Departamento de Informática
Universidad Carlos III de Madrid

Grado en Informática

Heurística y Optimización

27 de Junio de 2019

Normas generales del examen

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

② Para obtener la puntuación de cada apartado, las respuestas deben estar debidamente razonadas.

③ No se responderá a ninguna pregunta sobre el examen transcurridos los primeros **30 minutos**

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)

El *tipo de cambio* entre dos monedas c_i y c_j, r_{ij}, se aplica regularmente en la compra de una moneda c_j pagando con otra diferente, c_i. Por ejemplo, si el tipo de cambio de cambio del euro (€) a la libra inglesa (£) es de $0,89$, entonces es posible comprar hasta $12 \times 0,89 = 10,68$ libras con 12 euros.

A continuación se muestran las tasas de cambio oficiales a fecha del 23 de Junio de 2019 entre las siguientes monedas: euro (€), dólar estadounidense ($), libra inglesa (£), yen japonés (¥), won surcoreano (₩), y naira nigeriano (₦).

r_{ij}	Moneda destino					
Moneda origen	€	$	£	¥	₩	₦
€	–	1,14	0,89	122,30	1.320,94	410,24
$	0,88	–	0,78	107,32	1.159,17	360,00
£	1,12	1,27	–	136,74	1.477,00	458,71
¥	0,0082	0,0093	0,0073	–	10,80	3,35
₩	0,00076	0,00086	0,00068	0,093	–	0,31
₦	0,0024	0,0028	0,0022	0,30	3,22	–

Como se ve en la tabla anterior, es posible también adquirir libras inglesas con euros, comprando primero dólares estadounidenses. De esa manera, con 12 euros es posible comprar hasta $12 \times 1,14 \times 0,78 = 10,67$ libras, con lo que se pierden $0,01$ libras en comparación con la compra directa. Por supuesto, cualesquiera otras combinaciones mostradas en la tabla anterior son posibles también.

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ puntos)** Dada una moneda origen y una moneda destino, ¿qué propiedad debe verificar la secuencia de cambios con la que se gana la mayor cantidad de dinero?

Téngase en cuenta que para evitar razonar con productos, es posible tomar el logaritmo decimal de cada tipo de cambio, puesto que $\max \prod r_{ij} = \max \sum \log(r_{ij})$.

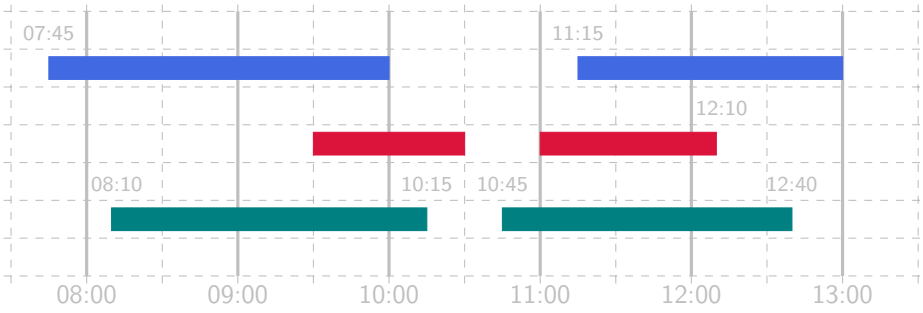
(b) **(½ puntos)** Dada una moneda origen y una moneda destino, ¿existe algún algoritmo de *Programación Dinámica* que sirva para determinar la secuencia de cambios con la que se pierde la mayor cantidad de dinero?

Téngase en cuenta que para evitar razonar con productos, es posible tomar el logaritmo decimal de cada tipo de cambio, puesto que $\min \prod r_{ij} = \min \sum \log(r_{ij})$.

(c) **(½ puntos)** Asimismo, ¿existe algún algoritmo de *Programación Dinámica* para determinar la secuencia de cambios con la que se pierde la mayor cantidad de dinero entre dos monedas cualesquiera?

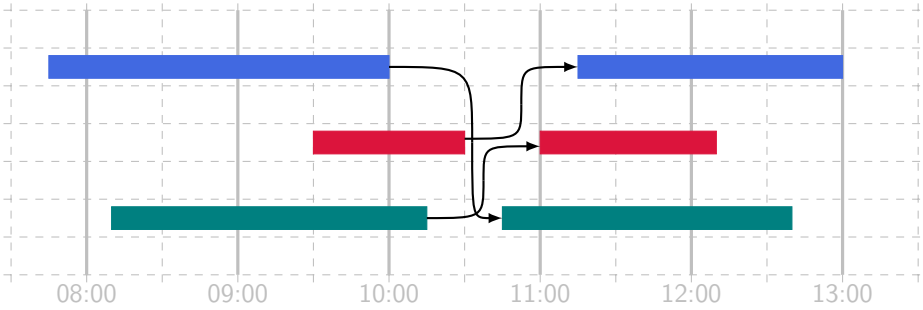
Pregunta 2 (1½ puntos)

Una de las operaciones fundamentales en el diseño robusto de las operaciones de las compañías aéreas es el *re-routing*, que consiste en re-asignar los vuelos de cada avión de modo que se reduzca tanto como sea posible el tiempo máximo en tierra de todos los aviones. Considérese, por ejemplo, la secuencia de aterrizajes y despegues mostrados en la siguiente figura:

