Departamento de Informática
Universidad Carlos III de Madrid

Grado en Informática

Heurística y Optimización

26 de Junio de 2017

Normas generales del examen

① El tiempo para realizar el examen es de **4 horas**

② No se responderá a ninguna pregunta sobre el examen transcurridos los primeros **30 minutos**

③ **Justifica debidamente todas tus respuestas**

④ Cada pregunta debe responderse en páginas separadas en el mismo orden de sus apartados. Si no se responde, se debe entregar una página en blanco

⑤ Numera todas las páginas de cada ejercicio separadamente

⑥ Escribe con claridad y en limpio, de forma ordenada y concisa

⑦ Si se sale del aula, no se podrá volver a entrar durante el examen

⑧ No se puede presentar el examen escrito a lápiz

Pregunta 1 (1½ puntos)

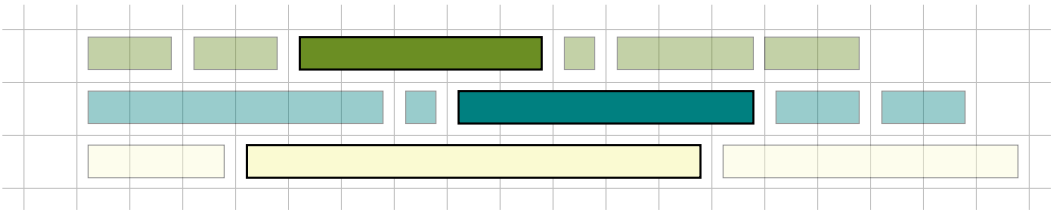
Un sistema automático de distribución de carga de trabajo recibe un conjunto de n trabajos, $\mathcal{W} = \{T_i\}_{i=1}^{n}$, para cada uno de los cuales se sabe con precisión el tiempo, t_i, que se requiere para su ejecución. y el beneficio, p_i, que resulta de completarlo. Diferentes restricciones físicas obligan al sistema de distribución de cargas a elegir un conjunto de trabajos tal que su suma de tiempos sea mayor o igual que un umbral L, e inferior o igual a un umbral máximo, U.

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ puntos)** Modelizar el enunciado como un problema de *Programación Lineal*, sabiendo que se desea *maximizar* el beneficio total.

Indica claramente el significado de las variables de decisión elegidas, y el significado de cada restricción y de la función objetivo.

En otro caso, los trabajos, T_i, se dividen en varias categorías, de tal modo que cada trabajo pertenece a una y sólo una categoría. La siguiente figura muestra esquemáticamente la división de trabajos en categorías, donde la longitud de cada trabajo representa el tiempo requerido para completarlo, y cada fila se corresponde con una categoría diferente. Los segmentos resaltados representan una solución factible.

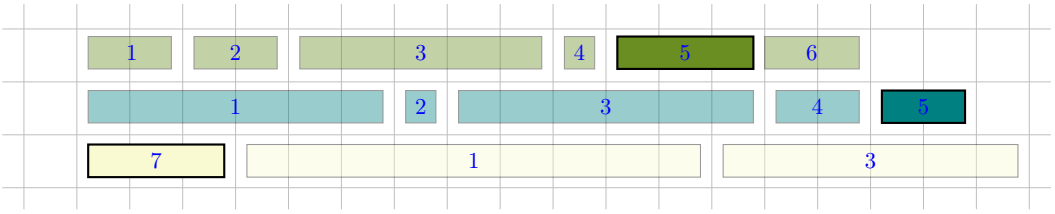


Se pide responder razonadamente la siguiente pregunta:

(b) **(½ puntos)** Modelizar el nuevo caso como un problema de *Programación Lineal*, en el que ahora sólo se puede escoger un trabajo de cada categoría como mucho.

Si fuera preciso hacer alguna modificación detállala claramente, incluso reformulando el modelo si fuera preciso.

Por último, se desea considerar que existen *grupos complementarios* de trabajos. Cada grupo contiene una colección arbitraria de trabajos de diferentes categorías, y si se escoge un trabajo de un grupo complementario, entonces es obligatorio escoger todos los del mismo grupo. La siguiente figura muestra diferentes trabajos en el mismo grupo con un mismo número. Los segmentos resaltados representan una solución factible.

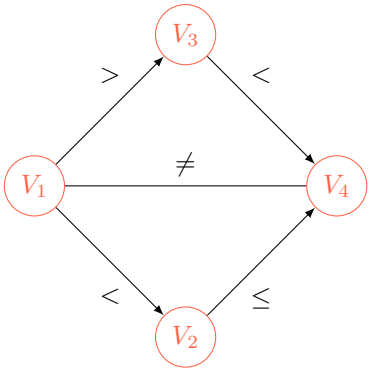


(c) **(½ puntos)** Modelizar el nuevo caso como un problema de *Programación Lineal*, en el que ahora sólo se puede escoger un trabajo de cada categoría como mucho, y todos los trabajos del mismo grupo.

Si fuera preciso hacer alguna modificación detállala claramente, incluso reformulando el modelo si fuera preciso.

Pregunta 2 (2 puntos)

Un sistema de cuatro válvulas eléctricas está conectado como se muestra en la siguiente figura:



Se quiere determinar el instante de activación, t_i, de cada válvula, V_i, *en uno y sólo uno* de los tres instantes de tiempo $t_i = \{1, 2, 3\}$, tal que se satisfagan todas las restricciones indicadas en la figura anterior, donde:

- $V_i > V_j$ significa que el instante de activación de la válvula i-ésima es posterior al de la válvula j-ésima.
- $V_i < V_j$ significa que el instante de activación de la válvula i-ésima es anterior al de la válvula j-ésima.
- $V_i \leq V_j$ significa que el instante de activación de la válvula i-ésima es anterior o igual al de la válvula j-ésima.
- $V_i \neq V_j$ significa que el instante de activación de la válvula i-ésima es diferente del de la válvula j-ésima.

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ punto)** Modelizar el problema propuesto como un problema de *satisfabilidad lógica*.
Indica claramente el significado de cada proposición utilizada, y de todas las cláusulas resultantes.

(b) **(½ puntos)** Modelizar el problema propuesto como un problema de *satisfacción de restricciones*.
Indica claramente el significado y propósito de cada variable y todas las restricciones que haya entre ellas.

Ahora, empleando únicamente la modelización del segundo apartado, se pide responder razonadamente a las siguientes preguntas:

(c) **(½ puntos)** Determinar si la variable que representa el instante de activación de la primera válvula, V_1 es o no *arco-consistente* con el instante de activación de la segunda válvula, V_2. ¿Si o no? ¿Es necesario hacer algún cambio para que lo sean?

(d) **(½ puntos)** Determinar si la variable que representa el instante de activación de la primera válvula, V_1 es o no *camino-consistente* con el instante de activación de la tercera válvula, V_3, respecto del de la cuarta válvula, V_4. ¿Si o no? ¿Es necesario hacer algún cambio para que lo sean?

Pregunta 3 (3 puntos)

Considerése el siguiente problema de Programación Lineal:

$$\begin{array}{rrcrcr} & \text{máx}\, z = 2x_1 + 3x_2 & & & & \\ - & 2x_1 & + & 5x_2 & \geq & 3 \\ - & x_1 & + & x_2 & \leq & 0 \\ & x_1 & + & 2x_2 & \leq & 12 \\ & & & \mathbf{x} \geq \mathbf{0} & & \end{array}$$

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ puntos)** Resolver el problema utilizando el método de *resolución gráfica*

Considérese ahora, en su lugar, la función objetivo $z = x_1^2 + x_2^2$. Se pide responder razonadamente la siguiente pregunta:

(b) **(½ puntos)** ¿Es posible resolver el mismo problema con la nueva función objetivo utilizando SIMPLEX? ¿Si o no y por qué?

En lo sucesivo, considérese que la función objetivo es $z = 0$. Se pide responder razonadamente las siguientes preguntas:

(c) **(½ puntos)** Expresar esta tarea de Programación Lineal en forma *estándar* de maximización de modo que, además, sea posible iniciar la aplicación del algoritmo SIMPLEX con una base igual a la matriz identidad.

