psicolóxico, que permiten pausas ou cambios de orde.
- Procesos *automatizados ou semi-automatizados*, como a calibración de equipos ou simulacións previas ao tratamento.
- Modelos teóricos para o estudo de algoritmos de planificación con interrupcións (*preemptive scheduling*), aplicables a contextos diversos como a xestión de tarefas en sistemas informáticos ou centros de investigación con recursos compartidos.

No Cadro 2.3 recóllense os datos de cada paciente, mantendo os que tiñamos no Exemplo 2.6 e ademais, engadindo os r_j:

- O tempo total requerido para completar o tratamento (p_j).
- A prioridade asociada ao paciente (w_j).
- O horario desexado para finalizar o tratamento (d_j).
- O tempo de chegada ao servizo (r_j).

O obxectivo é analizar distintas políticas de asignación, entre elas a *Weighted Shortest Processing Time first (WSPS)*, e observar como se comportan cando se permite *interromper os tratamentos e retomalos máis adiante*, asegurando sempre que o tempo total asignado a cada paciente sexa completado.

No momento no que comeza o horario dos tratamentos só temos dous pacientes, o 1 e o 2. Calculamos para cada un deles a relación entre a prioridade do tratamento (w_j) e o tempo de tratamento restante, que neste momento será o propio tempo de tratamento (p_j).

Paciente	Tempo de tratamento (min)	Horario desexado (minuto do día)	Prioridade	Chegada
1	30	60	3	0
2	20	110	1	0
3	40	100	2	5
4	25	90	4	5
5	35	180	5	30
6	50	210	2	65

Cadro 2.3: Datos dos pacientes

$$\text{Paciente 1} \quad \rightarrow \quad \frac{3}{30} = 0.1$$
$$\text{Paciente 2} \quad \rightarrow \quad \frac{1}{20} = 0.05$$

Observamos que o factor asociado ao paciente 1 é maior que ó do paciente 2. No momento que comece o tratamento, o tempo de procesamento restante diminuirá, é dicir, o denominador será cada vez menor, polo que a diferencia entre os factores seguirá aumentando, é dicir, o tratamento do paciente 1 nunca será interrompido polo do paciente 2.

Pasados 5 minutos chegan á consulta 2 novos pacientes, pois $r_3 = r_4 = 5$. Neste momento o paciente 1 segue recibindo o seu tratamento e o seu tempo restante é de $p_1 - 5 = 30 - 5 = 25$ minutos. Mentres que factor do paciente 2 segue sendo o mesmo, recalculamos o novo factor do paciente 1 e calculamos o correspondente aos pacientes 3 e 4.

$$\text{Paciente 1} \quad \rightarrow \quad \frac{3}{25} = 0.12 \qquad \text{Paciente 3} \quad \rightarrow \quad \frac{2}{40} = 0.05$$
$$\text{Paciente 2} \quad \rightarrow \quad \frac{1}{20} = 0.05 \qquad \text{Paciente 4} \quad \rightarrow \quad \frac{4}{25} = 0.16$$

O factor do paciente 4 supera a todos os demais, incluido o do paciente 1, polo tanto, o tratamento deste último será interrumpido para que comece o do paciente 4. De forma análoga ao que dixemos antes para o paciente 1, o factor do paciente 4 a partir deste intre non fará máis que aumentar, e polo tanto rematará co seu tratamento sen ser interrompido polos pacientes 1, 2 e 3.

Repetimos este mesmo proceso ata que todos os tratamentos estén finalizados. O paciente 4 finalizará sen interrupcións o seu tratamento pois non chega ningún novo paciente ata o minuto 30, que é xustamente C_4, o tempo no que o paciente 4 finaliza co seu tratamento. Os factores dos pacientes 1, 2 e 3 non variaron dende o seu último cálculo, mais agora temos un paciente máis, o 5, pois $r_5 = 30$. Calculamos o seu factor para comparalo co dos outros pacientes.

$$\text{Paciente 1} \rightarrow \frac{3}{25} = 0.12 \qquad \text{Paciente 3} \rightarrow \frac{1}{20} = 0.5$$
$$\text{Paciente 2} \rightarrow \frac{1}{20} = 0.5 \qquad \text{Paciente 5} \rightarrow \frac{5}{35} = 0.14$$

É maior o factor do paciente 5 polo que nada máis chegar comeza co seu tratamento. Igual que nos dous casos anteriores, o factor non fará máis que aumentar e desta maneira rematará o tratamento sen ser interrumpido polos pacientes 1, 2 e 3. Dado que non chega un novo paciente ata o minuto 65 e $C_5 = 65$, o paciente 5 recibirá o tratamento sen ser interrompido. Chegamos ao minuto 65, volvemos a ter 4 pacientes pois vén de chegar o número 6 ($r_6 = 65$). Calculamos o factor do novo paciente mentres que os de 1, 2 e 3 seguen sen variar.

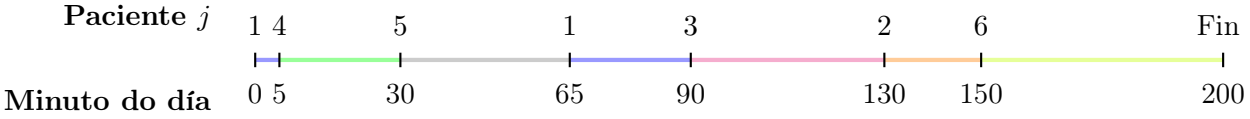
$$\text{Paciente 1} \rightarrow \frac{3}{25} = 0.12 \qquad \text{Paciente 3} \rightarrow \frac{1}{20} = 0.05$$
$$\text{Paciente 2} \rightarrow \frac{1}{20} = 0.05 \qquad \text{Paciente 6} \rightarrow \frac{2}{50} = 0.04$$

Observamos que o maior factor é o do paciente 1, polo que retoma o seu tratamento e como xa non se esperan novos pacientes rematará sen ser interrompido, como lle restaban 25 minutos de tratamento, finalizará no minuto 90. Neste intre os factores dos pacientes 2 e 3 son iguais e maiores que o do paciente 6. En caso de igualdade, non hai ningún criterio establecido, polo que pódese escoller calquera paciente. Consideramos por exemplo o que ten menor d_j, é dicir, o paciente 3. Este pasa a recibir o tratamento e remata no minuto 130 e como os factores de 2 e 6 non variaron e o do paciente 2 é maior, recibirá o tratamento 2 e logo 6.

Finalmente obtivemos a secuencia de tratamentos 1 - 4 - 5 - 1 - 3 - 2 - 6. Vexámola gráficamente nunha liña temporal. Empregaremos o seguinte código de cores para visualizar que paciente está a recibir o tratamento en cada intre.

1 2 3 4 5 6

Dito código empregarase tamén nas seguintes liñas temporais que representemos ao longo do traballo. Ademais, o número do paciente aparece cando este comeza o seu tratamento ou cando o retoma tras ser interrompido. A liña temporal asociada á secuencia que vimos de obter é a seguinte.



Vexamos cal é o tempo total ponderado empregando a fórmula $\sum_{j=1}^{n} w_j C_j$. Recollemos no Cadro 2.4 a información necesaria para o cálculo. Na fila dos p_j os números co * denotan o

tempo de tratamento que recibe o paciente j en cada etapa, é dicir, o tempo que o recibe ata ser interrompido e o tempo que o recibe tras retomalo de novo. A suma dos valores co * debe ser igual a p_j.

Orde dos tratamentos	1	4	5	1	3	2	6
p_j	5*	25	35	25*	40	20	50
w_j		4	5	3	2	1	2
C_j		30	65	90	130	150	200

Cadro 2.4: Solución posible

Entón, o tempo total ponderado será:

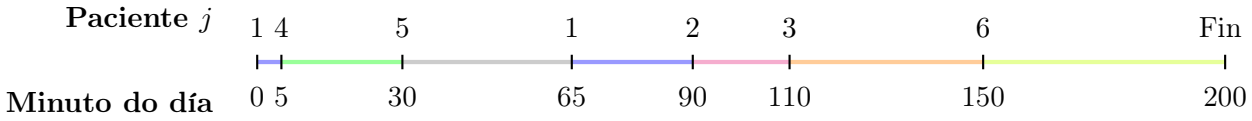
$$\sum_{j=1}^{n} w_j C_j = 3 \cdot 90 + 1 \cdot 150 + 2 \cdot 130 + 4 \cdot 30 + 5 \cdot 65 + 2 \cdot 200 = 1525 \text{ minutos.}$$

Vexamos agora que sucedería se no momento que obtivemos o mesmo factor para o paciente 2 e 3 escollemos ao paciente 2 para recibir o tratamento antes que ao número 3. A secuencia final sería 1 - 4 - 5 - 1 - 2 - 3 - 6. Calculemos o tempo total ponderado de dita secuencia empregando os datos do seguinte cadro.

Orde dos tratamentos	1	4	5	1	2	3	6
p_j	5*	25	35	25*	20	40	50
w_j		4	5	3	1	2	2
C_j		30	65	90	110	150	200

$$\sum_{j=1}^{n} w_j C_j = 3 \cdot 90 + 1 \cdot 110 + 2 \cdot 150 + 4 \cdot 30 + 5 \cdot 65 + 2 \cdot 200 = 1525 \text{ minutos.}$$

A liña temporal correspondente á nova secuencia será a seguinte.



Observamos que en ambos casos obtemos o mesmo valor para o tempo total ponderado polo que efectivamente, ao obter o mesmo factor para dous pacientes, é indiferente cal tratar primeiro. Como dicíamos na explicación da regra, trátase dun problema moi complicado para o que dita regra non asegura a optimalidade global da solución.

Capítulo 3

Funcións obxectivo dependentes do tempo de finalización das tarefas

En todas as funcións obxectivo do capítulo anterior tiñamos o parámetro w_j, que indicaba a importancia da tarefa j, e no caso concreto dos nosos exemplos, da prioridade do tratamento de cada paciente. Con esta información podiamos dar unha aproximación simplificada da realidade mais ao tratarse de prioridades discretas, limitan a capacidade de representar a realidade con precisión.

Centrándonos agora na situación dos tratamentos de radioterapia, estes valores fixos son independentes do momento no que se atende ao paciente e supoñen que o impacto de cada minuto de retraso no tratamento é constante, suposición errónea xa que non todos os pacientes teñen a enfermidade igual de avanzada. Outra limitación das w_j é que non teñen en conta situacións onde o risco ou a urxencia do paciente aumente de forma acelerada ou cambiante no tempo.

Por todas esas cuestións, neste capítulo introducimos as **funcións de custo** $h_j(C_j)$, onde C_j representa o tempo de finalización da tarefa (ou paciente) j. Estas funcións permiten modelar situacións máis realistas ao poder ter en conta moitos dos factores que comentamos antes.