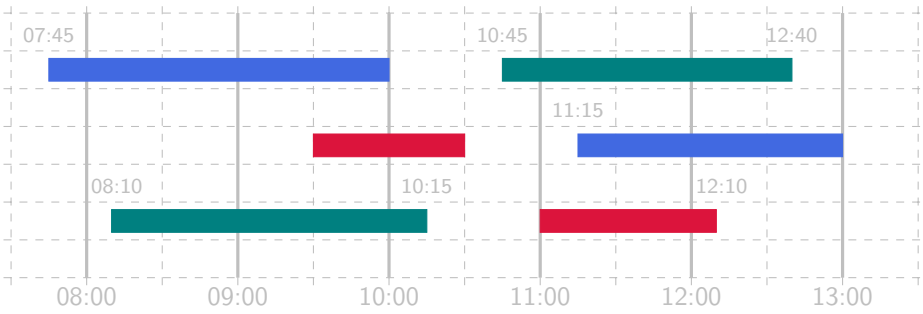


donde cada rectángulo representa un vuelo, de modo que el margen izquierdo representa la hora de despegue, y el margen derecho, su hora de aterrizaje, y todos los vuelos mostrados en la misma línea son operados por el mismo avión. Es fácil comprobar que el avión que pasa más tiempo en tierra es el primero, 75 minutos en total, mientras que los aviones segundo y tercero están 30 minutos cada uno.

Habida cuenta de que en este caso, todos los aviones aterrizan en el mismo aeropuerto, es posible re-asignar los vuelos como se indica en la siguiente figura:



donde se ha tenido especial cuidado en no asignar un despegue a un avión que aterriza después, naturalmente, de modo que la nueva red de vuelos sería la que se muestra en la siguiente figura:



y con el que se consigue reducir el tiempo máximo en tierra de todos los aviones, puesto que ahora todos esperan exactamente 45 minutos.

Se pide responder razonadamente las siguientes preguntas, explicando claramente en cada apartado las *variables de decisión* elegidas, y las *restricciones* necesarias:

(a) **(½ puntos)** Dada una lista de las horas de aterrizaje y despegue de todos los aviones que operan en el mismo aeropuerto como *parámetros*, se pide determinar las *variables de decisión* y *restricciones* de una tarea de Programación Lineal que sirvan para re-asignar todos los aviones de la flota a los despegues.

(b) **(½ puntos)** Dada la tarea de Programación Lineal definida en el apartado anterior, se pide determinar la *función objetivo* que sirva para minimizar el tiempo máximo en tierra de todos los aviones.

Es preciso indicar claramente cualquier cambio en la definición de restricciones y variables de decisión del apartado anterior, si hiciera falta alguno.

Recientemente, el departamento de *Data Science* de la compañía aérea ha desarrollado un simulador a partir de datos históricos, con el que es posible estimar el retraso en el aterrizaje y despegue de cada vuelo.

Se pide responder razonadamente la siguiente pregunta:

(c) **(½ puntos)** Dada una lista de retrasos en el aterrizaje y despegue de cada vuelo como *parámetros*, se pide indicar claramente todos los cambios necesarios en la tarea de Programación Lineal del apartado anterior para minimizar el tiempo máximo de espera en tierra de todos los aviones considerando todos los retrasos.

Pregunta 3 (3 puntos)

Considerése el siguiente problema de Programación Lineal:

$$\text{máx } z = x_1 + 7x_3$$

$$\begin{array}{rcrcrcr} 2x_1 & - & x_2 & + & 2x_3 & \geq & -1 \\ 3x_1 & + & 2x_2 & - & 6x_3 & \leq & 2 \\ 2x_1 & + & 5x_2 & + & 4x_3 & \leq & 7 \end{array}$$

$$\mathbf{x} \geq \mathbf{0}$$

Se pide responder razonadamente las siguientes preguntas:

(a) **(½ puntos)** ¿Es posible (o necesario) realizar alguna transformación en la tarea de Programación Lineal anterior para resolver la maximización de la función objetivo $|x_1 + 7x_3|$?

(b) **(½ puntos)** Expresar la primera tarea de Programación Lineal en forma *estándar* de maximización, y preparar el problema de Programación Lineal para su aplicación con el algoritmo SIMPLEX para garantizar que pueda comenzarse con la matriz identidad.

(c) **(1 punto)** Resolver el problema de Programación Lineal obtenido en el apartado anterior con el algoritmo SIMPLEX.

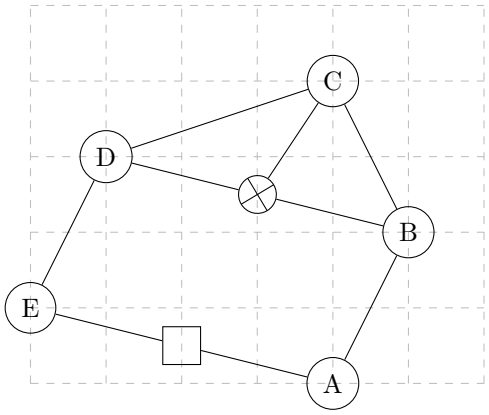
Es imprenscindible indicar claramente, en cada iteración: las variables escogidas en la base, su valor, y el valor de la función objetivo

(d) **(½ puntos)** Calcula la contribución por unidad de recurso al valor óptimo de la función objetivo calculado en el apartado anterior.

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

Pregunta 4 (4 puntos)

Una compañía dispone de un vehículo eléctrico para la inspección de diferentes estaciones de trabajo, distinguidas con una letra en un círculo en el siguiente diagrama:



Al inicio de cada turno de trabajo, el vehículo eléctrico está en la estación central, mostrada con un cuadrado en el diagrama anterior, y debe inspeccionar todas las estaciones de trabajo, para acabar aparcado de nuevo en la estación central cuando finaliza el turno. La figura muestra, asimismo, las únicas carreteras transitables que unen la estación central con las estaciones de trabajo, y a éstas entre sí. Además, desde algunas estaciones de trabajo es posible acceder a una estación de carga, mostradas con el símbolo —⊗—.