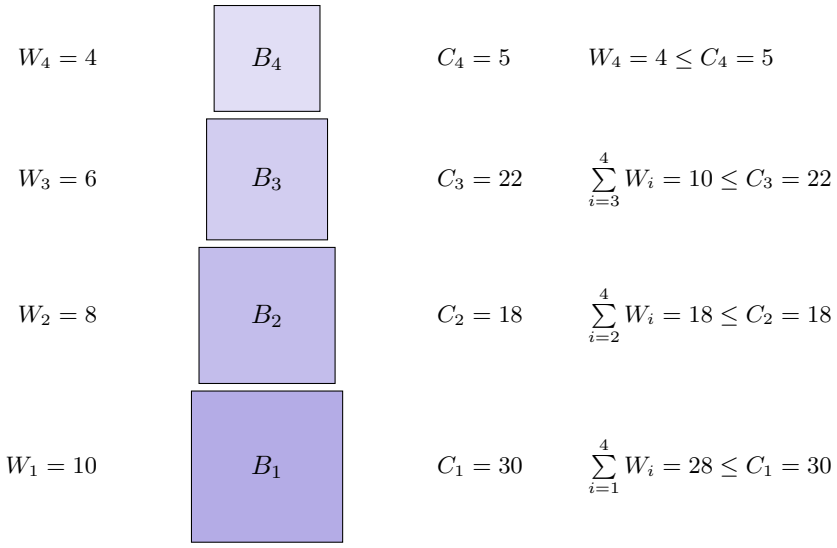
(d) **(½ puntos)** Resolver el problema de Programación Lineal obtenido en el apartado anterior con el algoritmo SIMPLEX.
Es imprenscindible indicar claramente, en cada iteración: las variables escogidas en la base, su valor, y el valor de la función objetivo

(e) **(½ puntos)** Interpretar las soluciones hayadas y explicar qué conclusiones pueden extraerse.

(f) **(½ puntos)** Calcula la contribución por unidad de recurso al crecimiento de la función objetivo.

Pregunta 4 (3½ puntos)

Se desea resolver *óptimamente* el problema de apilar verticalmente la mayor cantidad de cajas posibles entre una colección de ellas $\mathcal{S} = \{B_i\}_{i=1}^n$. Todas las cajas, B_i, son indistinguibles, salvo porque tiene cada una un peso W_i, y puede soportar una carga diferente, C_i, sumando el peso de todas las que estén encima de ella, más el suyo propio. La siguiente figura muestra una disposición legal de cajas:



El peso y capacidad de cada caja se muestran a la izquierda y derecha de cada una de ellas respectivamente. La columna derecha demuestra que cada caja puede efectivamente soportar la carga impuesta por esta configuración en particular.

Dado un conjunto arbitrario de cajas $\mathcal{S} = \{B_i\}_{i=1}^{n}$, caracterizadas por su peso, W_i, y capacidad, C_i, se pide *maximizar* el número de cajas que pueden apilarse verticalmente.

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ puntos)** Representar el problema como un *espacio de estados*

(b) **(½ puntos)** ¿Cuál es el tamaño del espacio de estados?

(c) **(½ puntos)** ¿Qué profundidad tendría un árbol de búsqueda desarrollado por un algoritmo de búsqueda de fuerza bruta para resolver óptimamente este problema? ¿Y cuál sería su factor de ramificación?

(d) **(½ puntos)** Habida cuenta de que queremos encontrar soluciones óptimas, ¿qué algoritmo de búsqueda *no informada* sugerirías para su resolución?

(e) **(1 punto)** Diseñar una función heurística $h(n)$ que sea *admisible* y que esté bien informada.

(f) **(½ puntos)** Suponiendo que se dispone de una función heurística $h(n)$ admisible, ¿qué algoritmo de búsqueda *heurística* es el más indicado para resolver este problema óptimamente? ¿Por qué?

Soluciones del examen de Heurística y Optimización Junio 2017

Problema 1

1. El enunciado de la primera parte es muy parecido al *problema de la mochila*, en el que la única diferencia es que además debe ocuparse una cantidad mínima de tiempo.

   Como en cualquier problema de modelización de *Programación Lineal*, los pasos consisten en determinar las *variables de decisión*, la representación de las *restricciones* y, por último, de la *función objetivo*:

   Variables de decisión Para la modelización que sigue se sugiere el uso de variables que determinen qué trabajos se seleccionan:

   $$x_i = \begin{cases} 1 & \text{si el trabajo } i\text{-ésimo es seleccionado} \\ 0 & \text{en caso contrario} \end{cases}$$

   Puesto que todas las variables de decisión sólo pueden escoger uno entre los valores $\{0, 1\}$, se trata de un problema de *Programación Lineal Entera Pura* con variables *binarias*.

   Restricciones El problema tiene dos restricciones:

   a) La primera es la misma que también se conoce en el *problema de la mochila*, y consiste en que no debe excederse un máximo de ocupación, U —en el caso del problema, se trata de ocupación *temporal*, en vez de capacidad *volumétrica*, pero ambos representan esencialmente el mismo concepto.

   $$\sum_{i=1}^{N} x_i t_i \leq U$$

   donde U es el umbral máximo indicado en el enunciado.

   b) Además, se pide ocupar una cantidad mínima de tiempo, L:

   $$\sum_{i=1}^{N} x_i t_i \geq L$$

   Como de costumbre, debe añadirse que las variables de decisión tomen valores no negativos pero, en este caso se exigirá que sean binarias:

   $$x_i \in \{0, 1\}$$

   Función objetivo Por último, el objetivo del problema consiste en maximizar el beneficio que resulta de la ejecución de los trabajos escogidos. Si x_i representa la selección del trabajo i-ésimo, entonces $x_i p_i$ es el beneficio que resulta de ejecutar el mismo trabajo —porque si $x_i = 0$ entonces no se produce beneficio, y si $x_i = 1$, entonces se obtiene un beneficio igual a $x_i p_i = p_i$. Razonando de la misma manera para la colección completa de trabajos resulta la siguiente función objetivo:

   $$\text{máx}\, z = \sum_{i=1}^{N} x_i p_i$$

   Por lo tanto, el modelo queda como sigue:

   $$\begin{aligned} \text{máx}\, z &= \textstyle\sum_{i=1}^{N} x_i p_i \\ \textstyle\sum_{i=1}^{N} x_i t_i &\leq U \\ \textstyle\sum_{i=1}^{N} x_i t_i &\geq L \\ x_i &\in \{0, 1\} \end{aligned}$$

2. El segundo apartado ejercita la representación de relaciones *exclusivas* en la que sólo una de varias selecciones debe ser posible.

   Formalmente hablando, sea $\{C_j\}_{j=1}^{M}$ la colección de categorías que se hayan definido, donde $M \leq N$ puesto que una tarea puede pertenecer a una y sólo una categoría —y, por lo tanto, no puede haber más categorías que trabajos. Por conveniencia de representación, las variables de decisión definidas en el apartado anterior se notarán ahora como $x_i^{(j}$ si y sólo si $x_i \in C_j$.

   Como cada variable de decisión sólo puede tomar los valores 0 ó 1, entonces sólo pueden darse dos casos:

   a) Ningún trabajo de una misma categoría es seleccionado, en cuyo caso la suma de sus variables de decisión será 0:

   $$\sum_{i=1}^{N} x_i^{(j} = 0, \quad \forall x_i^{(j} \in C_j$$

   b) El trabajo k-ésimo, de entre todos los de la misma categoría, C_j, es escogido. En ese caso, su variable de decisión, $x_k^{(j}$ tomará el valor 1, y ahora debe asegurarse que el resto tomarán el valor 0:

   $$\sum_{i=1}^{N} x_i^{(j} = 0, \quad \forall x_i^{(j} \in C_j, i \neq k$$

   Para componer ambos casos en una única restricción basta con observar que cualquier tarea i-ésima, o es ignorada (primer caso), $x_i^{(j} = 0$, o es elegida (segundo caso), $x_i^{(j} = 1$. Por lo tanto:

   $$\sum_{i=1}^{N} x_i^{(j} \leq 1, \quad \forall x_i^{(j} \in C_j$$

   de modo que la tarea de Programación Lineal resultante es:

   $$\begin{aligned} \text{máx}\, z = \textstyle\sum_{i=1}^{N} x_i^{(j} p_i & \\ \textstyle\sum_{i=1}^{N} x_i^{(j} t_i &\leq U \\ \textstyle\sum_{i=1}^{N} x_i^{(j} t_i &\geq L \\ \textstyle\sum_{i=1}^{N} x_i^{(j} &\leq 1 \quad \forall x_i^{(j} \in C_j \\ x_i^{(j} &\in \{0, 1\} \end{aligned}$$

3. Por último, el tercer apartado pedía ejercitar relaciones *condicionales*, donde todas y cada una de las variables de un mismo grupo deben ser ciertas al mismo tiempo *si y sólo si* se hace cierta una cualquiera de éllas.

   Formalmente, sea $\{G_k\}_{k=1}^{Z}$ el conjunto de grupos complementarios que se han definido, donde $Z \leq N$, puesto que cada trabajo puede pertenecer a uno y sólo un grupo. Además, se extiende la notación introducida en el apartado anterior haciendo que $x_i^{(j,k}$ represente la misma variable de decisión del primer apartado, x_i, donde simplemente se recuerda que pertenece a la categoría j-ésima, y al grupo k-ésimo.

   Para representar la relación condicional del enunciado, se procede de la siguiente manera:

a) Primero, se introduce una variable binaria b_k por cada grupo G_k, $b_k \in \{0,1\}$, y que deberá tomar el valor 1 tan pronto como SIMPLEX escoja uno cualquiera de los trabajos del mismo grupo G_k:

$$x_i^{(j,k} \leq b_k, \quad \forall x_i^{(j,k} \in G_k$$

Puesto que la variable binaria, b_k sólo puede tomar los valores 0 ó 1, en cuanto un trabajo cualquiera del mismo grupo sea escogido, b_k tomará necesariamente el valor 1.