No contexto clínico que estamos a traballar, as h_j permiten modelar a realidade de forma dinámica xa que existe a posibilidade de asociar unha función diferente a cada paciente segundo as súas necesidades específicas. Nestas poderase incorporar información como a súa gravidade, o tempo que levan esperando, o número de sesións restantes ou a resposta que teñen ao tratamento.

A partir de agora, traballaremos con problemas descritos pola tripla $1|\ prec\ |\ h_{max}$, onde:

$$h_{max} = \text{máx}(h_1(C_1), ..., h_n(C_n))$$

con h_j, $j = 1, ..., n$, serán funcións de custo non decrecentes. Claramente, o obxectivo destes

problemas será minimizar h_{max}. Para os servizos de radioterapia con recursos limitados como os que estamos a considerar (pois presentan unha soa máquina), este tipo de problema de horarios son clave á hora de planificar os tratamentos e tomar decisións en base ao impacto real dos retrasos, mellorando así considerablemente a atención ao paciente.

Para o problema que vimos de definir, independientemente de cales sexan as funcións h_j, podemos aplicar o algoritmo que presentaremos a continuación. Trátase dun algoritmo que constrúe a secuencia ao revés, é dicir, decide primeiro que tarefa vai ao final e logo cal vai antes dela, así ata construir unha secuencia con todas elas. Primeiro introducimos unha serie de conceptos necesarios:

- $C_{max} = \sum_j p_j$: tempo no que se completa a última tarefa (independiente da secuencia na que estas sexan procesadas).
- J: conxunto de tarefas que xa foron secuenciadas. O seu intervalo de tempo de procesamento será $[C_{max} - \sum_{j \in J} p_j,\ \ C_{max}]$.
- J^c: o complementario de J. Conxunto de tarefas que aínda non foron secuenciadas.
- $J' \subset J^c$: tarefas que non teñen sucesores pendentes, é dicir, independientes das tarefas de J^c.

Algoritmo 3.1. ***O custo máis baixo de último****. O obxectivo deste algoritmo é minimizar o maior custo de finalización $h_j(C_j)$. Para levalo a cabo temos que seguir os seguintes pasos:*

1. *Temos $J = \emptyset$, $J^c = \{1, ..., n\}$ e J' o conxunto dos traballos dispoñibles para ser secuenciados.*
2. *Tomamos a tarefa $j^* \in J' \subset J^c$ tal que verifique:*

$$h_{j^*}(\sum_{k \in J^c} p_k) = \min_{j \in J'}(h_j(\sum_{k \in J^c} p_k))$$

   é dicir, que minimiza o custo de finalización se se programa ao final. Destacamos que $\sum_{k \in J^c} p_k$ representa o tempo total restante, pois estamos contruindo a secuencia ao revés. Engadimos j^* a J (á secuencia resultante), mentres que a eliminamos de J^c (xa non está pendente). Modificamos tamén o conxunto J' representando todas as tarefas que neste momento non teñen sucesores en J^c.
3. *Se $J^c = \emptyset$, teríamos todas as tarefas asignadas a unha posición na secuencia, e rematamos coa execución do algoritmo. En caso contrario, volvemos ao paso 2 para elixir a seguinte tarefa que irá xusto antes da que acabamos de engadir.*

   Desta forma estamos construindo unha secuencia que sitúa o máis pronto posible a tarefa cun custo maior, evitando así que aumente h_{max}.

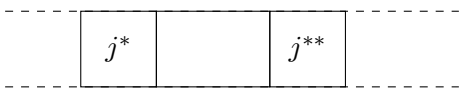
Antes de presentar un exemplo para comprender mellor como funciona este algoritmo, probaremos que este dará lugar a un horario óptimo para os problemas que estamos a considerar.

Teorema 3.2. *O Algoritmo 3.1 proporciona un horario óptimo para o problema 1| prec | h_{max}*

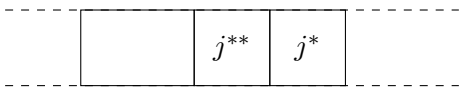
Demostración. Supoñamos que o algoritmo non determina un horario óptimo e vexamos como chegamos a unha contradición.

Nunha certa iteración, seleccionamos a tarefa $j^{**} \in J'$ tal que non ten o menor custo de finalización, é dicir, non verifica $h_{j^{**}}(\sum_{k\in J^c} p_k) = \min_{j\in J'}(h_j(\sum_{k\in J^c} p_k))$. Tras supoñer o anterior, sabemos que existe unha tarefa $j^* \in J'$ cun custo menor que j^{**} ($h_{j^*} < h_{j^{**}}$) que si verifica a igualdade anterior. Esta non foi seleccionada nesta iteración, é dicir, aínda non será secuenciada.

Como estamos a ordenar as tarefas dende a última ata a primeira, j^* aparecerá antes que j^{**}, podendo existir incluso outras tarefas entre elas. Apoiamonos na seguinte representación para visualizar a secuencia definida, onde o rectángulo en branco representa as posibles tarefas intermedias das que estamos a falar. Supoñamos que esta secuencia fose óptima.



Construimos agora unha nova secuencia de tarefas situando j^* xusto despois que j^{**}, podemos observala na seguinte representación. Vexamos que dita secuencia, construida cumprindo as condicións do Algoritmo 3.1, é mellor que a anterior, chegando así á contradición que buscamos.



É evidente que as tarefas entre j^* e j^{**}, incluíndo j^{**}, rematan antes nesta segunda secuencia que na primeira, e en consecuencia, reducen o seu custo de finalización. A única tarefa que se atrasa é j^*, é dicir, a única que aumenta o seu custo de finalización. Aínda así, baixo as nosas suposicións iniciais, o novo custo de j^* segue sendo menor que o custo orixinal de j^{**} (na primeira secuencia), pois tiñamos que $h_{j^*} < h_{j^{**}}$.

En síntese, aínda que o custo de j^* aumentase na segunda secuencia con respecto á primeira, o custo máximo de finalización diminuiu, chegando así a unha contradición con que a primeira secuencia que definimos fose a óptima. Queda probado que a única forma de chegar á secuencia óptima é seguindo o algoritmo do custo máis baixo de último. Esta demostración é unha ampliación da recollida en [4].

□

Seguindo a liña dos exemplos empregados ata o de agora, continuamos traballando na situación dos tratamentos de radioterapia. O seguinte será un breve exemplo que nos servirá para comprender o funcionamento do Algoritmo 3.1.

Exemplo 3.3. Minimizar o custo máximo de tratamento nunha máquina de radioterapia. Consideramos 3 pacientes xunto cos seus respectivos tempos de tratamento (p_j) e a súa función de custo asociada $h_j(C_j)$. Podemos interpretar esta última como o impacto que supón para o paciente o retraso dos tratamentos, sendo este impacto maior se o paciente se atopa nunha situación moi sensible e menor se o paciente está máis estable.

Pacientes	1	2	3
p_j	30	20	40
$h_j(C_j)$	$C_1 + 3$	$1.2C_2$	10

Observamos que o impacto do paciente 1 aumenta de maneira lineal en igual medida que o tempo que tarda en recibir o tratamento, cun malestar de base de 3 unidades se é atendido ou non de inmediato. O impacto do paciente 2 tamén aumenta a medida que aumenta o retraso pero cunha taxa maior, pode asociarse a que a súa enfermidade é máis agresiva e así este atópase máis sensible. Por último, claramente o paciente 3 é o máis estable dos presentes, penaliza os retrasos pero en menor medida que o paciente 2 e sen ningún malestar de base como o paciente 1.

Comezamos coa primeira iteración do algoritmo.

1. Temos $J = \emptyset$, $J^c = \{1, 2, 3\}$ e, neste caso, $J' = \{1, 2, 3\}$, pois ningún dos pacientes ten un sucesor pendente.

2. Calculamos a tarefa que minimiza o custo de finalización se se programa ao final baixo a fórmula:

$$h_{j^*}(\sum_{k \in J^c} p_k) = \min_{j \in J'}(h_j(\sum_{k \in J^c} p_k))$$

Primeiro obtemos $\sum_{k \in J^c} p_k = 30 + 20 + 40 = 90$, e logo calculamos:

$$\left.\begin{aligned} h_1(90) &= 90 + 3 = 93 \\ h_2(90) &= 1.2 \cdot 90 = 108 \\ h_3(90) &= 10 \end{aligned}\right\} \Rightarrow h_{j^*}(90) = 10 \text{ determinado polo paciente } j^{**} = 3.$$

Entón, colocamos ao paciente 3 como o último en recibir o tratamento, o que parece lóxico tendo en conta os comentarios que fixemos anteriormente en canto ao impacto que tiñan os retrasos nel.

Actualizamos os conxuntos do algoritmo: $J = \{3\}$, $J^c = \{1, 2\}$ e $J' = \{1, 2\}$.

3. Como $J^c = \{1, 2\} \neq \emptyset$, volvemos ao paso 2 para elixir ao seguinte paciente que irá antes que o 3.

Seguimos coa segunda iteración do algoritmo.

2. De entre os pacientes cuxo tratamento nos queda por secuenciar, vexamos cal é o que minimiza o custo de finalización.

   Primeiro obtemos $\sum_{k \in J^c} p_k = 30 + 20 = 50$, e logo calculamos:

$$\left.\begin{array}{r} h_1(50) = 50 + 3 = 53 \\ h_2(50) = 1.2 \cdot 50 = 60 \end{array}\right\} \Rightarrow h_{j^*}(50) = 53 \text{ determinado polo paciente } j^{**} = 1.$$

   O paciente 1 será quen reciba o tratamento xusto antes que o 3. Actualizamos os conxuntos do algoritmo: $J = \{1, 3\}$, $J^c = \{2\}$ e $J' = \{2\}$

3. Como $J^c = \{2\} \neq \emptyset$, volvemos ao paso 2 para xa rematar co proceso.

Pasamos á terceira e última iteración.

2. Só temos un elemento en J^c, logo, tendo en conta que $\sum_{k \in J^c} p_k = 20$

$$h_2(20) = h_{j^*}(20) = 1,2 \cdot 20 = 24$$

   Entón, o paciente 2 será o primeiro en recibir o tratamento, coherente co impacto que lle supoñen os retrasos. Actualizamos os conxuntos do algoritmo: $J = \{2, 1, 3\}$ e $J^c = J' = \emptyset$.

3. Como $J^c = \emptyset$, rematamos.

Finalmente, apoiándonos no Teorema 3.2, a secuencia óptima de tratamentos será 2-1-3, o que, como fomos comentando, ten sentido segundo o impacto que supoñen os retrasos nos tres pacientes.