El vehículo puede recargarse también en la estación central, de modo que al inicio de cada turno se sabe con seguridad que la batería está completamente llena. Sin embargo, es posible que no disponga de autonomía suficiente para visitar todas las estaciones de trabajo antes de finalizar el turno, en cuyo caso debería visitar las estaciones de carga tantas veces como sea necesario.

La compañía conoce con exactitud la distancia entre estaciones de cualquier tipo, así como de la cantidad de batería que se consume en el movimiento entre éllas, y desea optimizar la distancia recorrida en cada turno de trabajo para poder visitar todas las estaciones de trabajo antes de volver a la estación central.

Se pide responder razonadamente las siguientes preguntas:

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

(b) **(½ puntos)** ¿Cuál es el factor de ramificación máximo desarrollado por un árbol de búsqueda para resolver óptimamente este problema?

(c) **(½ puntos)** Si se desea calcular la solución óptima, ¿qué algoritmo de búsqueda de *fuerza bruta* sugerirías para su resolución? ¿Por qué?

(d) **(1 punto)** Sugiere una función heurística $h(\cdot)$ que sea *admisible* e *informada* para resolver óptimamente el problema de minimizar la distancia recorrida teniendo en cuenta todas las restricciones del problema.

(e) **(½ puntos)** ¿Qué algoritmo de búsqueda *heurística* es el más indicado para resolver óptimamente el problema? ¿Por qué?

La compañía ha adquirido recientemente un segundo vehículo eléctrico que desea usar también para la inspección de las estaciones de trabajo, de modo que cada estación de trabajo sea visitada por un único vehículo.

Se pide responder razonadamente las siguientes preguntas:

(f) **(½ puntos)** ¿Cuál es el factor de ramificación máximo desarrollado por un árbol de búsqueda para resolver óptimamente el problema de minimizar la distancia recorrida por los dos vehículos?

(g) **(½ puntos)** ¿Es necesario modificar o adaptar la función heurística del apartado d para minimizar la distancia recorrida por los dos vehículos en la visita a todas las estaciones de trabajo? Sí o no, y por qué.

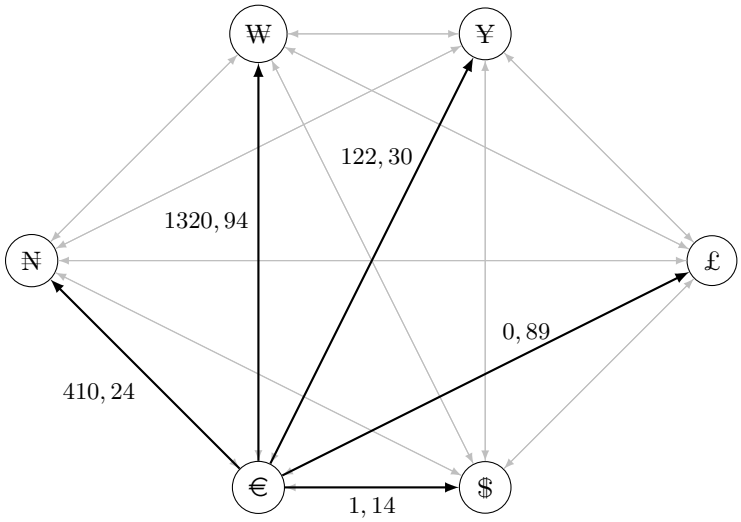
Soluciones del examen de Heurística y Optimización Junio 2019

Problema 1

1. Dada una secuencia de N cambios entre $(N+1)$ monedas, $\langle c_0, c_1, \ldots, c_N \rangle$, la cantidad de la última moneda, c_N, que puede adquirirse con una unidad de la primera moneda, c_0, es $\prod_{i=0}^{N-1} r_{i,i+1}$. Esta observación sugiere:

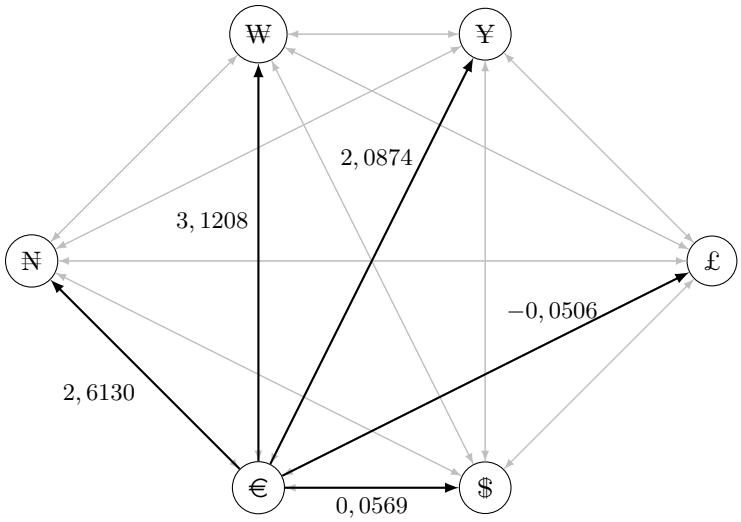
   - Que cualquier secuencia de cambios $\langle c_0, c_1, \ldots, c_N \rangle$ puede modelarse como un camino en un grafo $G(V, E)$, donde hay un vértice $v \in V$ por cada moneda disponible, y existe un arco $(v_i, v_j) \in E$ si y sólo si es posible adquirir la moneda c_j con la moneda c_i.
   - Que cada arco $(v_i, v_j) \in E$ tiene un valor que es igual al tipo de cambio entre las monedas c_i y c_j, r_{ij}.