Sin embargo, si el evento i-ésimo no es escogido, $x_i^{(j,k} = 0$, la restricción anterior permitiría que b_k tome el valor 1. Este efecto no deseado será también considerado en el siguiente paso.

b) Segundo, si algún trabajo del grupo k-ésimo ha sido escogido, entonces las restricciones anteriores harán $b_k = 1$, y ahora es preciso fozar la selección del resto de eventos del mismo grupo:

$$\sum_{i=1}^{N} x_i^{(j,k} \geq b_k |G_k|, \quad \forall x_i^{(j,k} \in G_k$$

donde $|G_k|$ representa el número de trabajos que pertenecen al grupo complementario k-ésimo.

Nótese que este segundo grupo de restricciones sirve, al mismo tiempo, para evitar el efecto no deseado del primer grupo de restricciones: si ningún trabajo del grupo complementario k-ésimo es escogido, entonces el primer grupo de restricciones permitirían que b_k tome el valor 1, pero entonces no se verificarían las restricciones de este segundo grupo, porque entonces todas las tareas del mismo grupo deberían escogerse.

En conclusión, la tarea de Programación Lineal resultante es:

$$\begin{array}{rcll} \text{máx } z = \sum_{i=1}^{N} x_i^{(j,k} p_i & & & \\ \sum_{i=1}^{N} x_i^{(j,k} t_i & \leq & U & \\ \sum_{i=1}^{N} x_i^{(j,k} t_i & \geq & L & \\ \sum_{i=1}^{N} x_i^{(j,k} & \leq & 1 & \forall x_i^{(j,k} \in C_j \\ x_i^{(j,k} & \leq & b_k & \forall x_i^{(j,k} \in G_k \\ \sum_{i=1}^{N} x_i^{(j,k} & \geq & b_k |G_k| & \forall x_i^{(j,k} \in G_k \\ x_i^{(j,k} \in \{0,1\}, b_k \in \{0,1\} & & & \end{array}$$

Problema 2

1. La modelización de un problema con *Lógica Proposicional* se hace distinguiendo proposiciones (que se definen como unidades indivisibles o atómicas de información), a las que se les asignan variables (que entonces pueden afirmarse o negarse, formando así los denominados *literales*), que pueden relacionarse con el uso de conectivas lógicas —que son la disyunción ($\vee$), conjunción ($\wedge$), o la negación.

   En el caso del enunciado propuesto, se distingue como unidad elemental de información que la válvula i-ésima esté activa, o no, en un instante determinado. Por lo tanto, se define:

   $$p_{ij} \leftrightharpoons \text{la válvula } i\text{-ésima está activada en el instante } j$$

   Con la que ahora pueden formarse literales que digan, por ejemplo, que la primera válvula está activa en el instante de tiempo 1, p_{11}, o que la segunda válvula no está activa en el tercer instante, $\overline{p}_{23}$.

   A continuación se propone el uso de fórmulas bien formadas de la lógica proposicional que representen las restricciones del enunciado.

$V_1 < V_2$ Significa que si la primera válvula se activa en el instante i, p_{1i}, entonces la segunda sólo podrá hacerlo en alguno de los instantes posteriores, $\{i+1, \ldots, 3\}$. Por lo tanto, no tiene sentido considerar la activación de la primera válvula en el tercer instante.

Si la primera válvula se activa en el primer instante de tiempo, p_{11}, entonces la segunda deberá hacerlo necesariamente en alguno de los dos últimos, $(p_{22} \vee p_{23})$:

$$p_{11} \rightarrow (p_{22} \vee p_{23})$$

Análogamente, si la primera válvula se activa en el segundo instante de tiempo, entonces la segunda debe activarse por necesidad en el último instante disponible:

$$p_{12} \rightarrow p_{23}$$

Puesto que $p \rightarrow q$ es lógicamente equivalente a $\overline{p} \vee q$, entonces las dos implicaciones anteriores se pueden reescribir como:

$$\begin{array}{l} \overline{p}_{11} \vee (p_{22} \vee p_{23}) \\ \overline{p}_{12} \vee \quad p_{23} \end{array}$$

$V_1 > V_3$ tiene el significado contrario al del caso anterior. Si la primera variable se activa en el instante i, p_{1i}, entonces la segunda sólo podrá hacerlo en alguno de los instantes anteriores, $\{1, \ldots, i-1\}$. Consecuentemente, no tiene sentido considerar la activación de la primera válvula en el primer instante de tiempo.

Si la primera válvula se activa en $t = 2$, entonces la tercera sólo podría hacerlo en el primero:

$$p_{12} \rightarrow p_{31}$$

Y, si se activara en $t = 3$, entonces la tercera puede hacerlo en cualquiera de los dos primeros instantes de tiempo:

$$p_{13} \rightarrow (p_{31} \vee p_{32})$$

que, como antes, producirán directamente cláusulas (i. e., disyunciones de literales) después de aplicar la equivalencia lógica de la implicación:

$$\begin{array}{l} \overline{p}_{12} \vee \quad p_{31} \\ \overline{p}_{13} \vee (p_{31} \vee p_{32}) \end{array}$$

$V_1 \neq V_4$ Significa que la primera y cuarta válvula, deben activarse necesariamente en instantes de tiempo diferentes. Esquemáticamente:

$$p_{1i} \rightarrow \bigvee_{t=1}^{3} p_{4t}, \quad i \neq t$$

que, para los distintos instantes de activación de la primera válvula, daría lugar a las siguientes relaciones:

$$\begin{array}{l} p_{11} \rightarrow (p_{42} \vee p_{43}) \\ p_{12} \rightarrow (p_{41} \vee p_{43}) \\ p_{13} \rightarrow (p_{41} \vee p_{42}) \end{array}$$

Y éstas se reescriben como cláusulas como sigue:

$$\begin{array}{l} \overline{p}_{11} \vee (p_{42} \vee p_{43}) \\ \overline{p}_{12} \vee (p_{41} \vee p_{43}) \\ \overline{p}_{13} \vee (p_{41} \vee p_{42}) \end{array}$$

$V_2 \leq V_4$ Es un caso muy similar al primero donde, únicamente, es preciso tener en cuenta el caso de la igualdad. En términos generales, si la segunda válvula se activa en el instante i, entonces la cuarta debe hacerlo en alguno de los instantes $\{i, i+1, \ldots, 3\}$

$$p_{2i} \rightarrow \bigvee_{t=i}^{3} p_{4t}, \quad i \neq j$$

que dará lugar a las siguientes implicaciones:

$$\begin{array}{rcl} p_{21} & \rightarrow & (p_{41} \vee p_{42} \vee p_{43}) \\ p_{22} & \rightarrow & (p_{42} \vee p_{43}) \\ p_{23} & \rightarrow & p_{43} \end{array}$$

que producen las siguientes cláusulas que deben sumarse al modelo propuesto:

$$\begin{array}{rl} \overline{p}_{21} \vee & (p_{42} \vee p_{42} \vee p_{43}) \\ \overline{p}_{22} \vee & (p_{42} \vee p_{43}) \\ \overline{p}_{23} \vee & p_{43} \end{array}$$

$V_3 < V_4$ Con un significado análogo al del primer caso, $V_1 < V_2$, y que se resuelve de la misma manera. A continuación se indican directamente las cláusulas resultantes:

$$\begin{array}{rl} \overline{p}_{31} \vee & (p_{42} \vee p_{43}) \\ \overline{p}_{32} \vee & p_{43} \end{array}$$

En total, se han generado hasta 12 cláusulas.

Ahora bien, aún existe otra restricción que no está modelizada en ninguna de las relaciones anteriores, y ésta es que cada válvula puede activarse en *un instante y sólo uno*. Esto es, si la válvula i-ésima se activa en el instante t, entonces la misma válvula debe estar desactivada en el resto de instantes:

$$p_{it} \rightarrow \bigwedge_{j=1}^{3} \overline{p}_{ij}, \quad j \neq t$$

Por ejemplo, si la primera válvula se activa en $t = 1$, p_{11}, entonces la regla anterior se instancia como sigue:

$$p_{11} \rightarrow (\overline{p}_{12} \wedge \overline{p}_{13})$$

que produce la siguiente expresión en lógica proposicional:

$$\overline{p}_{11} \vee (\overline{p}_{12} \vee \overline{p}_{13})$$

Aplicando ahora la ley distributiva de la lógica proposicional, la expresión anterior es equivalente a las dos cláusulas siguientes:

$$(\overline{p}_{11} \vee \overline{p}_{12}) \wedge (\overline{p}_{11} \vee \overline{p}_{13})$$

Razonando de la misma manera, si la primera válvula se activa en el segundo instante de tiempo, resultará:

$$(\overline{p}_{12} \vee \overline{p}_{11}) \wedge (\overline{p}_{12} \vee \overline{p}_{13})$$

Y, por último, la activación de la primera válvula en el último instante de tiempo dará lugar a:

$$(\overline{p}_{13} \vee \overline{p}_{11}) \wedge (\overline{p}_{13} \vee \overline{p}_{12})$$

Razonando ahora de la misma manera, con las otras tres válvulas se generarían en total hasta 24 cláusulas resultantes de este segundo grupo.