Unha vez tratado de forma xeralizada este tipo de problemas, imos introducir algunhas das opcións posibles para as funcións $h_j(C_j)$. A continuación presentamos tres exemplos xunto coas súas representacións gráficas:

$$(a)\ L_j = C_j - d_j \qquad (b)\ T_j = \text{máx}(C_j - d_j, 0) = \text{máx}(L_j, 0) \qquad (c)\ U_j = \begin{cases} 1 & \text{si } C_j > d_j \\ 0 & \text{noutro caso} \end{cases}$$

Como vemos na Figura 3.1, as tres funcións son non decrecentes. Nas seguintes seccións estudaremos os problemas de horarios asociados a cada unha delas e os algoritmos correspondentes.

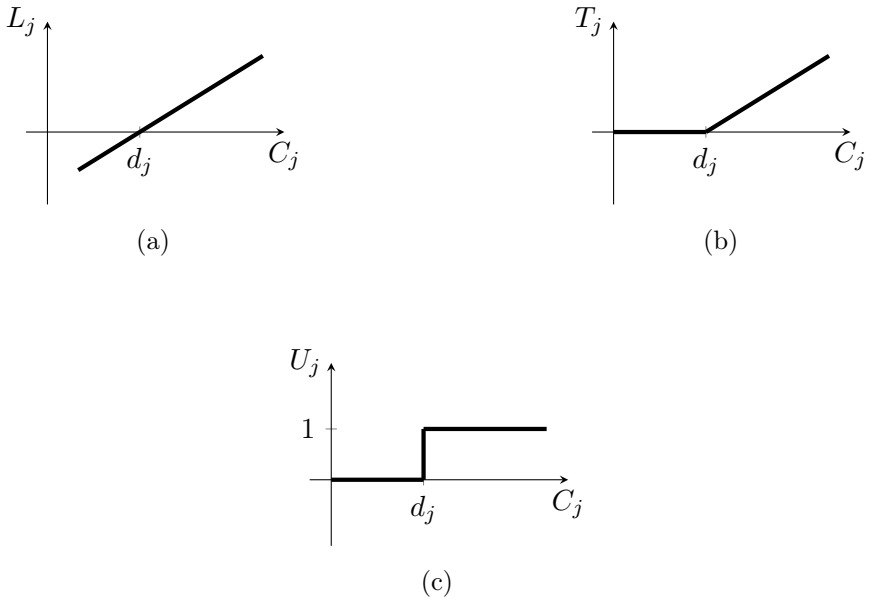


Figura 3.1: Representación das posibles h_j

3.1. Tardanza máxima

Comezamos estudando os problemas de horarios coa función de custo:

$$h_j(C_j) = L_j(C_j) = C_j - d_j.$$

Tendo en conta que C_j representa o momento no que finaliza a tarefa j, estamos a calcular o retraso de dita tarefa. Definimos estes problemas coa terna $1|\ prec\ |\ L_{max}$, onde o obxectivo a minimizar vén dado por:

$$L_{max} = \text{máx}(L_1(C_1), ..., L_n(C_n))$$

é dicir, buscamos minimizar o retraso máximo entre as tarefas que temos. Para resolver estes problemas empregamos a regra *Earliest Due Date first* (EDD), que elabora os horarios ordenando as tarefas en orde crecente da súa data límite d_j.

Este tipo de problemas están moi estudados na literatura xa que aparecen con frecuencia como subproblemas dentro de procesos heurísticos máis amplios. Son moi relevantes en contextos onde se prioriza o cumprimento da data límite como na planificación de proxectos ou na xestión de tarefas en sistemas en tempo real.

Unha posible xeralización dos problemas anteriores sería a terna $1|\ r_j\ |\ L_{max}$, coas tarefas comezando en distintos tempos. A pesar da súa formulación sinxela, estes problemas adoitan ser moi complexos. Presentamos o seguinte resultado do que podemos atopar a proba en [4].

Teorema 3.4. *Os problemas definidos por $1|\ r_j\ |\ L_{max}$ son fortemente NP-hard.*

Isto levou a desenvolver algoritmos exactos de busca sistemática con poda intelixente para situacións máis sinxelas e o uso de procesos heurísticos para casos de maior tamaño. O horario óptimo non será necesariamente un horario sen retrasos.

Nesta sección centrarémonos no primeiro caso de problemas máis simples, estudando procedementos de ramificación e acotación para resolvelos. Estes algoritmos enumeran unha serie de horarios de diferentes tipos buscando atopar a solución óptima sen ter que estudar todas as posibles combinacións de secuencias.

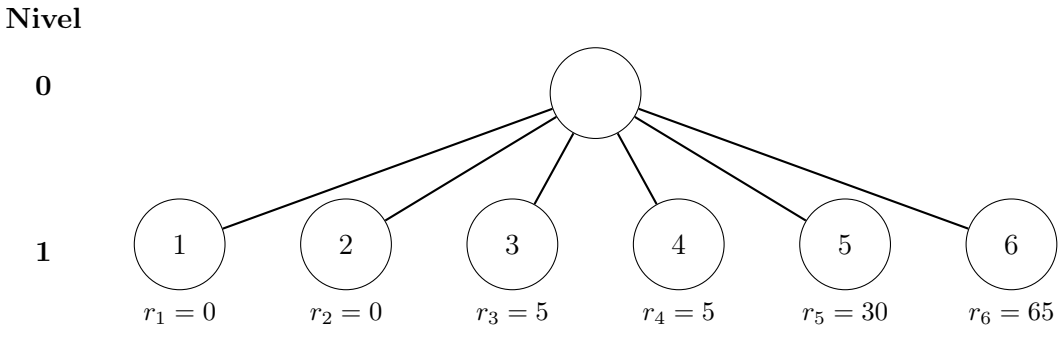
Para ver como se emprega este algoritmo aplicarémolo directamente no seguinte exemplo.

Exemplo 3.5. Problema de horarios para o tratamento de radioterapia nunha única máquina tendo en conta as datas de chegada e datas límite de cada paciente. Algoritmo de ramificación e acotación. Para este exemplo tomamos os mesmos datos dos pacientes do Exemplo 2.8 mais só necesitaremos os referidos ao tempo total do tratamento (p_j), o horario desexado para rematar con el (d_j) e o tempo de chegada ao servizo (r_j).

O problema de horarios que imos tratar podemos describilo coa terna $1|\ r_j\ |\ L_{max}$, é dicir, ao non aparecer o parámetro *prmp*, non se poderán interrumpir os tratamentos dos pacientes. Isto permítenos volver ao contexto dos tratamentos de radioterapia que describiamos no Exemplo 1.1. Recollemos os datos que nos interesan no seguinte cadro.

Pacientes (j)	1	2	3	4	5	6
p_j	30	20	40	25	35	50
r_j	0	0	5	5	30	65
d_j	60	110	100	90	180	210

En base a eles, comezamos coa elaboración da árbore coas diferentes secuencias de tratamentos. No nivel 0 temos tan só un nodo sen ningún paciente asociado. Deste sairán 6 arcos dirixidos cara 6 nodos onde cada un irá asociado a un paciente diferente, constituíndo así o nivel 1 da árbore.



Dado que o mellor é comezar cos tratamentos dende o minuto 0, observamos que nese momento só están dispoñibles o paciente 1 e 2, polo tanto, isto lévanos a desbotar os nodos 3, 4, 5 e 6 para tomalos como iniciais da secuencia. Unha vez feita esta observación seguimos elaborando a árbore representando por simplicidade só os nodos 1 e 2 no nivel 1 e seguindo cara abaixo. Segundo imos avanzando e engadindo novos pacientes á secuencia, os nodos recollerán a subsecuencia que lles corresponda en cada caso.

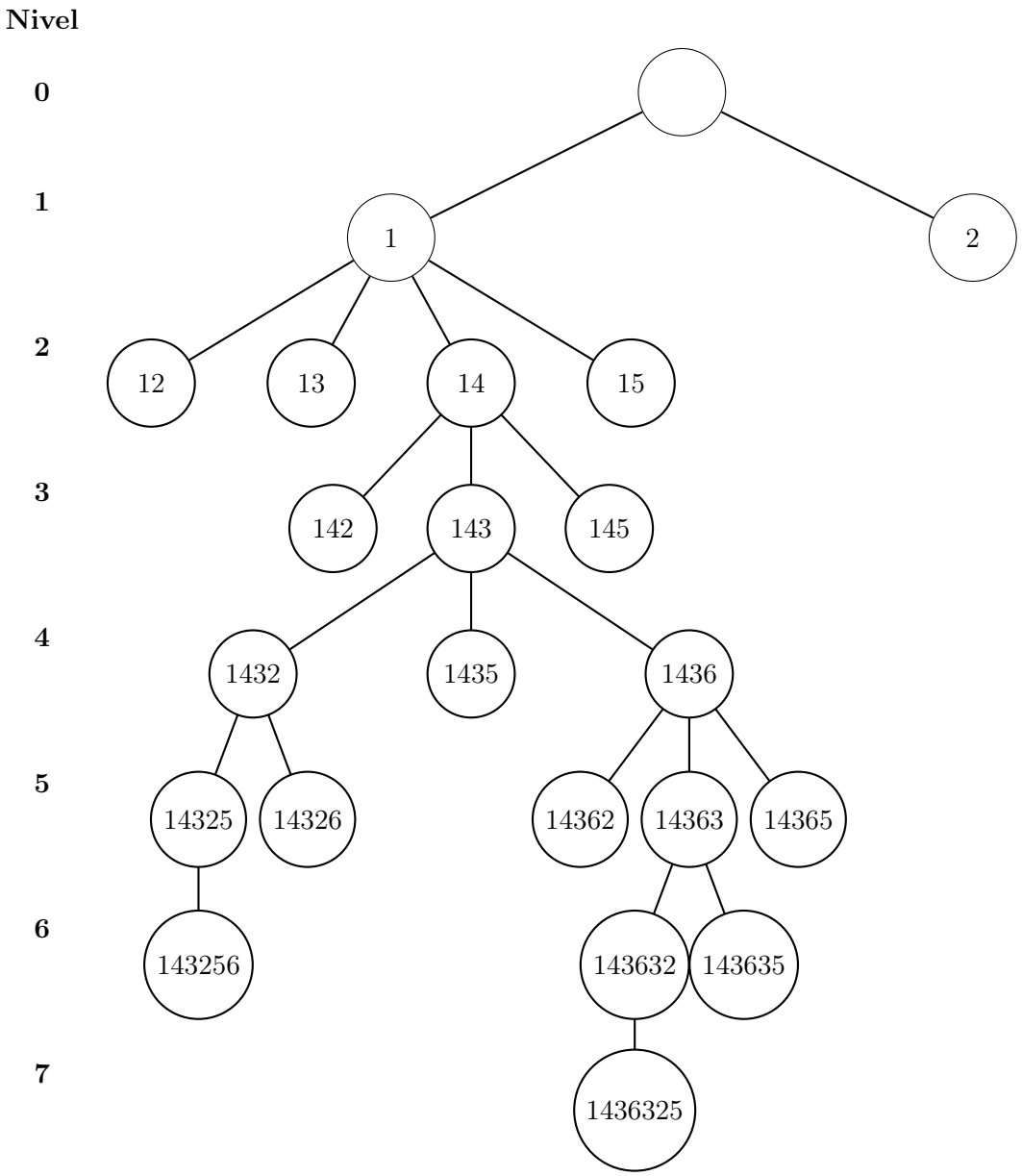


Figura 3.2: Árbore coas dúas solucións

Vexamos agora como foi a construcción desta árbore.