   La siguiente Figura muestra una vista parcial del grafo G que puede construirse a partir de las monedas indicadas en el enunciado y sus tipos de cambio. Para no dificultar su legibilidad, sólo se muestran los costes de los arcos que se originan desde el vértice que representa las adquisiciones de otras monedas con euros, €.



   De modo que dadas una moneda inicial, c_0, y una moneda final, c_N, el problema de encontrar la secuencia de cambios, $\langle c_0, c_1, \ldots, c_N \rangle$, con la que se gana la mayor cantidad de dinero, se resuelve encontrando un camino en el grafo G desde el vértice v_0, hasta el vértice v_N, que maximice $\prod_{i=0}^{N-1} r_{i,i+1}$. Sin embargo, los algoritmos de *Programación Dinámica* que buscan caminos en un grafo consideran el *modelo aditivo*, por el que el coste de un camino es la suma de los costes de sus arcos, en vez de su producto.

   Afortunadamente, tal y como explica el enunciado, el problema de maximizar $\prod r_{ij}$, es estrictamente equivalente al problema de maximizar $\sum \log(r_{ij})$. La siguiente Figura muestra la misma vista del grafo anterior G donde se han sustituido los costes de los arcos que representan los tipos de cambio de las adquisiciones con euros, €, por su logaritmo decimal:



Por lo tanto, modificando el grafo G mostrado anteriormente, para que el coste de cada arco $(v_i, v_j) \in E$ sea, simplemente, el logaritmo (en cualquier base), del tipo de cambio entre las monedas que une, $\log(r_{ij})$, el problema de de encontrar la secuencia de cambios con la que se gana más dinero es ahora equivalente al problema de encontrar el camino de mayor coste entre los vértices v_0 y v_N.

2. En el segundo apartado se pide determinar si existe algún algoritmo de *Programación Dinámica* con el que sea posible calcular la secuencia de cambios entre dos monedas, c_0, y c_N, con la que se pierde la mayor cantidad de dinero. Por lo tanto, ahora se trata de encontrar un camino en G desde el vértice v_0 hasta el vértice v_N que minimice $\prod r_{ij}$.

   Como antes, el problema de minimizar el producto de los tipos de cambio es equivalente al problema de minimizar la suma de los logaritmos (en cualquier base) de las mismas tasas de cambio.

   Ahora bien, el uso de logaritmos (de cualquier base) hará que el coste de algunos arcos sean positivos (si y sólo si $r_{ij} > 1$), y otros negativos —si y sólo si $r_{ij} < 1$—, tal y como puede verse en el segundo grafo mostrado anteriormente.

   Por lo tanto, el problema pedido se resuelve con un algoritmo de que encuentre el camino más corto en un grafo con costes tanto positivos como negativos. El algoritmo de *Programación Dinámica* de Bellman-Ford-Moore parece un buen candidato pero, desafortunadamente, no puede aplicarse si hay ciclos negativos.

   Por ello, si no hubiera ciclos negativos, el problema se resuelve con el algoritmo de Bellman-Ford-Moore. Desgraciadamente, para el caso en el que hubiera ciclos negativos, no se conoce ningún algoritmo de *Programación Dinámica* que resuelva eficientemente el problema.

3. Por último, calcular la secuencia de cambios que produce la mayor pérdida económica entre dos monedas cualesquiera puede resolverse calculando las mayores pérdidas económicas entre todos los pares de vértices $(v_i, v_j), v_i, v_j \in V$ o, equivalentemente, el camino de menor coste entre todos los pares de vértices $v_i, v_j \in V$ después de hacer que el coste de cada arco sea igual al logaritmo (en cualquier base) de la tasa de cambio entre las monedas que une.

   Para ello, podría usarse el algoritmo de Floyd-Warshall, que resuelve exactamente este problema si y sólo si no hay ciclos negativos.

   Por lo tanto, si no hubiera ciclos negativos, el problema se resuelve con el algoritmo de Floyd-Warshall. Desgraciadamente, para el caso en el que hubiera ciclos negativos, no se conoce ningún algoritmo de *Programación Dinámica* que resuelva eficientemente este problema.

Problema 2

1. Tal y como repite el enunciado del problema una y otra vez, se trata de un *problema de asignación*. Si i representa el aterrizaje del vuelo i-ésimo, y j el despegue del vuelo j-ésimo, entonces hay que asignar los aviones que aterrizan con cada uno de los despegues.

   Por lo tanto, sea x_{ij} una variable binaria definida como sigue:

   $$x_{ij} = \begin{cases} 1, & \text{si y sólo si el avión del aterrizaje } i\text{-ésimo se asigna al despegue } j\text{-ésimo} \\ 0, & \text{en otro caso} \end{cases}$$

   Por ejemplo, si los vuelos de la primera Figura se numeran como 1, 2 y 3 de arriba hacia abajo, entonces la solución mostrada en la tercera Figura del enunciado se modela como: $x_{ij} = 0, 1 \leq i, j \leq 3$, salvo $x_{13} = 1$, $x_{21} = 1$ y $x_{32} = 1$.