En total, la modelizacioń sugerida consiste en 12 variables y 36 cláusulas.

2. Un problema de satisfacción de restricciones se define como una terna (X, D, C) donde $X = \{x_i\}_{i=1}^n$ es el conjunto de variables; $D = \{D_i\}_{i=1}^n$ representa los dominios de cada variable respectivamente y $C = \{C_i\}_{i=1}^m$ es el conjunto de restricciones del problema.

   Como de costumbre, se considera primero la selección de variables X, para las cuales se definirán luego sus dominios, y todas las restricciones del problema.

   Como antes, se sugiere que la variable x_i represente el instante de activación de la válvula i-ésima. Con esta definición, resulta obvio entonces que su dominio serán únicamente los instantes plausibles de activación: $D_i = \{1, 2, 3\}$.

   A continuación se consideran ordenadamente todas las restricciones del problema (cuya interpretación se omite ahora, puesto que ya se dió antes), para cada una de las cuales se producen la colección de tuplas $R_{ij} = \{(a_i, a_j)\}, a_i \in D_i, a_j \in D_j$ que representan los valores simultaneamente legales para la activación de las válvulas i-ésima y j-ésima, respectivamente:

   $V_1 < V_2$ En este caso, son valores simultaneamente legales para la primera y segunda válvula los siguientes:

   $$R_{12} = \{(1, 2), (1, 3), (2, 3)\}$$

   $V_1 > V_3$ Si la primera válvula debe activarse después que la tercera, entonces:

   $$R_{13} = \{(2, 1), (3, 1), (3, 2)\}$$

   $V_1 \neq V_4$ En este caso:

   $$R_{14} = \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}$$

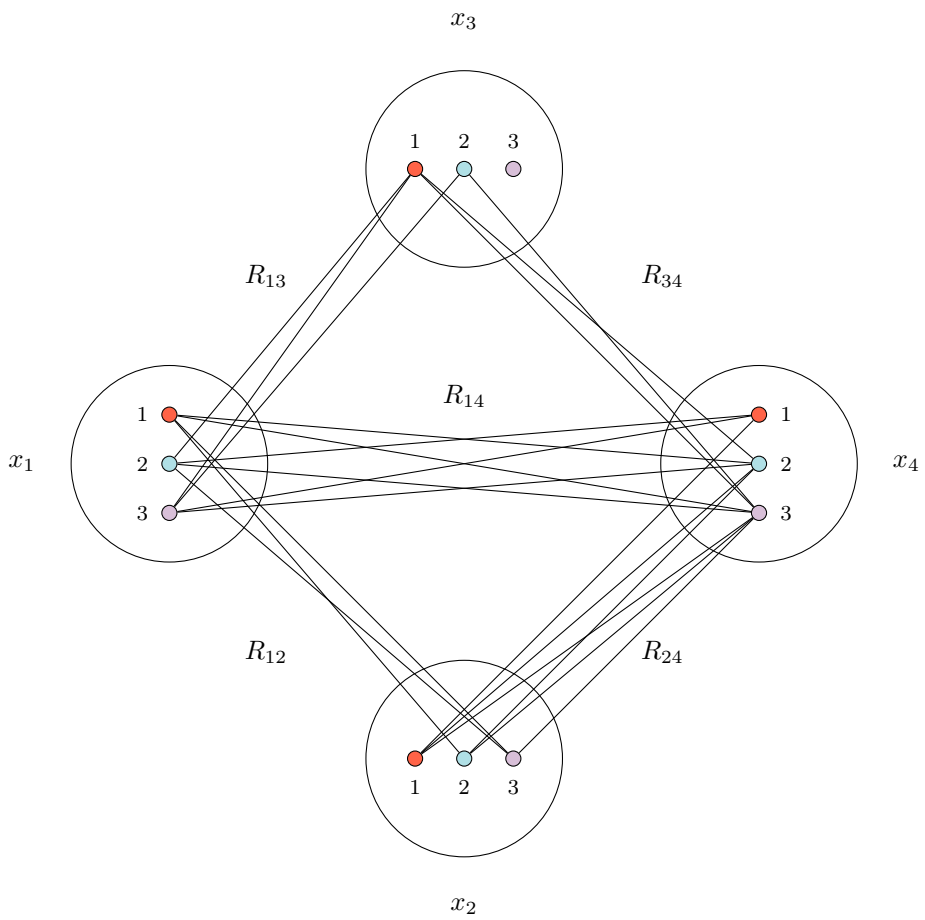
   $V_2 \leq V_4$ Que es, como ya se advirtió muy parecida al primer caso, $V_1 < V_2$, dando lugar a:

   $$R_{24} = \{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)\}$$

   $V_3 < V_4$ Análogamente al primer caso:

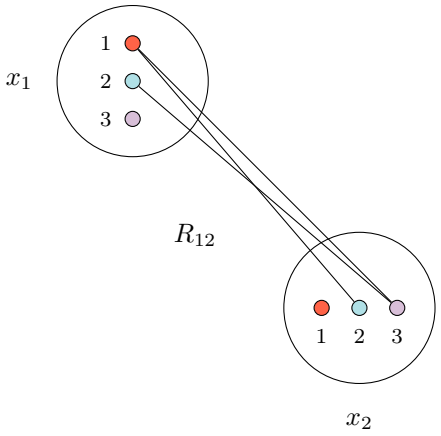
   $$R_{34} = R_{12} = \{(1, 2), (1, 3), (2, 3)\}$$

   La siguiente figura representa el *grafo de restricciones* del problema dado en el enunciado donde cada vértice es una variable x_i, y las restricciones R_{ij} se representan como arcos entre valores de cada dominio, mostrados dentro de cada vértice.



3. Se dice que la variable x_i es *arco-consistente* con la variable x_j, si y sólo si, para cada valor en el dominio de la primera variable, $a_i \in D_i$, existe otro valor en el dominio de la segunda variable, $a_j \in D_j$, tal que ambos valores son simultaneamente legales, esto es, $(a_i, a_j) \in R_{ij}$. La *arco-consistencia* es una propiedad que se debe forzar, y que sirve para eliminar valores imposibles de algunos dominios: cada vez que no exista ningún valor en el dominio de la segunda variable, x_j, que soporte la asignación de a_i a la primera variable, x_i, entonces a_i puede eliminarse del primer dominio.

   Para el estudio de la arco-consistencia entre las variables que representan el instante de activación de la primera y segunda válvula (x_1 y x_2 respectivamente), se sugiere usar la sección del grafo de restricciones que representa las relaciones entre estas dos variables:



Usando el grafo de restricciones resulta obvio que el valor 3, en el dominio de la primera variable, no está soportado por ningún valor en el dominio de la segunda y, por lo tanto, puede eliminarse del dominio de la primera variable.

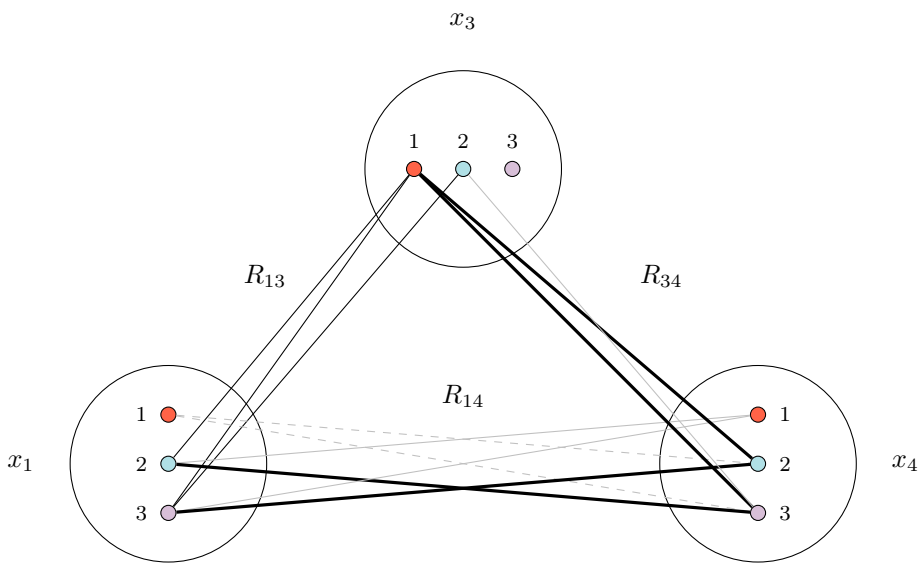
Como la arco-consistencia sólo puede eliminar valores del dominio de la primera variable (x_1 en este caso), se dice que es una propiedad *direccional*. Por lo tanto, debe forzarse también la arco-consistencia de x_2 hacia x_1.