- Seguindo a regra *EDD*, comezamos polo paciente 1 pois $d_1 = 60 < 110 = d_2$, como ademais $d_1 < d_j \ \forall j \in \{2, 3, 4, 5, 6\}$, este recibirá o tratamento sen ser interrompido independientemente da chegada de calquera paciente. Dado que $p_1 = 30$, situámonos no minuto 30, é dicir, $C_1 = 30$.
- Dende o nodo 1 saen 4 arcos cara 4 nodos situados no nivel 2, pois no minuto 30 temos dispoñibles para recibir o tratamento aos pacientes 2, 3, 4 e 5. Como $d_4 < d_j \ \forall \ j \in \ \{2, 4, 5\}$, comeza a recibir tratamento o paciente 4. Tendo en conta que $p_4 = 25$, obtemos $C_4 = 55$, e como non chega ningún novo paciente durante a súa sesión, completará o tratamento sen ser interrompido.
- Pasamos ao nivel 3. Dende o nodo 14 saen tan só 3 arcos, pois os pacientes dispoñibles no minuto 55 son 2, 3 e 5. Como $d_3 < d_j \ \forall j \in \{2, 5\}$, pasa a recibir o tratamento o paciente 3 e tendo en conta que $p_3 = 40$, $C_3 = 95$. Cómpre destacar que $r_6 = 65$, polo que no minuto 65 temos un novo paciente, o número 6. Comparamos as súas datas límite para finalizar o tratamento, e como $d_3 < d_6$, non se interrompe o tratamento do paciente 3. Máis adiante estudaremos o caso no que $d_6 < d_3$.

De forma análoga seguimos aplicando o algoritmo nos niveis 4 e 5, continuando co paciente 2 e 5, obtendo $C_2 = 115$ e $C_5 = 150$. Chegamos ao nivel 6 e pasa a recibir tratamento o paciente 6, obtendo $C_6 = 200$. Unha vez finalizado o proceso chegamos á secuencia de tratamentos 1- 4- 3- 2- 5- 6.

Calculamos a tardanza máxima empregando os datos da seguínte táboa.

Orde dos tratamentos	1	4	3	2	5	6
d_j	60	90	100	110	180	210
C_j	30	55	95	115	150	200
$L_j(C_j)$	-30	-35	-5	5	-30	-10

Finalmente, observamos que $L_{max} = 5$, é dicir, o retraso que se obtén na secuencia definida é de 5 minutos. A liña temporal dos tratamentos será a seguinte.

Como comentamos, vexamos que sucedería se no minuto 65 no que chega o paciente 6, temos que $d_6 < d_3$. Consideremos ademais que agora o noso problema descríbese pola terna

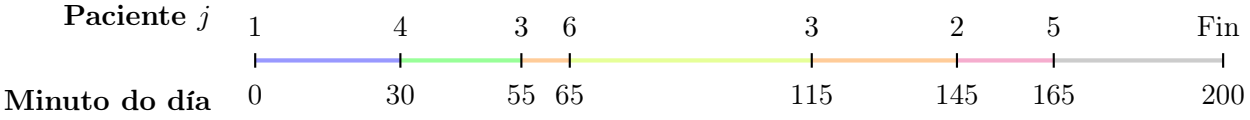
$1 \mid r_j, prmp \mid L_{max}$, é dicir, a diferenza do caso que acabamos de estudar, permítense interrupcións entre os tratamentos. Tomemos para iso $d_6 = 99\,(< d_3 = 100)$. Nesta situación, o paciente 3 deixa de recibir o tratamento e pasa á consulta o paciente 6. Como non chega ningún outro paciente novo, este rematará o tratamento sen ser interrompido. Tendo en conta que $p_6 = 50$, obtemos $C_6 = 115$.

Dende o nodo 1436 saen 3 arcos cara tres nodos correspondentes aos pacientes 2, 3 e 5, pois ao paciente 3 séguenlle quedando 30 minutos de tratamento, xa que comezou no minuto 55 e tivo que deterse no minuto 65 e $p_3 = 40$. Como xa vimos antes, $d_3 < d_j\ \forall i \in \{2, 5\}$ polo que se retoma o tratamento do paciente 3, chegando así a que $C_3 = 145$. Agora, en base ás súas datas límite, recibe o tratamento o paciente 2, obtendo $C_2 = 165$ e logo, para finalizar, o paciente 5, obtendo o tempo de completado $C_5 = 200$. Analizamos os datos da nova secuencia de tratamentos coa seguinte táboa.

Orde dos tratamentos	1	4	3	6	3	2	5
d_j	60	90	100	99	100	110	180
C_j	30	55		115	145	165	200
$L_j(C_j)$	-30	-35		16	45	55	20

Concluimos que coa secuencia 1- 4- 3- 6- 3- 2- 5- 6, $L'_{max} = 55$, sendo este un valor moito maior que o da secuencia anterior. Como sabemos, a situación na que se permiten interrupcións nos tratamentos non se pode aplicar aos tratamentos de radioterapia, polo que esta segunda casuística debe ser interpretada como algún dos casos clínicos comentados no Exemplo 2.8. En vista dos resultados obtidos, cabe destacar que se obtiveron moito mellores resultados empregando unha secuencia sen interrupcións que coa que permite interrupcións ($L_{max} < L'_{max}$).

Presentamos a liña temporal correspondente á segunda secuencia.



3.2. Número de tarefas con retraso.

O seguinte exemplo de función de custo a estudar será $h_j(C_j) = U_j(C_j)$, onde:

$$U_j(C_j) = \begin{cases} 1 & \text{si } C_j > d_j \\ 0 & \text{noutro caso} \end{cases}$$

Recordemos que C_j representa o momento no que finaliza a tarefa j, entón, $U_j(C_j)$ toma o valor 1 se dita tarefa remata despois da súa data de límite de completado, é dicir, se remata con retraso. Neste caso, para o parámetro γ temos $h_{max} = \sum_j U_j(C_j)$, e polo tanto, o obxectivo a minimizar será o número de tarefas que finalicen con retraso. A terna que describe estes problemas será $1|\ |\ \sum_j U_j(C_j)$.

O horario óptimo para este tipo de problemas secuenciará inicialmente as tarefas que vaian cumprir coa súa data límite, seguidas das que non cumpran con elas. As primeiras estarán organizadas seguindo a regra EDD para así asegurar obter $L_{max} < 0$.

Para resolver estes problemas podemos empregar un algoritmo de avance que reordena as tarefas segundo as súas datas límite: $d_1 \leq d_2 \leq ... \leq d_n$. Se supoñemos que temos n tarefas, o algoritmo terá n iteracións. No número k tivéronse en conta as tarefas $1, 2, ..., k$, e pertencerán a algún dos seguintes conxuntos que imos definir:

- J: conxunto das tarefas que na secuencia óptima poden finalizar antes que a súa data de entrega.
- J^d: conxunto coas tarefas xa descartadas pois non van cumprir coas súas correspondentes datas límite na secuencia óptima.
- J^c: o complementario de J, é dicir, contén ás tarefas que aínda non se tiveron en conta, $k+1, k+2, ..., n$.

Algoritmo 3.6. ***Minimizar o número de tarefas con retraso.*** *Antes de comezar, debemos ordenar as tarefas de maneira que $d_1 \leq d_2 \leq ... \leq d_n$. Para aplicar o algoritmo debemos seguir os seguintes pasos:*

1. *Temos que $J = \emptyset$, $J^c = \{1, ..., n\}$ e $J^d = \emptyset$. Poñemos o contador en $k = 1$.*
2. *Engadimos a tarefa k en J e eliminámola de J^c.*
3. *Se se verifica que:*

$$\sum_{j \in J} p_j \leq d_k$$

pasamos ao paso 4.

En caso contrario, denotamos por l á tarefa que cumpre:

$$p_l = \max_{j \in J}(p_j)$$

é dicir, á tarefa co maior tempo de procesamento dentro das que pertencen a J. Á vez, eliminamos a tarefa l de J e engadímola en J^d .

4. *Definimos un novo conxunto $J_k = J$, que representa as tarefas candidatas a cumprir coas súas d_j na secuencia óptima. É importante destacar que estas candidatas van cambiando ao longo das iteracións polo que haberá un conxunto J_k diferente en cada iteración.*

   Se $k = n$, rematamos co algoritmo;

   en caso contrario, tomamos $k = k + 1$ e volvemos ao paso 2.

En resumo, as tarefas ordénanse en orde crecente das súas d_j correspondentes. Ao aplicar o algoritmo engádense tarefas que vaian a cumprir coas súas datas límite ao conxunto J. Se incluir a tarefa k supón que non se vaia a finalizar dentro do seu prazo límite (d_k), pois o seu tempo de completado ($\sum_{j\in J} p_j$) é maior, eliminamos a tarefa con maior tempo de procesamento de J (denotada por l) e engadímola en J^d (conxunto das tarefas completadas con retraso). O conxunto J_n estará formado por todas as tarefas que cumpran coas súas datas límite na secuencia óptima xerada.

Temos o seguinte resultado do que podemos atopar a proba en [4].

Teorema 3.7. *O Algoritmo 3.6 proporciona un horario óptimo para os problemas definidos por $1|\ |\ \sum_j U_j$.*

Para comprender mellor o algoritmo anterior vamos aplicalo no seguinte exemplo.

Exemplo 3.8. Problema de horarios para o tratamento de radioterapia nunha única máquina buscando minimizar o número de tratamentos con retraso. Retomamos os datos que tiñamos dos 6 pacientes do Exemplo 2.8, centrándonos soamente no tempo de duración do tratamento (p_j) e no horario desexado para que o tratamento esté rematado (d_j).

Pacientes (j)	1	2	3	4	5	6
p_j	30	20	40	25	35	50
d_j	60	110	100	90	180	210

Antes de aplicar o algoritmo ordenamos os pacientes segundo as súas d_j de menor a maior:

$$d_1 < d_4 < d_3 < d_2 < d_5 < d_6 \ \Rightarrow\ J^c = \{1, 4, 3, 2, 5, 6\}$$

Comezamos coa primeira iteración do Algoritmo 3.6.