   Las restricciones del problema deben establecer, para una cantidad arbitraria N de aviones:

   - Que cada avión debe asignarse a un despegue, y sólo a uno:

     $$\sum_{j=1}^{N} x_{ij} = 1, 1 \leq i \leq N$$

   - Y, además, que cada despegue debe ser realizado por un avión, y sólo uno:

     $$\sum_{i=1}^{N} x_{ij} = 1, 1 \leq j \leq N$$

   - Por último, es preciso asegurar que nunca se asignará un despuegue a un avión antes de su hora de aterrizaje. Para ello, se tendrá en cuenta que la hora de aterrizaje y despegue de cada avión están disponibles como parámetros tal y como indicaba el enunciado:

     $$\begin{array}{ll} \alpha_i: & \text{Hora de aterrizaje del avión } i\text{-ésimo} \\ \beta_j: & \text{Hora de despuegue del avión } j\text{-ésimo} \end{array}$$

     de modo que si $\beta_j \leq \alpha_i$, entonces necesariamente $x_{ij} = 0$. Para ello, basta con la restricción:

     $$(\beta_j - \alpha_i)x_{ij} \geq 0 \tag{1}$$

     Si la hora de aterrizaje del avión i-ésimo, α_i, es menor o igual que la de despegue del avión j-ésimo, β_j, $\alpha_i \leq \beta_j$, entonces $(\beta_j - \alpha_i) \geq 0$, y x_{ij} puede tomar uno cualquiera de los valores 0 ó 1. Sin embargo, si el avión i-ésimo aterrizara después de la hora de despegue del avión j-ésimo, $\alpha_i > \beta_j$, entonces $(\beta_j - \alpha_i) < 0$, y la única forma de hacer que se cumpla la restricción anterior es con $x_{ij} = 0$[1].

2. Para poder minimizar el tiempo máximo en tierra, es preciso crear otra *variable de decisión* que calcule el tiempo en tierra cuando el avión que aterriza el i-ésimo, es asignado al despegue j-ésimo:

   $$t_{ij} = \begin{cases} \beta_j - \alpha_i, & \text{si y sólo si } x_{ij} = 1 \\ 0 \quad , & \text{si y sólo si } x_{ij} = 0 \end{cases}$$

   Y para ello, basta con añadir la siguiente restricción:

[1] En un escenario más realista, la verificación de esta restricción debería hacerse asegurando que la hora de despegue del vuelo j-ésimo, β_j, debe ser mayor o igual que la hora de aterrizaje del vuelo i-ésimo, α_i, más una tolerancia τ, de modo que la restricción mostrada debería ser en realidad: $(\beta_j - \alpha_i - \tau)x_{ij} \geq 0$.

$$t_{ij} \geq (\beta_j - \alpha_i)x_{ij} \tag{2}$$

que resulta de un razonamiento muy parecido al que se siguió para la tercera restricción del apartado anterior: si $x_{ij} = 1$, entonces $(\beta_j - \alpha_i)x_{ij} \geq 0$, y t_{ij} será, al menos, igual a la diferencia de tiempos $(\beta_j - \alpha_i)$; por el contrario, si $x_{ij} = 0$, entonces $(\beta_j - \alpha_i)x_{ij} = 0$, y t_{ij} podría tomar cualquier valor positivo.

Ciertamente, la restricción anterior no acota superiormente los valores de t_{ij} pero, si se calcula el máximo de ellos añadiendo otra *variable de decisión*, $t_{\text{máx}}$, definida como el máximo de todos los t_{ij}:

$$t_{\text{máx}} \geq t_{ij}, 1 \leq i, j \leq N$$

entonces la función objetivo:

$$\text{mín}\, z = t_{\text{máx}}$$

hará que la solución óptima minimice el tiempo de espera más largo de todos los aviones después de hacer la asignación.

3. Dada una lista de parámetros con los retrasos de todos los aterrizajes y despegues:

$$\begin{array}{ll} \delta_i : & \text{Retraso del aterrizaje del avión } i\text{-ésimo} \\ \epsilon_j : & \text{Retraso del despuegue del avión } j\text{-ésimo} \end{array}$$

la tarea de Programación Lineal del apartado anterior puede reformularse sumando, simplemente, los retrasos a las horas de aterrizaje y despegue para obtener las horas efectivas. Nótese que δ_i y ϵ_j pueden tomar valores positivos (es decir, retrasos), negativos (anticipaciones) o nulos —sin cambios en el horario previsto.

Las únicas restricciones que emplean explícitamente las horas de aterrizaje y despegue son las restricciones (1) y (2) que simplemente deben reescribirse como:

$$\begin{array}{l} (\beta_j + \delta_j - \alpha_i - \epsilon_i)x_{ij} \geq 0 \\ t_{ij} \geq (\beta_j + \delta_j - \alpha_i - \epsilon_i)x_{ij} \end{array}$$

y no es preciso realizar ningún cambio adicional.

Problema 3

1. No es necesario realizar ninguna transformación en absoluto y, en este caso en particular, la maximización de la función objetivo $|x_1 + 7x_3|$ es estrictamente equivalente a la maximización de la función objetivo dada en el enunciado, $x_1 + 7x_3$, puesto que:

   - Primero, todas las variables de decisión deben tomar valores no negativos;
   - Segundo, todos los coeficientes de la función objetivo son no negativos.

   Por lo tanto, el resultado de $x_1 + 7x_3$ es siempre no negativo y, por ello $|x_1 + 7x_3| = x_1 + 7x_3$.

2. 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, sólo verifica la segunda condición. 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.

Además, las variables de holgura que se añadan a las restricciones para forzar igualdades, se añadirán a la función objetivo z con un coeficiente nulo.

