Intel·ligència Artificial

Col·lecció de Problemes

Solucions

Departament de Ciències de la Computació

Grau en Enginyeria Informàtica

Curs 2023/2024 1Q

UPC

FIB Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Departament de Ciéncies de la Computació

Facultat d'Informática de Barcelona

Universitat Politècnica de Catalunya

Imágenes creadas con Stable Diffusion

Primera edición, Septiembe 2008

Esta edición, septiembre 2023

Les solucions han estat elaborades pels professors Javier Béjar i Javier Vázquez.

També han elaborat algunes de les solucions els estudiants d'enginyeria informàtica:

Badia Orive, Esteve
Cami Muntane, Jordi
Iglesias Sanchez, Patricia
Lopez Cervilla, Javier
Padrol Sureda, Arnau
Ramos Garcia, Claudia
Tripiana Montes, Carlos
Tur Moreno, David

Responsable de la publicació: Javier Béjar (bejar@cs.upc.edu)

Índice general

I

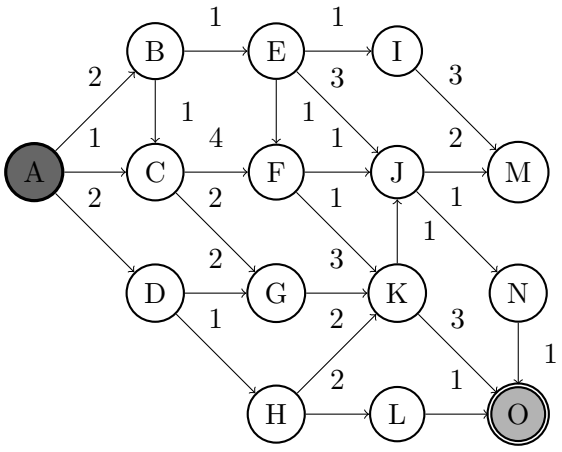
Cerca

1. Cerca Heurística

1.1 Problemas solucionados

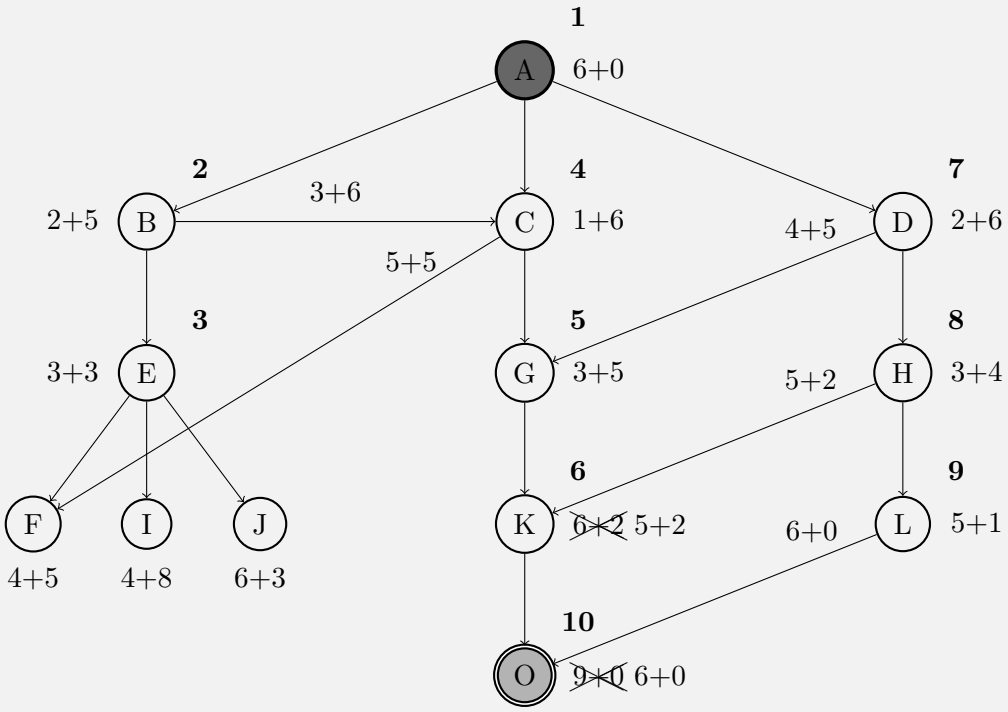
1.1.1 Problema 7

Dado el siguiente grafo donde cada arco indica su coste y la tabla que indica la estimación del coste h hasta la solución, indica cual sería el árbol de búsqueda que se obtendría mediante el algoritmo de A* e IDA* para encontrar el camino entre el nodo A y el nodo O. Haz la generación de los nodos siguiendo el orden alfabético e indica claramente las repeticiones de los nodos y los cambios de coste que aparezcan. ¿Es la función heurística admisible?