1. Temos $J = \emptyset$, $J^c = \{1, 4, 3, 2, 5, 6\}$ e $J^d = \emptyset$. Poñemos o contador en $k = 1$.

2. Engadimos o paciente 1 en J e eliminámolo de J^c.

3. Comprobamos se verifica a desigualdade:

$$\left.\begin{array}{r} \sum_{j\in J} p_j = 30 \\ d_1 = 30 \end{array}\right\} \Rightarrow \sum_{j\in J} p_j \leq d_k \text{ polo que pasamos ao paso 4.}$$

4. Definimos $J_1 = J = \{1\}$ e como $k = 1 \neq 6$, tomamos $k = 1 + 1 = 2$ e volvemos ao paso 2.

Continuamos coa segunda iteración:

2. Engadimos o paciente 4 en J e eliminámolo de J^c, quedando $J = \{1, 4\}$ e $J^c = \{3, 2, 5, 6\}$.

3. Comprobamos se verifica a desigualdade:

$$\left.\begin{array}{r} \sum_{j\in J} p_j = 30 + 25 = 55 \\ d_4 = 90 \end{array}\right\} \Rightarrow \sum_{j\in J} p_j \leq d_k \text{ polo que pasamos ao paso 4.}$$

4. Definimos $J_2 = J = \{1, 4\}$ e como $k = 2 \neq 6$, tomamos $k = 2 + 1 = 3$ e volvemos ao paso 2.

Comenzamos a terceira iteración do algoritmo:

2. Engadimos o paciente 3 en J e eliminámolo de J^c, quedando $J = \{1, 2, 3\}$ e $J^c = \{2, 5, 6\}$.

3. Comprobamos se verifica a desigualdade:

$$\left.\begin{array}{r} \sum_{j\in J} p_j = 30 + 25 + 40 = 95 \\ d_3 = 100 \end{array}\right\} \Rightarrow \sum_{j\in J} p_j \leq d_k \text{ polo que pasamos ao paso 4.}$$

4. Definimos $J_3 = J = \{1, 4, 3\}$ e como $k = 3 \neq 6$, tomamos $k = 3 + 1 = 4$ e volvemos ao paso 2.

Pasamos á cuarta iteración e vexamos que sucede:

2. Engadimos o paciente 2 en J e eliminámolo de J^c, quedando $J = \{1, 4, 3, 2\}$ e $J^c = \{5, 6\}$.

3. Comprobamos se verifica a desigualdade:

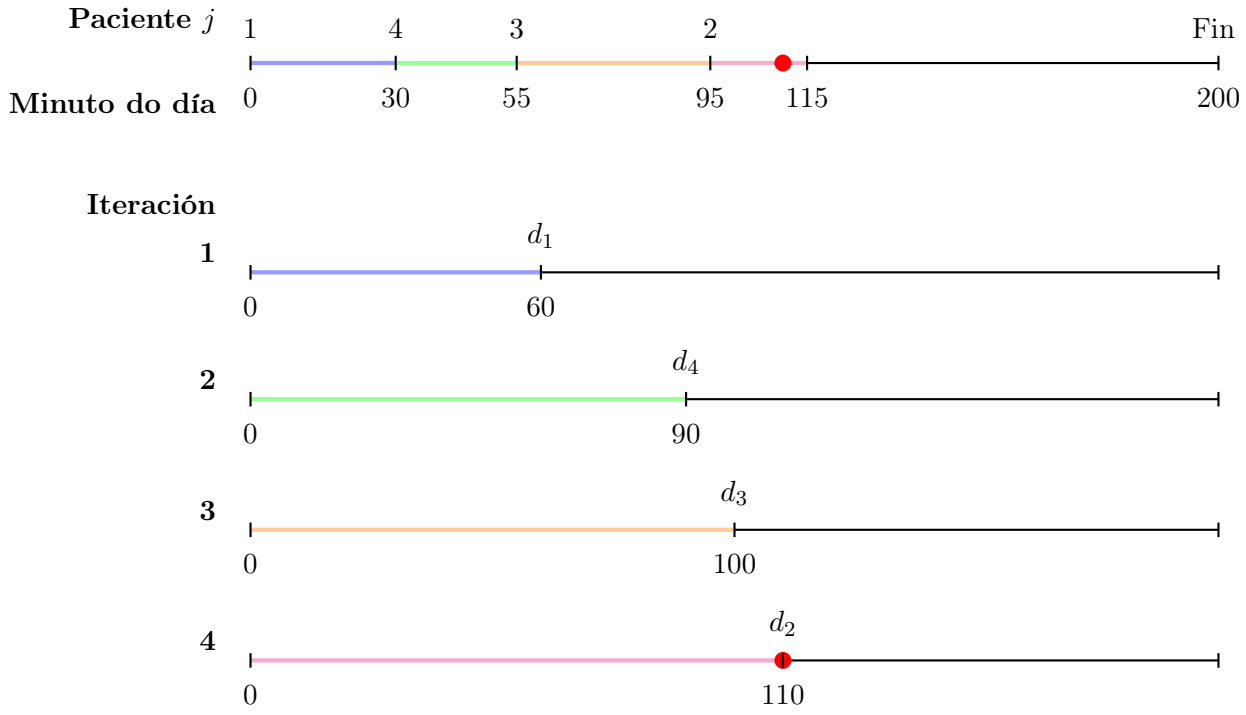
$$\left.\begin{array}{r} \sum_{j\in J} p_j = 30 + 25 + 40 + 20 = 115 \\ d_2 = 110 \end{array}\right\} \Rightarrow \sum_{j\in J} p_j \nleq d_k$$

Como non se cumpre, denotamos por l ao paciente 3, pois é o que verifica $p_l = \text{máx}_{j\in J}(p_j)$, é dicir, é o que ten un periodo de tratamento maior de entre os pacientes considerados ata o momento (dende o 1 ata o 4).

Eliminamos ao paciente 3 do conxunto J e engadímolo a J^d, obtendo $J = \{1, 4, 2\}$ e $J^d = \{3\}$. Observamos que estamos en condicións de afirmar que o paciente 3 rematará o seu tratamento con retraso.

4. Definimos $J_4 = J = \{1, 4, 2\}$ e como $k = 4 \neq 6$, tomamos $k = 4 + 1 = 5$ e volvemos ao paso 2.

Vexámos gráficamente coas seguintes liñas temporais o que estamos comparando nas desigualdades anteriores.



Observamos que ata chegar á cuarta iteración, ningún $\sum_{j \in J} p_j$ supera o valor da d_j correspondente. Nesta última, destacamos cun punto vermello o valor correspondente a d_2, e como este é superado polo tempo total de procesamento das tarefas secuenciadas ata o momento, verifícase, efectivamente, que $\sum_{j \in J} p_j \nleq d_k$. Despois desta aclaración seguimos aplicando o algoritmo xa sen o paciente 3 na secuencia inicial.

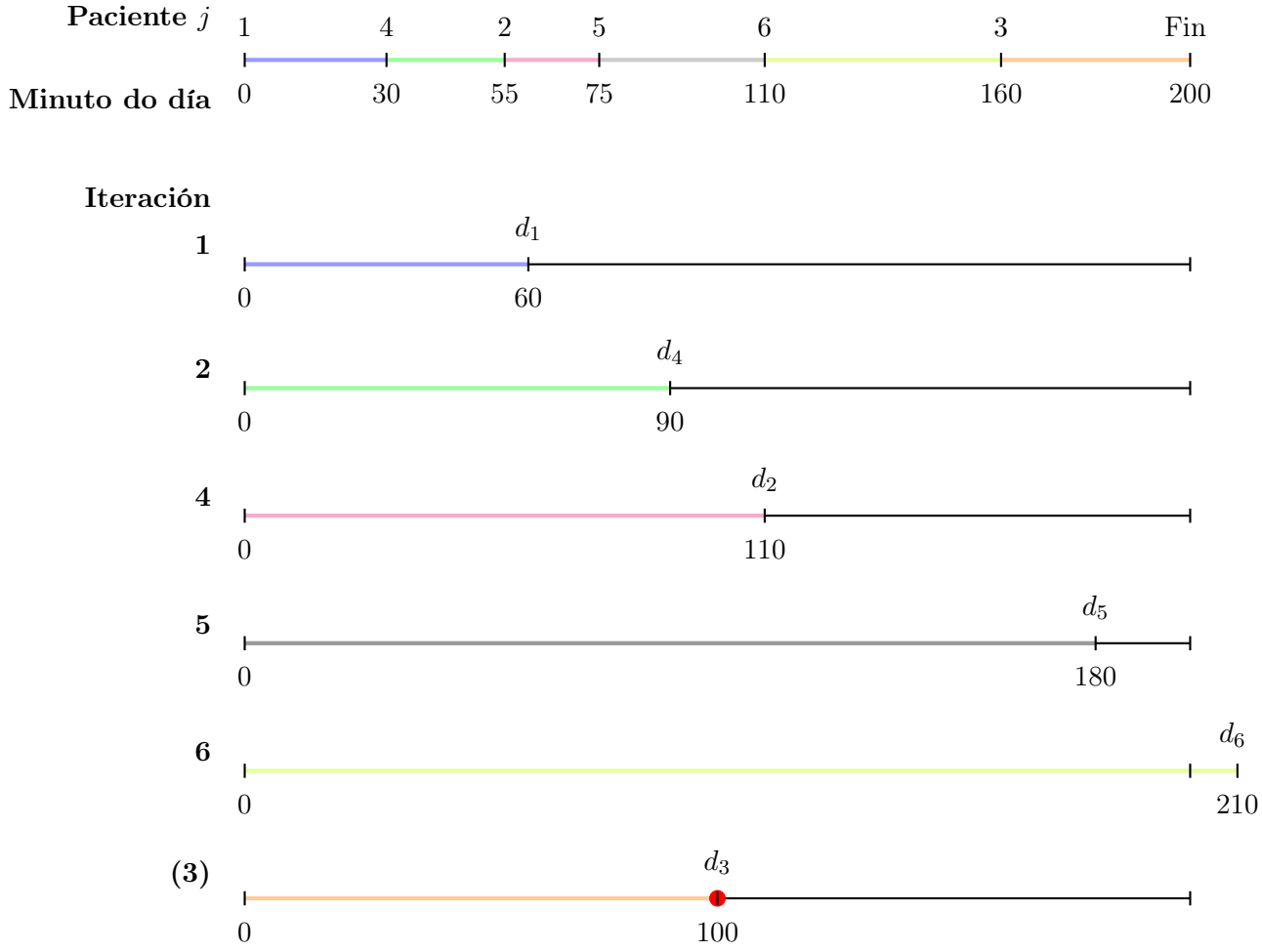
A quinta e a sexta iteración serán como as primeiras xa que se cumpren as desigualdades correspondentes. Para a quinta temos $J = \{1, 4, 2, 5\}$ e en consecuencia:

$$\left.\begin{array}{r} \sum_{j \in J} p_j = 30 + 25 + 20 + 35 = 110 \\ d_5 = 180 \end{array}\right\} \Rightarrow \sum_{j \in J} p_j \leq d_k, \text{ tomamos } J_5 = J \text{ e } k = 5 + 1 = 6.$$

E para a sexta e última iteración, con $J = \{1, 4, 2, 5, 6\}$, temos:

$$\left.\begin{array}{r} \sum_{j \in J} p_j = 30 + 25 + 20 + 35 + 50 = 160 \\ d_5 = 210 \end{array}\right\} \Rightarrow \sum_{j \in J} p_j \leq d_k, \text{ tomamos } J_6 = J \text{ e } k = 6.$$

Unha vez considerados todos os pacientes, rematamos co algoritmo. Os tratamentos iniciaranse coa secuencia de pacientes determinada no conxunto $J_6 = \{1, 4, 2, 5, 6\}$, que se corresponde coa secuencia de tratamentos que finalizarán a tempo na solución óptima. Por último, recibirá tratamento o paciente 3, que foi o único que a medida que avanzamos co algoritmo quedou descartado por ter un tempo de tratamento superior aos pacientes considerados ata o momento. Entón, a orde final dos tratamentos será: 1 - 4 - 2 - 5 - 6 - 3. Vexámola na seguinte liña temporal.