En primer lugar, se cambia el signo del recurso de la primera restricción para hacer que sea positivo. Para ello, basta con multiplicar los dos términos de la primera restricción por -1 (con lo que, además, se cambia el sentido de la desigualdad):

$$\text{máx } z = x_1 + 7x_3$$

$$\begin{array}{rrcrcrcl} - & 2x_1 & + & x_2 & - & 2x_3 & \leq & 1 \\ & 3x_1 & + & 2x_2 & - & 6x_3 & \leq & 2 \\ & 2x_1 & + & 5x_2 & + & 4x_3 & \leq & 7 \end{array}$$

$$\mathbf{x} \geq \mathbf{0}$$

A continuación se añaden variables de holgura para convertir todas las desigualdades en igualdades tal y como se explicaba anteriormente. Por lo tanto, el problema de Programación Lineal queda, como sigue, en forma estándar de maximización:

$$\text{máx } z = x_1 + 7x_3$$

$$\begin{array}{rrcrcrcrcrcl} - & 2x_1 & + & x_2 & - & 2x_3 & + & x_4 & & & = & 1 \\ & 3x_1 & + & 2x_2 & - & 6x_3 & & & + & x_5 & = & 2 \\ & 2x_1 & + & 5x_2 & + & 4x_3 & & & & & = & 7 \end{array}$$

$$\mathbf{x} \geq \mathbf{0}$$

El problema de Programación Lineal obtenido de esta forma, contiene todas las columnas de la matriz identidad (de dimensión 3): $\mathbf{a_4}$, $\mathbf{a_5}$ y $\mathbf{a_6}$, por lo que es posible iniciar la aplicación del SIMPLEX inmediatamente.

3. 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 es infactible. Esto ocurre cuando en el segundo paso, todos los costes reducidos son positivos 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.
   - El problema tiene soluciones infinitas. En este caso todos los costes reducidos calculados en el segundo paso son positivos y hay, al menos, uno con coste nulo. La arista que une el punto calculado en la iteración actual con aquél que se alcanzaría si entra una variable con coste reducido nulo (cuando no hay costes reducidos negativos) son todas soluciones óptimas de la tarea de Programación Lineal.

   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 $\{x_4, x_5, x_6\}$, precisamente en este orden puesto que son precisamente las columnas $\mathbf{a_4}$, $\mathbf{a_5}$ y $\mathbf{a_6}$ las que recrean la matriz identidad de rango 3, I_3:

$$B_0 = I_3 \qquad B_0^{-1} = I_3$$
$$x_0^* = B_0^{-1}b = b = \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix} \quad z_0^* = c_{B_0}^T x_0^* = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 7 \end{pmatrix} = 0$$

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

$$z_1 - c_1 = c_{B_0}^T B_0^{-1} a_1 - c_1 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} -2 \\ 3 \\ 2 \end{pmatrix} - 1 = -1$$
$$z_2 - c_2 = c_{B_0}^T B_0^{-1} a_2 - c_2 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix} - 0 = 0$$
$$z_3 - c_3 = c_{B_0}^T B_0^{-1} a_3 - c_3 = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} I_3 \begin{pmatrix} -2 \\ -6 \\ 4 \end{pmatrix} - 7 = -7$$

Como se ve, la variable no básica con el coste reducido más negativo de todos es x_3 que será, por lo tanto, la variable elegida para entrar en la base 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} (con valores y_{ij} estrictamente positivos) donde x_i es la variable elegida en el paso anterior (x_2):

$$\text{mín}\left\{\frac{1}{\not 2}, \frac{2}{\not 6}, \frac{7}{4}\right\}$$

y sale la variable x_6.

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 $\{x_3, x_4, x_5\}$:

$$B_1 = \begin{pmatrix} -2 & 1 & 0 \\ -6 & 0 & 1 \\ 4 & 0 & 0 \end{pmatrix} \qquad B_1^{-1} = \begin{pmatrix} 0 & 0 & \frac{1}{4} \\ 1 & 0 & \frac{1}{2} \\ 0 & 1 & \frac{3}{2} \end{pmatrix}$$
$$x_1^* = B_1^{-1}b = \begin{pmatrix} \frac{7}{4} \\ \frac{9}{2} \\ \frac{25}{2} \end{pmatrix} \quad z_1^* = c_{B_1}^T x_1^* = \begin{pmatrix} 7 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{7}{4} \\ \frac{9}{2} \\ \frac{25}{2} \end{pmatrix} = \frac{49}{4}$$

Como puede verse, la función objetivo ha crecido (como debe ser, puesto que se trata de un problema de maximización) respecto de la iteración anterior de 0 a $\frac{49}{4}$.

b) Selección de la variable de entrada
Como antes, el cálculo de los vectores columna $y_i = B_1^{-1}a_i$ se ha embebido en el cálculo de los costes reducidos:

$$z_1 - c_1 = c_{B_1}^T B_1^{-1} a_1 - c_1 = \begin{pmatrix} 7 & 0 & 0 \end{pmatrix} B_1^{-1} \begin{pmatrix} -2 \\ 3 \\ 2 \end{pmatrix} - 1 = \tfrac{5}{2}$$

$$z_2 - c_2 = c_{B_1}^T B_1^{-1} a_2 - c_2 = \begin{pmatrix} 7 & 0 & 0 \end{pmatrix} B_1^{-1} \begin{pmatrix} 1 \\ 2 \\ 5 \end{pmatrix} - 0 \ = \tfrac{35}{4}$$

$$z_6 - c_6 = c_{B_1}^T B_1^{-1} a_6 - c_6 = \begin{pmatrix} 7 & 0 & 0 \end{pmatrix} B_1^{-1} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - 0 \ = \tfrac{7}{4}$$

y ninguna variable puede entrar en la base sin reducir el valor de la función objetivo, z_0^*, puesto que todos los costes reducidos son estrictamente positivos. Con ello, finaliza la aplicación del algoritmo SIMPLEX.