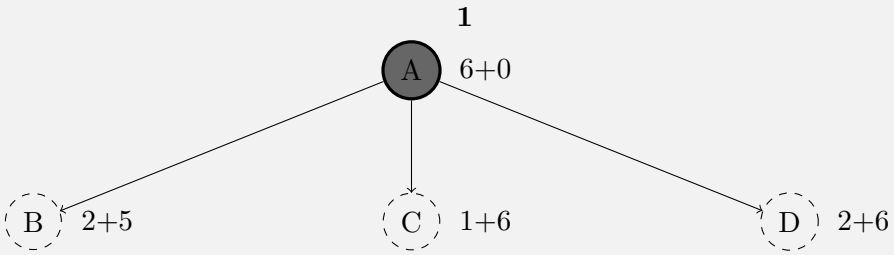
Nodo	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
h(nodo)	6	5	6	6	3	5	5	4	8	3	2	1	5	1	0

> Árbol de búsqueda con A*

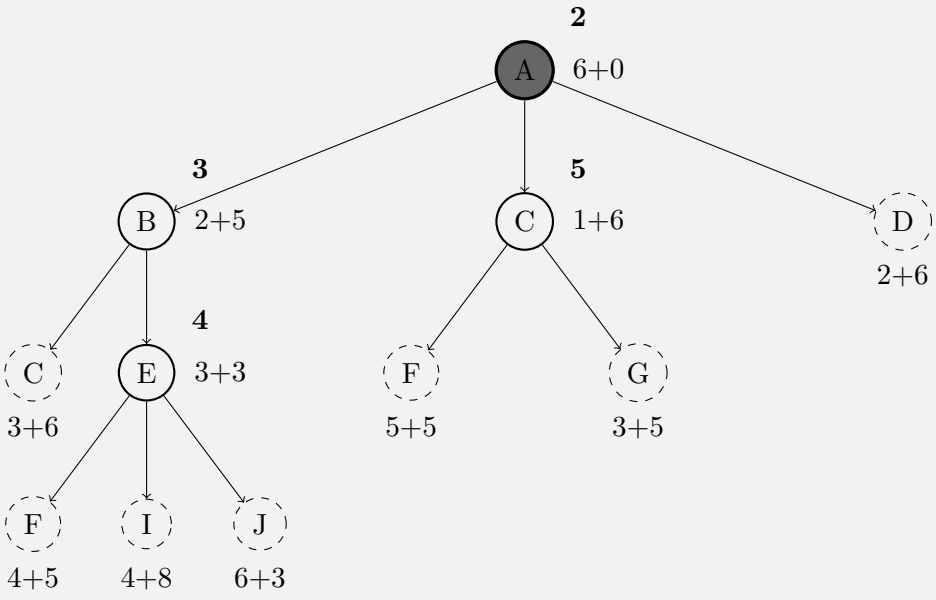


Árbol de búsqueda con IDA*

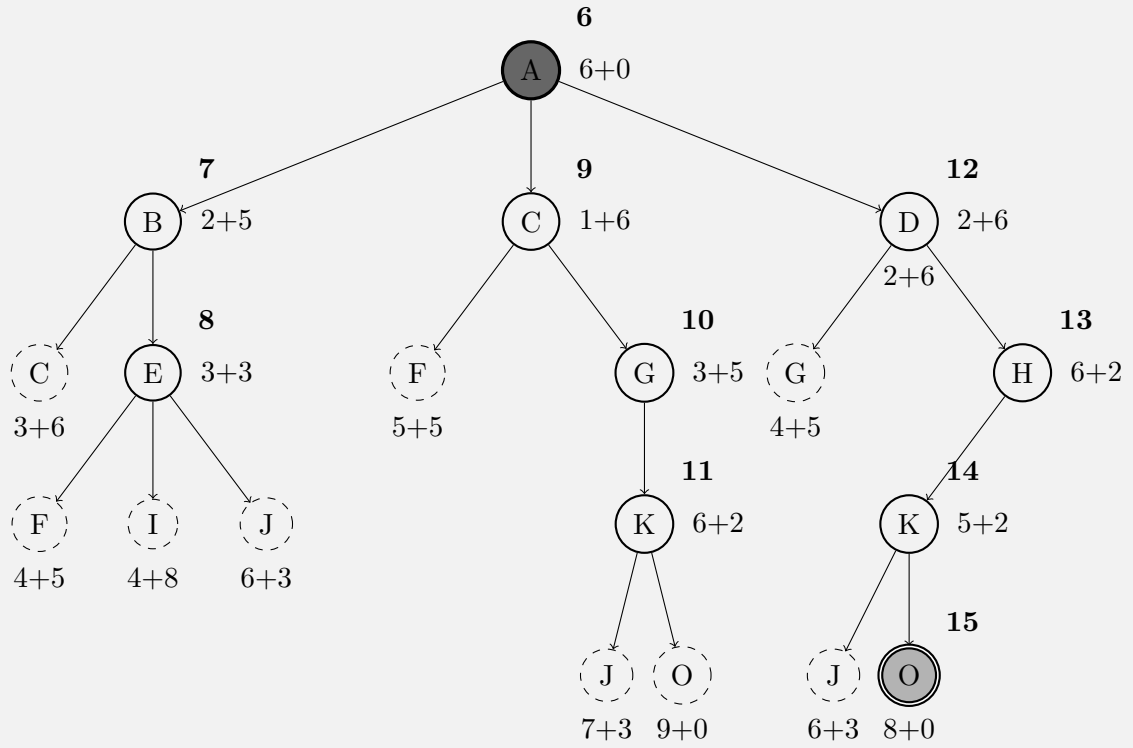
Profundidad 6



Profundidad 7



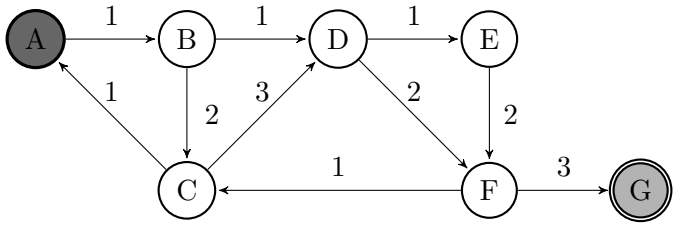
Profundidad 8



La función heurística no es admisible ya que en varias ocasiones el coste estimado es mayor que el coste real.

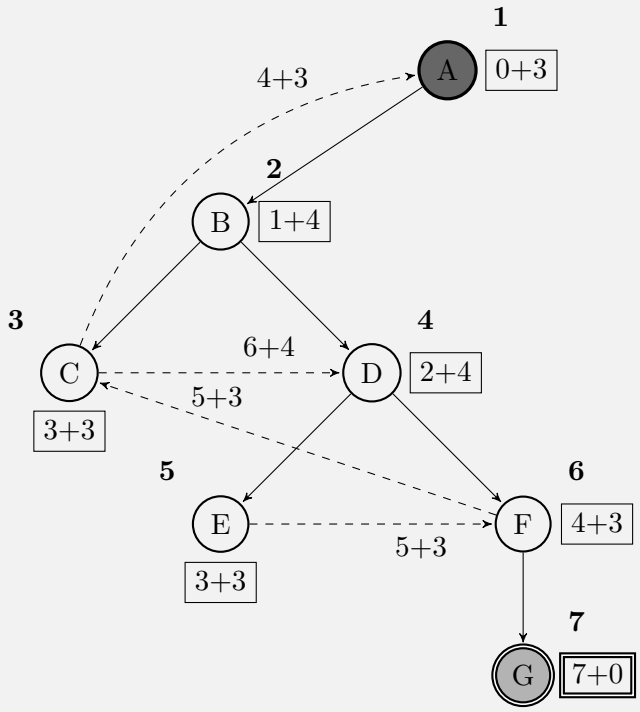
1.1.2 Problema 14

Dado el siguiente grafo donde cada arco indica su coste y la tabla que indica la estimación del coste h