Desta maneira, é fácil ver que $h_{max} = \sum U_j(C_j) = 1$ pois dado que 3 é o único paciente que recibe o tratamento con retraso, $U_3(C_3) = 1$, mentres que para $i \in \{1, 2, 4, 5, 6\}$ temos que $U_i(C_i) = 0$.

Unha vez visto como funciona o algoritmo, consideremos unha xeralización dos problemas anteriores tendo en conta os pesos w_j, os problemas de horarios definidos pola terna $1|\,|\sum_j w_j U_j$. Estes novos problemas pertencen á categoría dos problemas *NP-hard* polo que o Algoritmo 3.6

deixa de calcular unha solución óptima. O caso concreto no que as datas límites para completar as tarefas (d_j) son iguais, é equivalente ao coñecido como problema da mochila, que como é sabido, é moi difícil de resolver. Atopamos a explicación correspondente a este problema en [5].

As analoxías que podemos facer entre os parámetros de cada problema son os seguintes:

- As d_j son equivalentes ao tamaño da mochila.
- Os p_j son equivalentes ao tamaño dos obxectos que temos como opcións para introducir na mochila.
- Os w_j son equivalentes aos beneficios que obtemos ao seleccionar o obxecto j para levar na mochila.

3.3. Retraso total

Nos problemas da sección anterior tiñamos como obxectivo minimizar o número de tarefas que finalizan con retraso, $\sum_j U_j(C_j)$. Isto pode levarnos a que haxa un certo número de tarefas que teñan un retraso demasiado grande como para que sexa aceptable, como sucedía no Exemplo 3.8. Tras aplicar o algoritmo correspondente, obtivemos unha secuencia na que tan só tiñamos un paciente que recibía o tratamento con retraso, o paciente 3. Agora ben, tendo en conta que $C_3 = \sum_{j=1}^{6} p_j = 200$ e observando que $d_3 = 100$, o retraso no tratamento deste paciente era de 100 minutos, unha cifra que deixa ver que a atención aos pacientes non está a ser a correcta.

Nesta sección buscaremos unha solución para este inconvinte co terceiro e último tipo de función de custo que imos estudar:

$$h_j(C_j) = T_j(C_j) = \text{máx}(C_j - d_j, 0) = \text{máx}(L_j, 0)$$

Nos problemas de horarios que imos considerar, o obxectivo a minimizar é $h_{max} = \sum_j T_j(C_j)$, a suma dos retrasos de todas as tarefas. A terna que describe estes problemas será $1|\ |\sum_j T_j(C_j)$.

Ao igual que os problemas da sección anterior, estes tamén están moi estudados na literatura, pois non foi ata 1990 que se incluíron na categoría dos problemas *NP-hard* como vemos en [2]. Ata ese momento descoñecíase cal era a súa complexidade. O algoritmo que presentaremos a continuación para resolver este tipo de problemas de horarios baséase nos seguintes tres lemas.

Lema 3.9. *Se $p_j \leq p_k$ e $d_j \leq d_k$, existe unha secuencia óptima na que a tarefa j secuénciase antes que a tarefa k.*

Para o seguinte lema consideramos dúas situacións diferentes, ambas con n tarefas con tempos de procesamento $p_1, ..., p_n$.

1. Para o primeiro caso temos as datas límites $d_1, ..., d_n$. Definimos por $\mathcal{S}'$ a secuencia óptima para este caso e denotamos por C'_k o tempo de finalización máis tardio posible para o traballo k nesta secuencia.

2. Para o segundo caso temos as datas límites $d_1, ..., d_{k-1}, \text{máx}(d_k, C'_k), d_{k+1}, ..., d_n$ e definimos por $\mathcal{S}''$ a secuencia óptima correspondente. Denotamos por C''_j o tempo de completado da tarefa j nesta secuencia.

Lema 3.10. *Calquera secuencia que sexa óptima para a situación 2, tamén será óptima para a situación 1.*

A partir deste punto asumimos que os tempos de procesamento das tarefas son todos diferentes e que $d_1 \leq ... \leq d_n$. Ademais, definimos a tarefa **k**, é dicir, a tarefa coa data límite d_k (a k-ésima máis pequena) como aquela que ten o maior tempo de procesamento, $p_k = \text{máx}(p_1, ..., p_n)$.

Polo Lema 3.9 podemos afirmar que existe unha secuencia óptima onde as tarefas $\{1, ..., k-1\}$ aparecen todas, nunha certa orde, antes que a tarefa k. Para as $n - k$ tarefas restantes, é dicir, para as tarefas $\{k+1, ..., n\}$ aparecerán algunhas antes e outras despois que a tarefa k. O seguinte lema céntrase nestas últimas $n - k$ tarefas.

Lema 3.11. *Existe un $\delta \in \mathbb{Z}$ tal que $0 \leq \delta \leq n - k$, de forma que existe unha secuencia óptima $\mathcal{S}$ na que a tarefa k está precedida por todas as tarefas j que verifiquen $j \leq k + \delta$ e seguida das tarefas j que verifiquen $j > k + \delta$.*

O algoritmo que resolve os problemas desta sección precisa dunha subrutina para xerar unha secuencia óptima para as tarefas $1, ..., \mathbf{l}$ comezando o seu procesamento en tempo igual a t. A tarefa k segue a verificar as mesmas condicións que antes, será aquela co maior tempo de procesado desas l tarefas ($p_k = \text{máx}(p_1, ..., p_l)$).

Apoiándonos no Lema 3.11 sabemos que existe un $\delta \in \mathbb{Z}$ tal que $0 \leq \delta \leq l - k$ de forma que existe unha secuencia óptima que comeza en t que pode considerarse como unha concatenación dos seguintes tres conxuntos de tarefas:

1. tarefas $1, 2, ..., k-1, k+1, ..., k-\delta$ secuenciadas nunha certa orde.

2. tarefa k.

3. tarefas $k+\delta+1, k+\delta+2, ..., l$, secuenciadas nunha certa orde.

O tempo de completado da tarefa k vén dado pola expresión $C_k(\delta) = \sum_{j \leq k+\delta} p_j$. É evidente que para obter unha secuencia óptima para o conxunto completo das tarefas, primeiro debemos atopar unha secuencia óptima para as tarefas do primeiro e terceiro conxunto.

Por último, antes de enunciar o algoritmo correspondente desta sección, definimos os seguintes conceptos:

- $J(j,l,k) = \{j, j+1, ..., l-1, l\}$, serán as tarefas cun tempo de procesamente menor que o asociado á tarefa k, p_k.
- $V(J(j,l,k),t)$ denota o retraso total do conxunto anterior nunha secuencia óptima, asumindo que comeza en tempo t.

Estamos en condicións de presentar o seguinte algoritmo.

Algoritmo 3.12. ***Minimización do retraso total.***

Inicialmente temos:

$$V(\emptyset, t) = 0$$
$$V(\{j\}, t) = \text{máx}(0, t + p_j - d_j)$$

De forma recursiva temos as relacións:

$$V(J(j,l,k),t) = \min_{\delta} \Big(V(J(j, k'+\delta, k'), t) + \text{máx}(0, C_{k'}(\delta) - d_{k'}) + V(J(k'+\delta+1, l, k'), C_{k'}(\delta)) \Big)$$

onde k' é a tarefa que cumpre $p_{k'} = max(\, p_{j'} \mid j' \in J(j,l,k))$.

Entón, o valor óptimo para a función obxectivo vén dado por

$$V(\{1, ..., n\}, 0).$$

Para comprender mellor o algoritmo, apliquémolo no seguinte exemplo continuando na liña dos tratamentos de radioterapia.

Exemplo 3.13. Problema de horarios para os tratamentos de radioterapia nunha única máquina, buscando minimizar o retraso total acumulado dos pacientes. Presentamos unha situación independente á dos exemplos tratados ata o de agora, pois imos variar o número de pacientes e os seus datos asociados. Malia iso, continuamos no contexto dos tratamentos de radioterapia cunha única máquina dispoñible e sen interrupcións entre pacientes.

Na seguinte táboa presentamos os datos médicos relativos ao tempo de duración do tratamento (p_j) e ao horario desexado para que o tratamento esté rematado (d_j) de cinco pacientes, é dicir, considerando $n = 5$. Este problema descríbese coa tripa $1|\ |\ \sum_j T_j(C_j)$.

Pacientes (j)	1	2	3	4	5
p_j	121	79	147	83	130
d_j	260	266	266	336	337

Comezamos o proceso identificando o que será o paciente **k**, é dicir, o que ten o maior tempo de tratamento, sendo, neste caso, o paciente $k = 3$ con $p_3 = 147$.

Desta maneira, acorde ao Lema 3.11, existe un δ tal que:

$$0 \leq \delta \leq n - k = 5 - 3 \Rightarrow 0 \leq \delta \leq 2,$$

é dicir, δ poderá tomar os valores 0, 1 e 2.

O obxectivo final é calcular $V(\{1,2,3,4,5\},0)$, é dicir, o retraso total dos tratamentos do conxunto de pacientes $\{1,2,3,4,5\}$, secuenciados de forma óptima, asumindo que comezan en $t = 0$.

Empregando a fórmula recursiva do Algoritmo 3.12, temos o seguinte:

$$V(\{1,2,3,4,5\},t) = \min_{\delta} \Big(V(J(j,k'+\delta,k'),t) + \max(0, C_{k'}(\delta) - d_{k'}) + V(J(k'+\delta+1,l,k'), C_{k'}(\delta)) \Big)$$

Destacamos que $k' = k = 3$, pois verifica a igualdade $p_{k'} = max(\, p_{j'} \mid j' \in \{1,2,3,4,5\}) = 147$.