Otra vez, examinando la sección de interés del grafo de restricciones, resulta obvio que el valor 1 en el dominio de x_2, no está soportado por ningún valor en el dominio de la primera variable, x_1 y, por lo tanto, puede ser eliminado.

Una vez realizadas estas dos operaciones puede concluirse que efectivamente x_1 y x_2 son arco-consistentes.

4. Se dice que la variable x_i es camino consistente con la variable x_j respecto de una tercera variable, x_k, si y sólo si, para cada par de valores simultaneamente legales para las dos primeras variables, $(a_i, a_j) \in R_{ij}$, existe al menos un valor en el dominio de la variable de referencia x_k, $a_k \in D_k$, que soporte al mismo tiempo los valores de las dos primeras variables. Esto es, $(a_i, a_k) \in R_{ik}$, y $(a_j, a_k) \in R_{jk}$. Como antes, la camino-consistencia es una propiedad que debe forzarse en la red de restricciones, y que sirve para eliminar tuplas de aquellas restricciones que no estén debidamente soportadas por una tercera variable: cada vez que no haya un valor $a_k \in D_k$ que soporte la asignación de a_i y a_j a las dos primeras variables, entonces la tupla (a_i, a_j) puede eliminarse de la restricción R_{ij}, sin riesgo de introducir inconsistencias.

   Para resolver la camino-consistencia de x_1 con x_3 respecto de x_4 se sugiere otra vez examinar la sección del grafo de restricciones que contiene a estas tres variables:



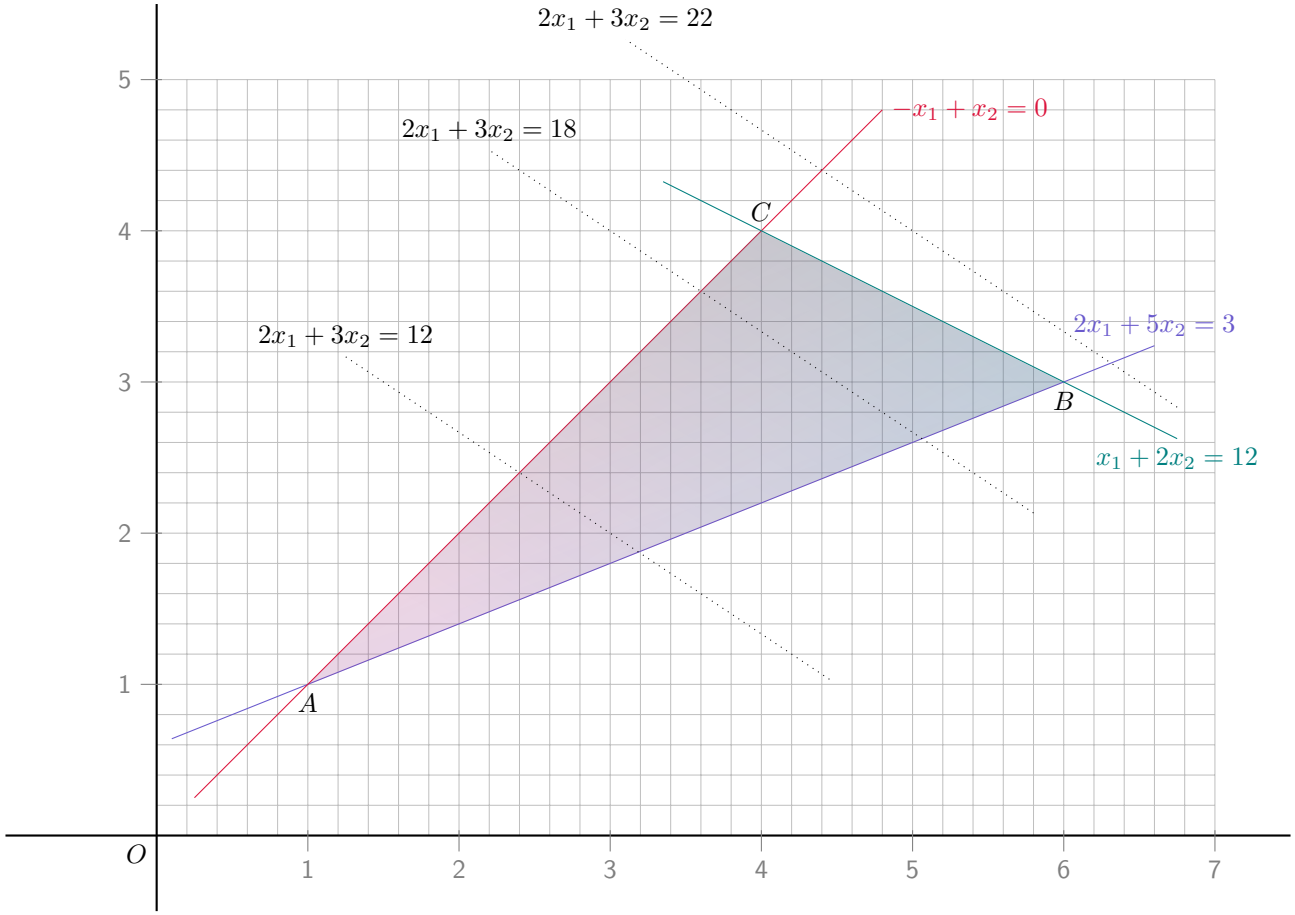
La figura preserva todas las tuplas de la restricción R_{13} como se mostraban en el grafo de restricciones original. Como el valor 1 había sido eliminado del dominio de la primera restricción, su consideración se omite, y por lo tanto su relación con la cuarta variable, x_4 se muestra con arcos punteados en gris. Para cada tupla $(a_1, a_2) \in R_{12}$ se han resaltado con arcos más gruesos aquellos valores de la variable de referencia, x_4 que soportan la asignación de ese par de valores a las dos primeras variables, x_1 y x_3 respectivamente. Como puede verse:

- Las tuplas $\{(2,1),(3,1)\} \in R_{13}$ están debidamente soportadas por un valor en el dominio de x_4, 3 y 2 respectivamente.
- Sin embargo, la tupla $(3,2) \in R_{13}$ no está soportada por ningún valor en el dominio de x_4. Por este motivo, la tupla $(2,3) \in R_{34}$ se muestra con una línea continua en gris, y la tupla $(3,2)$ puede eliminarse de la restricción R_{13}.

Una vez realizadas estas operaciones, puede concluirse que efectivamente x_1 es camino-consistente con x_3 respecto de x_4.

! Problema 3

1. La siguiente figura muestra la región factible que resulta de la intersección de las regiones factibles de cada restricción (cuyas fronteras están marcadas en rojo, azul y verde).



Los puntos A, B y C se calculan como la intersección de las rectas correspondientes y resultan, por lo tanto, de la resolución de sistemas de ecuación compatibles determinados de dos variables con dos ecuaciones (cuyo cálculo se omite aquí):

$$A(1,1)$$
$$B(6,3)$$
$$C(4,4)$$

Tal y como asegura el teorema fundamental del *simplex*, la solución óptima será un *punto extremo* de la región factible. Para ver cuál será, la misma figura muestra en negro varias curvas de isobeneficio (denominadas así porque la función objetivo es de maximización). Como se puede ver, según aumenta el valor de z, el último punto que se toca de la región factible es el punto B que será la solución buscada con un valor de la función objetivo z:

$$z^* = c^T B = \begin{pmatrix} 2 & 3 \end{pmatrix} \begin{pmatrix} 6 \\ 3 \end{pmatrix} = 21$$

Una forma alternativa de calcular la solución consiste en evaluar la función objetivo en cada punto extremo:

$$z(A)=c^T A=\begin{pmatrix}2 & 3\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}=5$$

$$z(B)=c^T B=\begin{pmatrix}2 & 3\end{pmatrix}\begin{pmatrix}6\\3\end{pmatrix}=21$$

$$z(C)=c^T C=\begin{pmatrix}2 & 3\end{pmatrix}\begin{pmatrix}4\\4\end{pmatrix}=20$$

Como puede verse, el punto extremo con el mayor valor de la función objetivo es el punto B, de modo que la solución óptima del problema planteado es:

$$x^* = \begin{pmatrix}6\\3\end{pmatrix}$$

con un valor de la función objetivo $z^* = 21$

2. ¡Por supuesto que no! Una de las consideraciones fundamentales del diseño del algoritmo SIMPLEX es que se aplica a la resolución de tareas que se expresan con restricciones lineales, y con una función objetivo lineal —y, por eso a esta técnica se la conoce como *Programación Lineal*.

   Por el contrario, $z = x_1^2 + x_2^2$, no es una función lineal, es decir, no se trata de un polinomio de primer grado en sus variables y, por lo tanto, no puede resolverse con SIMPLEX.