4. Para la resolución del último 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. 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 $\mathbf{x'}^T = \mathbf{c_B}^T\mathbf{B}^{-1}$ 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.

En particular:

$$x'^* = c_B^T B^{-1} = \begin{pmatrix} 7 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & \frac{1}{4} \\ 1 & 0 & \frac{1}{2} \\ 0 & 1 & \frac{3}{2} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \frac{7}{4} \end{pmatrix}$$

de donde resulta que la contribución del tercer recurso al crecimiento de la función objetivo es $\frac{7}{4}$, mientras que el primer y segundo recurso no contribuyen en absoluto a su crecimiento.

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 es factible porque la solución no contiene valores positivos para ninguna variable artificial. De hecho, ni siquiera existen variables artificiales.
- Existe una solución óptima única, puesto que los costes reducidos de la última iteración del algoritmo SIMPLEX son todos estrictamente positivos.
- Por ello, además, el valor de la función objetivo está naturalmente acotado.

Interpretación de los recursos La interpretación de los recursos se hace, fundamentalmente, observando los valores de la variable de holgura en la solución óptima y la solución del problema dual.

- En la solución óptima, el valor de las variables de holgura $x_4 = \frac{9}{2}$ y $x_5 = \frac{25}{2}$, indican que hay esa cantidad de recursos en las restricciones primera y segunda que no se usan. Esto es, que sobran.
- Por último, la solución del problema dual en el apartado anterior sirve para observar que el único recurso que puede contribuir al crecimiento de la función objetivo es el tercero, mientras que el primer y segundo recurso no lo hacen.
  De hecho, tal y como advertía el punto anterior, hay una cantidad de los recursos de la primera y segunda restricción que, de hecho, sobran.

Problema 4

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 En este problema un estado se representa con información *estática* sobre el dominio de aplicación y, por otra parte, con la posición del vehículo en cada momento. La información de carácter estático es aquella que no cambia en ningún estado, y se refiere a los diferentes tipos de estaciones y su caracterización.

   Estación central La estación central podría estar caracterizada por un identificador, `central.id`, que, aparentemente, es innecesario porque es única por definición, pero como se verá más adelante se necesita para poder acceder a otras estructuras que representan la distancia y el consumo eléctrico.

   Estación de trabajo Las estaciones de trabajo se distinguen por su identificador (puesto que, de hecho, puede haber una cantidad arbitraria de ellas), `trabajo.id`.

   Estación de carga Aunque en el ejemplo del enunciado se mostraba una única estación de carga, lo cierto es que podría haber una cantidad arbitraria de ellas, de modo que también se distinguirán por un identificador único, `carga.id`.

   Todos estos tipos de estaciones podrían heredar de otra definición genérica de `estación`:

   Estación Un atributo fundamental de cualquier estación es la distancia hasta cualquier otro tipo de estación (y, por lo tanto, de definición genérica de `estación`), que se almacena en un atributo dedicado, `estación.distancia` y que se indexa por el identificador único de la `estación` de destino, `estacion.distancia[estacion-destino.id]`.
   Asimismo, para poder verificar la factibilidad de los movimientos planeados para el vehículo eléctrico, es necesario disponer de una tabla con el consumo de la batería entre estaciones que se almacenará de manera análoga a la distancia, esto es, con un atributo dedicado que se indexa por la estación de destino, `estacion.consumo[estacion-destino.id]`.

Operador	Precondiciones	Postcondiciones	Coste
$\texttt{mover}(i, j)$	$\texttt{vehiculo.estacion} = i \wedge$ $i.\texttt{distancia}[j] \neq -1 \wedge$ $i.\texttt{bateria}[j] \leq \texttt{vehiculo.bateria}$	$\texttt{vehiculo.estacion} = j \wedge$ $\texttt{vehiculo.bateria} -= i.\texttt{bateria}[j]$	$i.\texttt{distancia}[j]$
$\texttt{recargar}(i, j)$	$\texttt{vehiculo.estacion} = i \wedge$ $i.\texttt{distancia}[j] \neq -1 \wedge$ $i.\texttt{bateria}[j] \leq \texttt{vehiculo.bateria}$	$\texttt{vehiculo.estacion} = j \wedge$ $\texttt{vehiculo.bateria} = 100$	$i.\texttt{distancia}[j]$

Tabla 1: Definición de operadores

Es importante que tanto la distancia como el consumo de la batería se almacenan sólo para aquellas estaciones que son inmediatamente accesibles. Por lo tanto, si desde una estación i no es posible llegar inmediatamente hasta otra estación j, entonces $i.\texttt{distancia}[j] = -1$ y, análogamente, $i.\texttt{consumo}[j] = -1$.

Por último, la posición del vehículo se representa con una instancia específica como se indica a continuación:

Vehiculo La posición del vehículo se almacena simplemente con una referencia al identificador de la estación en la que se encuentra, (`vehiculo.estacion`) puesto que el sistema no necesitará guardar información en ruta del movimiento del vehículo. Además, es preciso almacenar la cantidad de batería de la que dispone, `vehiculo.bateria`.

Tanto en el estado inicial como en el estado final, `vehiculo.estacion` será igual al identificador de la estación central. En el estado inicial, la carga de la batería del vehículo será igual al 100 %, `vehiculo.bateria` = 100, mientras que a la finalización del turno, el valor puede ser cualquiera —pero necesariamente no negativo.