Entón, obtemos a seguinte igualdade:

$$V(\{1,2,3,4,5\},0) = \min_{\delta} \Big(V(J(1,3+\delta,0),0) + \max(0, C_3(\delta) - d_3) + V(J(3+\delta+1,5,3), C_3(\delta)) \Big)$$

Vexamos cales son as relacións anteriores en función dos valores que pode tomar δ.

Sexa $\delta = 0$. Calculemos o seguinte:

- $V(J(1,3+0,3),0)$, onde $J(1,3,3) = \{1,2\}$ e $t = 0$. Vexamos cal é a secuencia óptima para este conxunto. Temos dúas posibilidades:
  - Primeiro caso: 1 - 2, cos correspondentes retrasos (R_j)

    $$\left.\begin{aligned} R_1 &= \max(0 + 121 - 260, 0) = 0 \\ R_2 &= \max(121 + 79 - 266, 0) = 0 \end{aligned}\right\} \Rightarrow R_1 + R_2 = 0 \text{ polo que non temos retraso.}$$

  - Segundo caso: 2 - 1, cos correspondentes retrasos

    $$\left.\begin{aligned} R_2 &= \max(0 + 79 - 266, 0) = 0 \\ R_1 &= \max(79 + 121 - 260, 0) = 0 \end{aligned}\right\} \Rightarrow R_2 + R_1 = 0 \text{ polo que non temos retraso.}$$

Entón, as dúas secuencias son óptimas e $V(J(1,3,3),0) = 0$

- máx$(0, C_3(0) - d_3)$, sendo

$$C_3(0) = \sum_{j \leq 3+0} p_j = p_1 + p_2 + p_3 = 121 + 79 + 147 = 347,$$

obtemos máx$(0, C_3(0) - d_3)$ = máx$(0, 347 - 266) = 81$

- $V(J(3+0+1,5,3), C_3(0))$, onde $J(4,5,3) = \{4,5\}$ e $C_3(0) = 347$. De forma análoga ao primeiro punto, vexamos cal é a secuencia óptima para este conxunto. Temos dúas posibilidades:

  - Primeiro caso: 4 - 5, cos correspondentes retrasos

$$\left.\begin{aligned} R_4 &= \text{máx}(347 + 83 - 336, 0) = 94 \\ R_5 &= \text{máx}(347 + 83 + 130 - 337, 0) = 223 \end{aligned}\right\} \Rightarrow R_4 + R_5 = 94 + 223 = 317$$

  - Segundo caso: 5 - 4, cos correspondentes retrasos

$$\left.\begin{aligned} R_5 &= \text{máx}(347 + 130 - 337, 0) = 140 \\ R_4 &= \text{máx}(347 + 130 + 83 - 336, 0) = 224 \end{aligned}\right\} \Rightarrow R_5 + R_4 = 140 + 224 = 364$$

  Entón, a secuencia óptima é 4 - 5 e $V(J(4,5,3),347) = 317$

Desta maneira, para as secuencias 1 - 2 - 3 - 4 - 5 e 2 - 1 -3 - 4 - 5 temos

$$V(J(1,3+0,3),0) + \text{máx}(0, C_3(0) - d_3) + V(J(4,5,3), C_3(0)) = 0 + 81 + 317 = 398.$$

Sexa $\delta = 1$. Analogamente a ao caso anterior mais de forma simplificada, calculemos os seguintes valores:

- $V(J(1,3+1,3),0)$, onde $J(1,4,3) = \{1,2,4\}$ e $t = 0$.

  Para as secuencias 1 - 2 - 4 e 2 - 1 - 4, que serán as secuencias óptimas, temos

$$V(J(1,4,3),0) = 0$$

- máx$(0, C_3(1) - d_3)$, sendo

$$C_3(1) = \sum_{j \leq 3+1} p_j = p_1 + p_2 + p_3 + p_4 = 121 + 79 + 147 + 83 = 430,$$

obtemos máx$(0, C_3(1) - d_3)$ = máx$(0, 430 - 266) = 164$

- $V(J(3+1+1,5,3),C_3(1))$, onde $J(5,5,3)=\{5\}$ e $C_3(1)=430$. Entón,

$$R_5 = \text{máx}(430+130-337,0) = 223$$

  Polo que $V(J(5,5,3),C_3(1)) = 223$

Logo, para as secuencias 1 - 2 - 4 - 3 - 5 e 2 - 1 - 4 - 3 - 5 temos

$$V(J(1,4,3),0) + \text{máx}(0,C_3(1)-d_3) + V(J(5,5,3),C_3(1)) = 0+164+223 = 387.$$

Sexa $\delta = 2$. Analogamente a aos casos anteriores calculemos os seguintes valores:

- $V(J(1,3+2,3),0)$, onde $J(1,5,3)=\{1,2,4,5\}$ e $t=0$.

  Para as secuencias 1 - 2 - 4 - 5 e 2 - 1 - 4 - 5, que serán as secuencias óptimas, temos

$$V(J(1,5,3),0) = 76$$

- $\text{máx}(0,C_3(2)-d_3)$, sendo

$$C_3(2) = \sum_{j\leq 3+2} p_j = p_1+p_2+p_3+p_4+p_5 = 121+79+147+83+130 = 560,$$

  obtemos $\text{máx}(0,C_3(2)-d_3) = \text{máx}(0,560-266) = 294$

- $V(J(3+2+1,5,3),C_3(2)) = V(\emptyset,C_3(2)) = 0$

Logo, para as secuencias 1 - 2 - 4 - 5 - 3 e 2 - 1 - 4 - 5 - 3 temos

$$V(J(1,5,3),0) + \text{máx}(0,C_3(2)-d_3) + V(\emptyset,C_3(2)) = 76+294+0 = 370.$$

Comparamos os resultados segundo os tres valores de δ:

$$V(\{1,2,3,4,5\},0) = \text{mín}\left\{\begin{array}{c} 0+81+317 \\ 0+164+223 \\ 76+294+0 \end{array}\right\} = 370$$

Finalmente, chegamos a que dúas secuencias diferentes serán óptimas cun retraso total asociado de 370 minutos, 1 - 2 - 4 - 5 - 3 e 2 - 1 - 4 - 5 - 3.

Vexamos entón cal é o valor da función de custo $h_{max} = \sum_j T_j(C_j) = \sum_j \text{máx}(L_j,0)$, respetando ditas secuencias. Como xa vimos que as dúas secuencias son óptimas, basta facer os cálculos para tan só unha delas. No seguinte cadro recóllense os datos necesarios para o seu cálculo.

Orde dos tratamentos	1	2	4	5	3
d_j	260	266	399	337	266
C_j	121	200	283	413	560
$L_j(C_j)$	-139	-66	-116	76	294
$T_j(C_j)$	0	0	0	76	294

Concluimos que o retraso total das tarefas será $h_{max} = \sum_j T_j(C_j) = 76 + 294 = 370$. Como era de esperar, o resultado coincide co xa calculado para $V(\{1, 2, 3, 4, 5\}, 0)$. Este exemplo non se pode comparar con ningún dos plantexados ata o de agora pois, non coinciden nin o número de pacientes nin os datos asociados a cada un destes.

Capítulo 4

Implementación dos Algoritmos

Nos capítulos anteriores, en cada sección presentouse un algoritmo concreto para resolver os diferentes tipos de problemas de horarios expostos, acompañado dun exemplo no que se aplicaba dito algoritmo. Neste apartado recóllese a implementación en código en R da maior parte destes algoritmos, así como os resultados obtidos tras aplicalos aos exemplos correspondentes.

4.1. Implementación do Algoritmo 2.5

```
#w = c() # vector prioridades
#p = c() # vector de tempos de procesamento

# índices das dúas cadeas
#c1 = c()
#c2 = c()

build_schedule_WSPT <- function(w, p, c1, c2) {
  n <- length(w)
  schedule <- c()
  iter <- 1

  while (length(schedule) < n) { # ata que se incluan todas as tarefas, repetimos
    cat("\nIteración", iter, "\n")

    # calculamos densidades acumuladas
    rhoc1 <- if (length(c1) > 0) cumsum(w[c1]) / cumsum(p[c1]) else -Inf
    rhoc2 <- if (length(c2) > 0) cumsum(w[c2]) / cumsum(p[c2]) else -Inf

    cat(" rhoc1:", if (is.finite(rhoc1[1])) round(rhoc1, 3) else "vacío", "\n")
```

```
    cat(" rhoc2:", if (is.finite(rhoc2[1])) round(rhoc2, 3) else "vacío", "\n")

    if (max(rhoc1) > max(rhoc2)) {
      #se rhoc1 ten valor max maior que rhoc2,
      #seleciónanse as k primeiras tarefas onde k éigual a:
      k <- which.max(rhoc1)
      cat(" Selecciono", k, "elemento(s) de c1:", c1[1:k], "\n")
      schedule <- c(schedule, c1[1:k]) # esas tarefas engádense ao horario
      c1 <- c1[-(1:k)] # elimínanse do conxunto
    } else {   # do contrario, faise o mesmo con c2
      k <- which.max(rhoc2)
      cat(" Selecciono", k, "elemento(s) de c2:", c2[1:k], "\n")
      schedule <- c(schedule, c2[1:k])
      c2 <- c2[-(1:k)]
    }

    cat(" Schedule actual:", schedule, "\n")
    iter <- iter + 1
  }

  cat("\nSchedule final:", schedule, "\n")
  C <- cumsum(p[schedule])
  custo<-sum(w[schedule]*C)
  cat("\nCusto final:", custo, "\n")
  return(schedule)
}
```

Apliquemos agora esta función aos datos do Exemplo 2.6 e observemos que obtemos o mesmo resultado que tiñamos.

```
w <- c(3, 1, 2, 4, 5, 2)
p <- c(30, 20, 40, 25, 35, 50)
c1 <- 1:4
c2 <- 5:6
schedule <- build_schedule_WSPT(w, p, c1, c2)

# saída
Iteración 1
  rhoc1: 0.1 0.08 0.067 0.087
  rhoc2: 0.143 0.082
  Selecciono 1 elemento(s) de c2: 5
  Schedule actual: 5
```

```
Iteración 2
  rhoc1: 0.1 0.08 0.067 0.087
  rhoc2: 0.04
  Selecciono 1 elemento(s) de c1: 1
  Schedule actual: 5 1

Iteración 3
  rhoc1: 0.05 0.05 0.082
  rhoc2: 0.04
  Selecciono 3 elemento(s) de c1: 2 3 4
  Schedule actual: 5 1 2 3 4

Iteración 4
  rhoc1: vacío
  rhoc2: 0.04
  Selecciono 1 elemento(s) de c2: 6
  Schedule actual: 5 1 2 3 4 6

Schedule final: 5 1 2 3 4 6

Custo final: 1705
```