3. A partir de este apartado se pedía considerar la función objetivo (constante) $z = 0$, de modo que la tarea de Programación Lineal que hay que tratar es:

$$\begin{array}{c} \max z = 0 \\ \begin{array}{rcrcrcl} - & 2x_1 & + & 5x_2 & \geq & 3 \\ - & x_1 & + & x_2 & \leq & 0 \\ & x_1 & + & 2x_2 & \leq & 12 \end{array} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

   Un problema de programación lineal está en forma *estándar* si todas las restricciones son de igualdad, las variables de decisión son no negativas y, por último, el vector de constantes o recursos $\mathbf{b}$ no contiene términos negativos. Estará, además, en forma de maximización si la función objetivo maximiza y de minimización en otro caso. El problema, tal y como estaba enunciado, ya verifica todas estas condiciones salvo la primera. Conviene aquí recordar:

   - Una restricción de la forma $\leq$ está acotada superiormente. Puesto que ninguna variable de decisión puede tomar valores negativos, es preciso *sumar* una *variable de holgura* para forzar la igualdad.
   - Análogamente, las restricciones de la forma $\geq$ están acotadas inferiormente de modo que, con variables de decisión que no pueden tomar valores negativos, es preciso *restar* una *variable de holgura* para forzar la igualdad.

   y, en cualquier caso, las variables de holgura se añaden a la función objetivo con coeficiente nulo.

   Por lo tanto, el problema de Programación Lineal queda, como sigue, en forma estándar de maximización:

$$\begin{array}{c} \max z = 0 \\ \begin{array}{rcrcrcrcrcl} - & 2x_1 & + & 5x_2 & - & x_3 & & & & = & 3 \\ - & x_1 & + & x_2 & & & + & x_4 & & = & 0 \\ & x_1 & + & 2x_2 & & & & & + & x_5 & = & 12 \end{array} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

Ahora bien, el enunciado pedía, además, transformar el problema de modo que fuera posible iniciar la aplicación del algoritmo SIMPLEX con una base igual a la matriz identidad. Actualmente, los vectores columna $\mathbf{a_4}$ y $\mathbf{a_5}$ son vectores columna de la matriz identidad, pero falta aún un vector que tenga un 1 en la primera posición. Para conseguirlo, simplemente se añade una *variable artificial* a la primera restricción:

$$\begin{array}{c} \max z = -\infty x_6 \\ \begin{array}{rcrcrcrcrcl} - & 2x_1 & + & 5x_2 & - & x_3 & + & x_6 & & & = & 3 \\ - & x_1 & + & x_2 & & & & & + & x_4 & = & 0 \\ & x_1 & + & 2x_2 & & & & & & & = & 12 \end{array} \\ \mathbf{x} \geq \mathbf{0} \end{array}$$

que se añade, además, a la función objetivo con una penalización infinitamente alta.

Como puede verse, la tarea de programación lineal resultante: uno, está en forma estándar de maximización; segundo, los vectores columna $\mathbf{a_6}$, $\mathbf{a_4}$ y $\mathbf{a_5}$ ya forman una base que es igual a la matriz identidad —y, precisamente, en ese mismo orden.

4. El algoritmo del SIMPLEX consiste en la aplicación iterativa de tres pasos: cálculo de las variables básicas, selección de la variable de entrada y selección de la variable de salida hasta que se detecte alguna de las siguientes condiciones:

   - El problema puede mejorar el valor de la función objetivo indefinidamente. Se dice entonces que el problema está *no acotado*. Este caso se detecta cuando todas las componentes y_i de la variable de decisión x_i elegida para entrar en la base son todos negativos o nulos.
   - El problema tiene soluciones infinitas. Este caso se detecta cuando todos los costes reducidos son no negativos, y hay al menos uno que es nulo. Además, no tiene que haber variables artificiales en la base actual.
   - El problema es irresoluble. Esto ocurre cuando en el segundo paso, todos los costes reducidos son no negativos y el primer paso asignó un valor no negativo a alguna variable artificial.
   - Se alcanza una solución factible y puede demostrarse que no es posible mejorarla. Esta condición se detecta como en el segundo caso pero cuando las variables artificiales (si las hubiera) tienen valores nulos.

   Paso 0 Cálculo de una solución factible inicial

   a) Cálculo de las variables básicas
   La primera iteración se inicia con una base igual a la matriz identidad de dimensión 3, tal y como se calculó ya en el apartado anterior. Por lo tanto, son variables básicas en este paso $\{x_6, x_4, x_5\}$.
   Nótese la importancia del orden en el que se indican las variables básicas, y es que ése es el único orden de los vectores columna $\mathbf{a_i}$ que pueden formar la matriz identidad.

   $$B_0 = I_3 \qquad B_0^{-1} = I_3$$
   $$x_0^* = B_0^{-1} b = b = \begin{pmatrix} 3 \\ 0 \\ 12 \end{pmatrix} \quad z_0^* = c_{B_0}^T x_0^* = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} \begin{pmatrix} 3 \\ 0 \\ 12 \end{pmatrix} = -3\infty$$

   En estos cálculos todos los coeficientes de las variables de decisión en la función objetivo son nulos, salvo el de la variable artificial, que vale $-\infty$.

   b) Selección de la variable de entrada
   En las expresiones siguientes el cálculo de los vectores y_i, $y_i = B_0^{-1} a_i$, se ha embebido en el cálculo de los *costes reducidos* directamente (aunque en una iteración con una base igual a la matriz identidad, $\mathbf{y_i} = \mathbf{a_i}$):

$$z_1 - c_1 = c_{B_0}^T B_0^{-1} a_1 - c_1 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} -2 \\ -1 \\ 1 \end{pmatrix} - 0 = 2\infty$$

$$z_2 - c_2 = c_{B_0}^T B_0^{-1} a_2 - c_2 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} 5 \\ 1 \\ 2 \end{pmatrix} - 0 = -5\infty$$

$$z_3 - c_3 = c_{B_0}^T B_0^{-1} a_3 - c_3 = \begin{pmatrix} -\infty & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} - 0 = \infty$$

En los cálculos anteriores, ∞ se ha utilizado como un símbolo cualquiera. También es posible usar una constante M muy alta (y, en particular, un valor de M mayor que la suma de los valores absolutos de todos los coeficientes en la función objetivo es suficiente para penalizar las variables artificiales). Usando ∞ como un símbolo cualquiera es posible saber qué valores son más grandes sustituyéndolo por valores arbitrariamente grandes.
En este caso particular, la variable no básica con el valor más negativo es x_2, asi que ésta será la variable que entre en la siguiente iteración.

c) Selección de la variable de salida
La regla de salida establece que debe salir aquella variable con el menor cociente x_i/y_{ij} donde x_i es la variable elegida en el paso anterior (x_2) y el vector $\mathbf{y_i}$ será entonces el vector columna $\mathbf{a_2}$ porque, como se advirtió en el punto anterior, con una base igual a la matriz identidad $\mathbf{y_i} = \mathbf{a_i}$:

$$\text{mín} \left\{ \frac{3}{5}, \frac{0}{1}, \frac{12}{2} \right\}$$

donde todos los cocientes son tenidos en cuenta, puesto que todos ellos son estrictamente positivos. El menor de los cocientes es el que se corresponde con la segunda de las variables básicas de esta iteración, x_4, que será, por tanto, la variable que abandonará la base.

Paso 1 Mejora de la solución actual (iteración #1)

a) Cálculo de las variables básicas
A continuación se mejora la calidad de la solución anterior. Las nuevas variables básicas son $\{x_2, x_5, x_6\}$.
Nótese que ahora no se ordenan las variables básicas con ningún criterio específico, puesto que cualquier base, que resultara de cualquier ordenación, será igualmente válida.

$$B_1 = \begin{pmatrix} 5 & 0 & 1 \\ 1 & 0 & 0 \\ 2 & 1 & 0 \end{pmatrix} \qquad B_1^{-1} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 1 \\ 1 & -5 & 0 \end{pmatrix}$$

$$x_1^* = B_1^{-1} b = b = \begin{pmatrix} 0 \\ 12 \\ 3 \end{pmatrix} \quad z_1^* = c_{B_1}^T x_1^* = \begin{pmatrix} 0 & 0 & -\infty \end{pmatrix} \begin{pmatrix} 0 \\ 12 \\ 3 \end{pmatrix} = -3\infty$$

b) Selección de la variable de entrada
A continuación se muestra el cálculo de *costes reducidos* para todas las variables no básicas. En primer lugar, se muestra el cálculo de todos los vectores columna $\mathbf{y_j} = B_1^{-1} \mathbf{a_j}$:

$$y_1=\begin{pmatrix}-1\\3\\3\end{pmatrix} \qquad y_3=\begin{pmatrix}0\\0\\-1\end{pmatrix}$$
$$y_4=\begin{pmatrix}1\\-2\\-5\end{pmatrix}$$

y que se usan a continuación:

$$z_1-c_1=c_{B_1}^T y_1-c_1=\begin{pmatrix}0 & 0 & -\infty\end{pmatrix}\begin{pmatrix}-1\\3\\3\end{pmatrix}-0=-3\infty$$
$$z_3-c_3=c_{B_1}^T y_3-c_3=\begin{pmatrix}0 & 0 & -\infty\end{pmatrix}\begin{pmatrix}0\\0\\-1\end{pmatrix}-0=\infty$$
$$z_4-c_4=c_{B_1}^T y_4-c_4=\begin{pmatrix}0 & 0 & -\infty\end{pmatrix}\begin{pmatrix}1\\-2\\-5\end{pmatrix}-0=5\infty$$