Acciones Conceptualmente hay una única acción, `mover`, que toma como parámetros una estación origen y otra final para realizar el movimiento del vehículo. Sin embargo, si la estación de destino es una estación de carga, además de realizar el movimiento del vehículo es preciso llenar la batería del vehículo.

Por lo tanto, se distinguirán dos operadores, `mover` y `recargar`. En cualquier caso, las precondiciones deben verificar que el vehículo se encuentra efectivamente en la estación de origen, y que la estación de destino es inmediatamente accesible desde la estación de origen y que, además, hay batería suficiente en el vehículo para hacer el recorrido. En cuanto a los efectos, ambos operadores actualizarán la posición del vehículo con una referencia a la estación de destino, pero el segundo, además, llenará la bateraí al 100 %.

Por último, es importante caracterizar el coste de cada uno de estos operadores. Como quiera que la métrica de optimización es la distancia recorrida, entonces el coste de cada operador será igual a la distancia que separa la estación de destino de la origen.

La Tabla 1 muestra los operadores definidos junto con sus precondiciones, postcondiciones y coste.

2. El factor de ramificación máximo es igual al número máximo de operadores que pueden instanciarse desde cualquier estado. El apartado anterior define sólo dos acciones, `mover` y `recargar` y, cualquiera de ellas, puede aplicarse únicamente en el tránsito desde una estación hasta otra inmediatamente accesible —puesto que las dos acciones están protegidas por la precondición $i.\texttt{distancia}[j] \neq -1$.

   Por lo tanto, el factor de ramificación máximo es igual al número máximo de estaciones inmediatamente adyacentes desde una cualquiera, y que será un número con frecuencia mucho menor que el número total de estaciones.

3. De la definición del espacio de estados se sigue la observación de que éste es un problema de minimización con costes arbitrarios. Por lo tanto, pueden usarse uno cualquiera de los algoritmos de *fuerza bruta* o *no informados* estudiados específicamente para el caso de costes variables:

Ramificación y acotación en profundidad (DFBnB) Tiene un coste de memoria lineal pero puede re-expandir muchos nodos en caso de que el dominio presente muchas *transposiciones* (como ocurre con todos los algoritmos de *el primero en profundidad*).

Dijkstra Consiste en un algoritmo de *el mejor primero* donde la función de evaluación, $f(n)$ es, simplemente, el coste del camino desde el estado inicial hasta n, $g(n)$. Tiene un coste de memoria exponencial pero garantiza que cada vez que expande un nodo habrá encontrado la solución óptima hasta él puesto que los nodos se expanden en orden creciente de su valor de $f(n)$ —o, equivalentemente, de $g(n)$.

En este problema es fácil advertir que el número de transposiciones (o formas diferentes de llegar desde una estación hasta cualquier otra) es particularmente alto y, por lo tanto, el algoritmo que se sugiere es el algoritmo de Dijkstra.

4. Una heurística muy sencilla que puede obtenerse con la técnica de *relajación de restricciones*, consiste en relajar el uso de la batería (imaginando, por lo tanto, que el vehículo no necesitará recargarla nunca), y que todas las estaciones son accesibles desde cualquier otra.

   A partir de la posición de cada estación es posible calcular la *distancia aérea* entre todos los pares de estaciones. De esta manera, considerando que aún es preciso visitar un determinado número de estaciones a partir de cualquier nodo n, el máximo de la distancia hasta cada una de ellas es una estimación admisible de la distancia que aún hay que recorrer.

   De hecho, es posible mejorar esta estimación añadiendo, por ejemplo, la distancia hasta la estación central de vuelta, y calculando entonces el máximo de las sumas de la distancia hasta cada estación y desde ella hasta la central de vuelta.

   Esta misma idea puede refinarse añadiendo siempre una estación más. De hecho, la distancia de todos los arcos del árbol de recubrimiento mínimo (*Minimum Spanning Tree*) del grafo que relaciona todas las estaciones de trabajo y la estación central (ignorando únicamente las estaciones de carga) sería una heurística aún mejor.

5. La selección de un algoritmo de búsqueda informada para este caso depende, como en el caso del apartado 3, del número de transposiciones y también, naturalmente, de la dificultad de los problemas:

   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 al número de estaciones de trabajo). Además, también es un algoritmo de búsqueda admisible.

   Como ya se indicó anteriormente, este problema tiene un gran número de transposiciones y, por ello, se sugiere el algoritmo A*.

6. En el caso en el que deba gestionarse el movimiento de dos vehículos ocurrirá que el número de operadores que puedan aplicarse desde cada estado se multiplicará por dos, puesto que cada vehículo puede acceder a todas sus posiciones inmediatamente accesibles.

   Por lo tanto, el factor de ramificación máximo se multiplicará asimismo por 2.

7. En este caso, cada estado está caracterizado por la posición de dos vehículos, y ambos deben volver hasta la estación central. Si bien puede seguir usándose la misma idea que con un único vehículo para obtener una función heurística, no es posible hacerlo directamente puesto que cada estación debe ser inspeccionada por un único vehículo.

De las heurísticas consideradas tómese la primera de ellas: el máximo de las distancias hasta todas las estaciones pendientes de inspección. Si este cálculo se hiciera por separado para cada vehículo en más de un caso el máximo ocurriría para cada vehículo con respecto a la misma estación, de modo que se podría estar sobre estimando el esfuerzo para llegar hasta ella. En su lugar, dividir el valor heurístico calculado para cada vehículo por dos sí es una estimación admisible.

Y, de hecho, dividir entre dos cualesquiera de las heurísticas discutidas anteriormente es, naturalmente, también una función heurística admisible.