4.2. Implementación do Algoritmo 2.7

```
#w = c()          # vector prioridades
#p = c()          # vector de tempos de procesamento
#r = c()          # vector con tempo de chegada
#d = c()          # vector con tempo desexado de finalización
#restante = c()   # vector con tempo restante (inicialmente igual a p)
#completado = NA # vector con tempo de finalización


build_schedule_WSPT_interrup <- function(tarefas){

  # inicializamos as variables necesarias
  t <- 0                    # tempo actual
  procesando <- NA          # tarefa que se está procesando actualmente
  schedule <- c()


  # bucle de simulación
```

```
while(any(tarefas$restante > 0)) {

  # filtramos as tarefas dispoñibles
  disponibles <- tarefas[tarefas$r <= t & tarefas$restante > 0, ]

  if (nrow(disponibles) > 0) {

    # creamos unha nova columna e calculamos a prioridade wj / tempo_restante
    disponibles$prioridad <- disponibles$w / disponibles$restante

    # escollemos a tarefa con maior prioridade
    mellor <- disponibles$id[which(disponibles$prioridad == max(disponibles$prioridad))]

    # en caso de que dous factores coincidan sendo máximos,
    # tomamos o que teña menor dj
    if(length(mellor)>1){
      mellor <-disponibles$id[which.min(disponibles$d)]
    }

    # procesamos por 1 unidade de tempo
    #(actualizamos o tempo restante 1 unidade na tarefa que está sendo procesada)
    tarefas$restante[tarefas$id == mellor] <- tarefas$restante[tarefas$id == mellor] - 1
    if(length(schedule)==0) {
    schedule <- c(schedule, mellor) #engadimos a tarefa con maior prioridade
    } else {
      if (schedule[length(schedule)]==mellor){ #para on ter un vector de 200 valores
        schedule <- c(schedule)
      } else {
        schedule <- c(schedule, mellor)
      }
    }

    # si se remata ca tarefa, gardamos o tempo de finalización (aumentando 1 unidade)
    if (tarefas$restante[tarefas$id == mellor] == 0) {
      tarefas$completado[tarefas$id == mellor] <- t + 1
    }

  } else {
    # non hai tarefas dispoñibles
    schedule <- c(schedule, NA) #non se engade ningunha tarefa
  }

  if (t %in% tarefas$r) {
```

```
      cat("\nPrioridade das tarefas:", disponibles$id, "\n")
      cat("                    ", round(disponibles$prioridad, 2), "\n")
    }


    t <- t + 1 #actualizamos o tempo
  }

  cat("\nSchedule final:", schedule, "\n") #mostramos a secuencia resultante

  # calculamos custo total ponderado:  wj * Cj
   tarefas$C <- tarefas$completado #calculamos o tempo total de finalización das tarefas
   tarefas$wC <- tarefas$w * tarefas$C
   custo <- sum(tarefas$wC) # calculamos o tempo total ponderado
   cat("Custo final:", custo, "\n")
   return(schedule)
}
```

Apliquemos agora esta función para os datos do Exemplo 2.8 e vexamos como obtemos a mesma secuencia que tiñamos.

```
tarefas <- data.frame(
   id = 1:6,               #tarefas
    w = c(3, 1, 2, 4, 5, 2),
    p = c(30, 20, 40, 25, 35, 50),
    r = c(0, 0, 5, 5, 30, 65),
    d = c(60, 110, 100, 90, 180, 210),
    restante = c(30, 20, 40, 25, 35, 50),
    completado = NA
  )
schedule <- build_schedule_WSPT_interrup(tarefas)

#Saída
Prioridade das tarefas: 1 2
                    0.1 0.05

Prioridade das tarefas: 1 2 3 4
                    0.12 0.05 0.05 0.16

Prioridade das tarefas: 1 2 3 5
                    0.12 0.05 0.05 0.14

Prioridade das tarefas: 1 2 3 6
```

```
                    0.12 0.05 0.05 0.04

Schedule final: 1 4 5 1 3 2 6
Custo final: 1525
```

4.3. Implementación do Algoritmo 3.1

```
#tarefas <- data.frame(
#  id =1:n
#  p=c()
#)
#   data frame con 2 columnas: as tarefas e os seus tempos de procesamento

#h_funcs <- list(
#  "1" = function(C) ...,
#  "n" = function(C) ...,
#)
#   lista coas funcións de custo asociadas a cada tarefa

build_sequence_min_hmax_func <- function(tarefas, h_funcs) {
  J <- c()              # secuencia construida (de atrás cara adiante)
  Jc <- tarefas$id      # tarefas pendentes de ser secuenciadas
  pks <- setNames(tarefas$p, tarefas$id) # asociamos o nome da tarefa ao seu pj

  while (length(Jc) > 0) {
    total_p <- sum(pks[Jc]) # tempo total restante (sumatorio de pj en Jc)
    J_prime <- Jc          # todas as pendentes son candidatas para ser secuenciadas

    # calculamos o custo h_j(C_j) para cada j, con C_j = total_p
    custo <- sapply(J_prime, function(j) {
      h_func <- h_funcs[[as.character(j)]]
      h_func(total_p)
    })

    # eleximos a tarefa con menor custo
    j_min <- J_prime[which.min(custo)]
    cat(" A tarefa", j_min, "ten o menor custo:", min(custo), "\n")

    # engadímola ao final da secuencia (construcción cara atrás)
    J <- c(j_min,J)
```

```
    # eliminámola das tarefas pendentes
    Jc <- setdiff(Jc, j_min) #devolve os elementos diferentes entre ambos vectores
  }

  cat("\nSchedule final:", J, "\n")
  return(J)
}
```

Apliquemos agora esta función para os datos do Exemplo 3.3 e vexamos como obtemos a mesma secuencia que tiñamos.

```
tarefas <- data.frame(
  id = 1:3,
  p = c(30, 20, 40)
)
h_funcs <- list(
  "1" = function(C) C + 3,
  "2" = function(C) 1.2 * C,
  "3" = function(C) 10
)
schedule <- build_sequence_min_hmax_func(tarefas, h_funcs)

#Saída
  A tarefa 3 ten o menor custo: 10
  A tarefa 1 ten o menor custo: 53
  A tarefa 2 ten o menor custo: 24

Schedule final: 2 1 3
```

4.4. Implementación do Algoritmo 3.6

```
min_delay_tasks <- function(tarefas) {
  # primeiro ordenamos as tarefas pola súa data límite (dj)
  tarefas <- tarefas[order(tarefas$d), ]

  J <- c()          # conxunto de tarefas que cumpren a data límite
  Jc <- tarefas$id # conxunto de tarefas pendientes (inicialmente todas)
  Jd <- c()         # conxunto de tarefas con retraso (descartadas)

  k <- 1 # inicializamos para comezar pola primeira fila de tarefas
         # (é dicir, a que teña menor dj pois xa están ordenadas)
```

```
  while (k <= nrow(tarefas)) {

    # Paso 2: engadimos a tarefa k a J e quitámola de Jc
    J <- c(J, tarefas$id[k])
    Jc <- setdiff(Jc, tarefas$id[k])

    # suma de tempos de procesamento en J
    sum_pj <- sum(tarefas$p[match(J, tarefas$id)])
              #empregamos a función match pois no dataframe: tarefas, témolas
              #ordenadas polas dj

    # Paso 3: comprobamos si sum_pj <= d_k, de nun cumprirse, facemos o seguinte:
    if (sum_pj > tarefas$d[k]) {

      # buscamos a tarefa con maior tempo de procesamento en J
      pjs <- tarefas$p[match(J, tarefas$id)]
          # vector soamente cos tempos de procesamento das tarefas en J
      l_idx <- which.max(pjs)
      # definimos a tarefa co maior valor por l
      l <- J[l_idx]

      # quitamos tarefa l de J e engadímola a Jd (descartámola)
      J <- J[-l_idx]
      Jd <- c(Jd, l)
    }

    # Paso 4:
    if (k == nrow(tarefas)) {
      break
    } else {
      k <- k + 1
    }
  }

  cat("Tarefas a tempo:", J, "\n")
  cat("Tarefas con retraso:", Jd, "\n")

  schedule<-c(J,Jd)
  cat("\nSchedule final:", schedule, "\n")

  cat("\nNúmero de tarefas con retraso:", length(Jd), "\n")
}
```

Apliquemos agora esta función para os datos do Exemplo 3.8 e vexamos como obtemos a mesma secuencia que tiñamos.

```
tarefas <- data.frame(
  id = 1:6,
  p = c(30, 20, 40, 25, 35, 50),
  d = c(60, 110, 100, 90, 180, 210)
)

resultado <- min_delay_tasks(tarefas)

#Saída
Tarefas a tempo: 1 4 2 5 6
Tarefas con retraso: 3

Schedule final: 1 4 2 5 6 3

Número de tarefas con retraso: 1
```

Capítulo 5

Conclusións

Unha vez finalizada a revisión dos diferentes problemas de horarios e as súas aplicacións, cómpre destacar a relevancia do seu estudo e o desenvolvemento de algoritmos específicos para a súa resolución. Tal e como se amosou ao longo dos distintos exemplos analizados, atopar solucións óptimas permite optimizar o uso do tempo dispoñible para asignar recursos limitados a un conxunto de tarefas, o que constitúe unha necesidade fundamental tanto na vida cotiá como no ámbito profesional.

Os exemplos propostos ao longo do traballo estiveron centrados no contexto do servizo sanitario, especialmente na planificación de tratamentos de radioterapia. Con esta elección pretendeuse poñer en valor a importancia da aplicación das matemáticas neste ámbito, así como o potencial que ofrecen para mellorar a calidade dos servizos prestados aos pacientes.

Así mesmo, destacamos que as situacións formuladas foron simplificadas intencionadamente para facilitar a súa análise aplicando os contidos teóricos nos que centramos o noso estudo. Porén, os problemas plantexados poden considerarse subproblemas de modelos máis complexos e realistas, nos que estas técnicas tamén poden ser de utilidade. Isto evidencia a versatilidade e aplicabilidade das ferramentas matemáticas na resolución de problemas reais e, en particular, no ámbito sanitario.

Bibliografía

[1] K. R. Baker and D. Trietsch, *Principles of Sequencing and Scheduling*, John Wiley & Sons, 2018.

[2] P. Brucker, *Scheduling Algorithms*, Chapter 4, Springer, 1999.

[3] C. A. Floudas and X. Lin, *Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications*, Annals of Operations Research, vol. 139, pp. 131–162, 2005.

[4] M. L. Pinedo, *Single Machine Models (Deterministic)*, in *Scheduling: Theory, Algorithms, and Systems*, pp. 35–67, Springer, 2012.

[5] H. M. Salkin and C. A. De Kluyver, *The knapsack problem: A survey*, Naval Research Logistics Quarterly, vol. 22, no. 1, pp. 127–144, 1975.