La única variable con un coste reducido negativo es x_1 que será, por lo tanto, la variable elegida para entrar en la base en la siguiente iteración.
Aunque la variable x_4 no podía volver a entrar, porque había salido en la iteración anterior, se ha calculado su coste reducido para asegurarse de que efectivamente tiene un valor no negativo —para cualquier valor arbitrariamente grande de ∞.

c) Selección de la variable de salida
Nuevamene, la variable de salida se calcula en atención al mínimo cociente x_i/y_{ij} donde x_i es la variable elegida en el paso anterior para añadirse a la base (x_1) e y_{ij} son las componentes de su vector **y** también calculados en el paso anterior:

$$\text{mín}\left\{\frac{0}{\cancel{-1}}, \frac{12}{3}, \frac{3}{3}\right\}$$

Como se ve, el primer cociente tiene denominador negativo y, por ese motivo, se desecha. Por lo tanto, la variable que sale es la última, x_6, precisamente la variable artificial.

Paso 2 Mejora de la solución actual (iteración #2)

a) Cálculo de las variables básicas
Las nuevas variables básicas son $\{x_1, x_2, x_5\}$

$$B_2=\begin{pmatrix}-2 & 5 & 0\\-1 & 1 & 0\\-5 & 2 & 1\end{pmatrix} \qquad B_2^{-1}=\begin{pmatrix}\frac{1}{3} & -\frac{5}{3} & 0\\\frac{1}{3} & -\frac{2}{3} & 0\\1 & -7 & 1\end{pmatrix}$$
$$x_2^*=B_2^{-1}b=b=\begin{pmatrix}1\\1\\15\end{pmatrix} \quad z_2^*=c_{B_2}^T x_2^*=\begin{pmatrix}0 & 0 & 0\end{pmatrix}\begin{pmatrix}1\\1\\15\end{pmatrix}=0$$

Como puede verse, la proyección sobre las dos primeras variables de decisión del punto calculado en esta iteración es:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

que es el punto A mostrado en la resolución gráfica del primer apartado.

b) Selección de la variable de entrada
A continuación se calculan todos los costes reducidos. En este paso, se obvia el cálculo del coste reducido de la variable artificial puesto que al restar su coeficiente en la función objetivo saldría un valor positivo arbitrariamente grande.
Primero, se calculan todos los vectores columna $\mathbf{a_j} = B_2^{-1}\mathbf{a_j}$:

$$y_3 = \begin{pmatrix} -\frac{1}{3} \\ -\frac{1}{3} \\ -1 \end{pmatrix} \quad y_4 = \begin{pmatrix} -\frac{5}{3} \\ -\frac{2}{3} \\ -7 \end{pmatrix}$$

y que se emplean en el cálculo de costes reducidos $(z_j - c_j)$ que se muestra a continuación:

$$z_3 - c_3 = c_{B_2}^T y_3 - c_3 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} -\frac{1}{3} \\ -\frac{1}{3} \\ -1 \end{pmatrix} - 0 = 0$$

$$z_4 - c_4 = c_{B_2}^T y_4 - c_4 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} -\frac{5}{3} \\ -\frac{2}{3} \\ -7 \end{pmatrix} - 0 = 0$$

Como se ve, no hay ningún coste reducido negativo. Ahora bien, se observa que todos los costes reducidos son nulos, lo que indica que hay puntos adyacentes con el mismo valor de la función objetivo que el punto extremo calculado en esta iteración.
De hecho, es ahora cuando conviene observar que la función objetivo es constante (siempre vale 0 independientemente de los valores que tomen las variables de decisión), lo que significa que cualquier punto de la región factible tiene el mismo valor. Se trata, como se indicará a continuación de un problema con *infinitas soluciones*.

5. La interpretación de un problema incluye varias consideraciones como son estudiar: si el problema es o no satisfacible, si la solución es única o hay varias soluciones o si está o no acotado. Además, debe estudiarse el uso de recursos: si sobra o no alguno y cual es su contribución al crecimiento de la función objetivo.

   Interpretación de la solución De la solución se puede advertir lo siguiente:

   - El problema tiene soluciones infinitas, porque la función objetivo es constante, de modo que cualquier par de puntos x y x' que pertenezcan a la región factible tendrán el mismo valor de la función objetivo, 0. Este caso se ha reconocido porque todos los costes reducidos son nulos.
   - Por lo tanto, no existe una solución única.

   Interpretación de los recursos La importancia de los recursos puede estudiarse con la resolución del problema dual puesto que el valor óptimo de la variable dual i-ésima representa la contribución al crecimiento de la función objetivo por unidad de recurso i-ésimo. Esto es exactamente lo que se hará en el siguiente apartado.

6. Para la resolución de este apartado, basta con recordar la *interpretación económica* de las soluciones de un problema dual que advierte que:

La variable dual $x_i'^*$ indica la contribución por unidad del recurso i-ésimo b_i a la variación el en valor óptimo z^* actual del objetivo

Puesto que el enunciado requiere la contribución unitaria de todos los recursos, entonces es preciso calcular la solución completa del problema dual.

Para ello, es posible iniciar la aplicación de otro SIMPLEX al problema dual (en forma *simétrica*) del primal. Sin embargo, en su lugar es preferible hacer uso del siguiente resultado teórico:

> Si el problema de programación lineal en forma simétrica tiene una solución óptima correspondiente a una base $\mathbf{B}$, entonces $\mathbf{x'}^T = \mathbf{c_B^T B^{-1}}$ es una solución óptima para el problema dual

En este teorema, los términos usados para el cálculo de la solución óptima del problema dual se refieren al problema primal, salvo que se indique explícitamente lo contrario. Por lo tanto c_B^T es el vector de costes de las variables básicas en la solución del problema primal y B la base usada para el cálculo de la misma solución —que se mostró en el último paso de aplicación del SIMPLEX. Por el contrario, x'^T es la solución del problema dual.

Como quiera que los coeficientes de las variables básicas son todos nulos, no es necesario realizar ningún cálculo puesto que c_B^T multiplicado por cualquier matriz dará lugar al vector nulo, por lo tanto:

$$x'^* = c_B^T B^{-1} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

El resultado es congruente con la interpretación ofrecida en el apartado anterior: si la función objetivo es constante, 0, entonces la disponibilidad de nuevos recursos no hará que crezca en absoluto, como de hecho testimonian las componentes del vector x'^*.

Problema 4

A diferencia de otros problemas de búsqueda heurística, este problema requiere *maximizar* una función objetivo.

1. El espacio de estados es una formalización que habilita la aplicación de algoritmos de búsqueda (informados o no) para la resolución de problemas. Consiste únicamente, en la definición de estados y operadores que sirven para transitar entre ellos. Relacionando, después, el conjunto de posibles estados con otro de vértices V y el de operadores (convenientemente instanciados) con otro de arcos E, resulta entonces de forma natural la definición de un grafo, el *grafo de búsqueda* que se recorrerá eficientemente con el uso de *árboles de búsqueda*. Este ejercicio propone ejercitar todos estos conceptos y, en el primer apartado, el del espacio de estados.

   Estados Un estado en este problema está constituído por una *permutación* de cajas (equivalentemente, por una ordenación única de objetos sin repetición) que ya han sido parcialmente dispuestos, y una lista (por lo tanto, sin ordenar necesariamente por ningún criterio) de cajas pendientes de apilarse, si fuera posible.

   El concepto esencial que debe modelarse, por lo tanto, es el de `Caja`:

   Caja Todas las `Caja`s son indistinguibles salvo por su capacidad, $\texttt{C}_\texttt{i}$, y peso, $\texttt{W}_\texttt{i}$, que se representarán con atributos dedicados. Además, para poder referenciar las cajas de forma única, cada una de ellas se identifica por un `id`.

   Tal y como se advertía antes, cada estado debe mantener información actualizada de los trabajos que ya han sido asignados, y de aquellos que aún están pendientes de ordenarse en la cola de impresión. Para ello, se sugiere el uso de dos contenedores:

   Pila Consiste en un contenedor *ordenado* (por ejemplo, desde la base) de instancias de `Caja` que represente la configuración física que se considera en cada estado.
   Inicialmente `Pila`$= \varnothing$. Un estado final debería tener una `Pila` no vacía que maximice el número de cajas que se pueden apilar verticalmente.

Pendientes Consiste en otro contenedor *no ordenado* de instancias de `Caja` que aún no han sido apilados.
Inicialmente, `Pendientes` contendrá la colección $\mathcal{S} = \{B_i\}_{i=1}^{n}$. Nótese que, en el estado final, no es necesariamente cierto que `Pendientes` debe ser el conjunto vacío, $\varnothing$.

Precisamente, la gestión de estas listas se llevará a cabo con el operador que se detalla a continuación:

Operadores En la definición de operadores es importante describir sus precondiciones/postcondiciones y, además, el coste que tienen. En este problema hay un único operador:

Apilar Este operador recibe un estado y una caja $B_j \in$ `Pendientes`, y la añade a la pila, poniéndola en la cima del contenedor `Pila`.

Precondiciones Para poder añadir un nuevo bloque en la cima de una pila, es preciso que todos los bloques que estén por debajo puedan soportarle. Por lo tanto, la capacidad de cada bloque en la pila debe ser mayor o igual que el del peso de todas las cajas que se dispongan sobre el:

$$W_j + \sum_{i=1}^{|\texttt{Pila}|} W_i \leq C_i$$

donde W_j es el peso del bloque B_j que se dispone en la cima, y $|$`Pila`$|$ es el número de cajas apiladas actualmente cuando se añade el bloque B_j.

Postcondiciones El operador debe eliminar el bloque B_j del contenedor `Pendientes` y añadirlo al contenedor ordenado `Pila`.

Coste En este problema no existe una noción de coste, puesto que el propósito consiste en maximizar una función objetivo. Sin embargo, es necesario disponer de una métrica que maximizar. Puesto que se pide maximizar el número de cajas, el *coste* de este operador es 1.
Se trata, por lo tanto, de un problema de *costes unitarios*.

2. Formalmente hablando, el tamaño del espacio de estados se calcula como el número diferente de estados que pueden definirse. Obviamente, habrá tantos estados como configuraciones parciales de bloques, y éstos se pueden calcular con el uso de números combinatorios.

   Sin embargo, la expresión que *domina* el número de estados es el del número de disposiciones de bloques diferentes que puede haber. Como advertía el apartado anterior, las pilas son *permutaciones* de cajas y, por lo tanto, el tamaño del espacio de estados es $O(N!)$.

3. Tal y como se decía en el apartado anterior, el operador `Apilar` definido en el primer apartado sugiere crear las soluciones incrementalmente. Por lo tanto, cualquier solución consistirá en un camino desde la raíz de un árbol de búsqueda (donde `Pila`$= \varnothing$), hasta otra donde $|$`Pila`$| = N$, como máximo. Puesto que cada invocación del operador `Apilar` sólo añade un bloque (el que recibe como argumento), entonces la profundidad está acotada por N, el número de bloques.

   Por otra parte, el factor de ramificación, b, se define como el número medio de sucesores. Inicialmente, el nodo raíz tendrá hasta N sucesores, puesto que es posible elegir uno cualquiera de los N bloques en `Pendientes`. A profundidad $d = 1$, ya se habrá apilado un bloque, de modo que cualquier nodo a esta profundidad deberá elegir uno entre $(N-1)$ bloques. En general, cualquier nodo a profundidad d tendrá hasta $(N-d)$ sucesores.

   Puesto que el número de sucesores de todos los nodos a la misma profundidad es el mismo, basta con considerar cualquier solución (o camino de longitud N desde la raíz del árbol de búsqueda) para calcular el número medio de sucesores:

$$b = \frac{N + (N-1) + (N-2) + \cdots + 2 + 1}{N} = \frac{N\frac{(1+N)}{2}}{N} = \frac{1+N}{2}$$

donde se ha dividido convenientemente por el número de nodos en cualquier camino, N, el número de sucesores generado en cada nodo que hay en el mismo camino.

4. Tal y como se explicaba en el apartado anterior, a profundidad d habrá hasta d bloques apilados. Como este es un problema de *maximización*, las mejores soluciones se encontrarán a profundidades mayores. Esta observación desacredita el uso de algoritmos de el mejor primero (como Dijkstra) o del primero en amplitud, puesto que tienen un consumo de memoria exponencial precisamente en la profundidad de los caminos que recorren.

   Por otra parte, los algoritmos del primero en profundidad son una elección razonable, puesto que como demostraba el apartado anterior, la profundidad máxima está acotada por N, el número de bloques.

   De entre los algoritmos del primero en profundidad que cabe considerar, no tiene sentido emplear el algoritmo del primero en profundización iterativa, puesto que en un problema de maximización no tiene ninguna utilidad recorrer profundidades menores antes de recorrer las siguientes.

   Son, por lo tanto, elecciones razonables: *Ramificación y acotación en profundidad (DFBnB)* o *Primero en profundidad*. Es importante observar que *cualquier solución* a profundidad N (esto es, que consiga apilar todos los bloques), es una solución óptima, puesto que no es posible apilar más bloques que esos.

   Estos algoritmos tienen un consumo de memoria lineal. Sin embargo, no tienen lista CERRADA (esto es, no tienen *memoria*) y, por lo tanto, pueden re-expandir el mismo nodo tantas veces como caminos haya hasta ellos.

5. Puesto que se pide maximizar una función objetivo (el número de cajas que se pueden apilar), la definición de admisibilidad en este caso consiste en una estimación que no infraestime el número de cajas que pueden disponerse, $h(n) \geq h^*(n)$, donde n es una estado cualquiera, y $h^*(n)$ es el número máximo de cajas que puede llegar a añadirse por encima de su `Pila`.

   En cualquier caso, la generación de heurísticas admisibles se sigue de la técnica de *relajación de restricciones*. En su aplicación, se observan las restricciones del problema y se relajan todas o un subconjunto de ellas hasta que es posible resolver el problema resultante de forma óptima. Como quiera que las restricciones del problema se encuentran típicamente en las *precondiciones* de los operadores del problema (estudiados en el primer apartado) son relajaciones factibles las siguientes:

   a) La capacidad, C_i, de todas las cajas es infinitamente grande (o, equivalentemente, el peso, W_i, de todas las cajas es nulo).

   En este caso, cualquier permutación de cajas sería una solución óptima. En otras palabras, cualquier estado podría extenderse con todas las cajas en `Pendientes` para formar pilas del mismo tamaño, N. Se trata, por lo tanto de una heurística *no informada*, puesto que siempre produciría la misma estimación para todos los estados.

   b) Una relajación más útil que la anterior consiste, entonces, en considerar que todas las cajas tienen una capacidad finita, igual a la de la caja que está en la base. Por lo tanto, si la caja inferior soporta toda la carga de las cajas que hay sobre élla, ellas también soportarán sus cargas —puesto que la relajación consiste en asumir que tienen la misma capacidad que la caja inferior, al tiempo que deben soportar menos peso.

   Con esta relajación es posible construir una función heurística resolviendo un problema transformado de éste, el *problema de la mochila*. El primer apartado del primer ejercicio del examen de Enero de 2017 consiste precisamente en la solución óptima de este problema en particular.

   c) Ahora bien, no es necesariamente cierto que la caja en la base sea la que más restringe el número de cajas que puedan apilarse.

   Otra relajación posible, que tendrá en cuenta todas las cajas ya apiladas, consiste en asumir que todas las cajas en `Pendientes` tienen el mismo peso, y que es igual al mínimo de ellas:

$$\omega = \min_{W_k \in \texttt{Pendientes}} \{W_k\}, \quad \forall W_j \in \texttt{Pendientes}$$

Sea el *slack* del bloque i-ésimo en la `Pila`, s_i, la carga que aún podría soportar. Esto es:

$$s_i = C_i - \sum_{j=i}^{|Pila|} W_i$$

que necesariamente debe ser una cantidad no negativa para todos los bloques apilados. Ahora, gracias a la relajación propuesta, la expresión:

$$\text{mín} \left\{ \frac{s_i}{w} \right\}, \quad \forall B_i \in \texttt{Pila}$$

devolverá el número máximo de bloques en `Pendientes` que pueden disponerse en la cima de la `Pila`

6. Son algoritmos a considerar los siguientes:

   A* El algoritmo A* es admisible y garantiza, por lo tanto, que encontrará soluciones óptimas si la función heurística que lo guía también es admisible. Además, es un algoritmo rápido puesto que no reexpande nodos (y, con frecuencia, las ordenaciones de la lista abierta se pueden hacer en $O(1)$ con las estructuras de datos adecuadas si la función objetivo sólo toma valores enteros. Sin embargo, tiene un consumo de memoria exponencial.

   IDA* El algoritmo IDA* reexpande nodos en caso de que haya transpociones pero no ordena nodos y, mucho más importante aún, tiene un consumo de memoria lineal en la profundidad de la solución (que en nuestro caso es siempre igual a N). Además, también es un algoritmo de búsqueda admisible.

   Ahora bien, incluso si hay una función heurística disponible, las consideraciones del apartado 4 siguen siendo ahora válidas, y el algoritmo A* no puede recomendarse tajantemente puesto que las mejores soluciones se encuentran a las mayores profundidades, y encontrar estas incurre en un gasto exponencialmente grande de memoria.

   Por lo tanto, independientemente del número de transposiciones, el algoritmo que se sugiere es IDA*.