hasta la solución, indica cual sería el árbol de búsqueda que se obtendría mediante el algoritmo de A* e IDA* para encontrar el camino entre el nodo A y el nodo G. Haz la generación de los nodos siguiendo el orden alfabético e indica claramente las reexpansiones de los nodos y los cambios de coste que aparezcan. ¿Es la función heurística admisible?



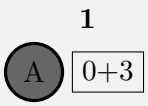
Nodo	A	B	C	D	E	F	G
h(nodo)	3	4	3	4	3	3	0

Árbol de búsqueda con A*

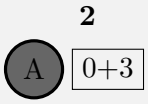


Árbol de búsqueda con IDA*

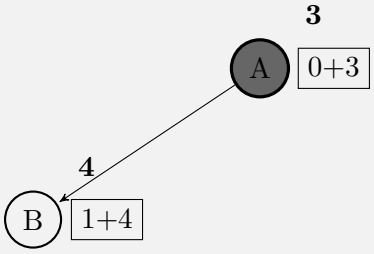
Profundidad 3



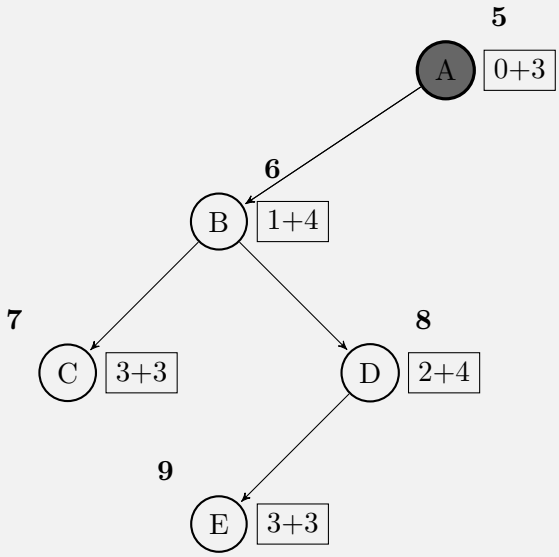
Profundidad 4



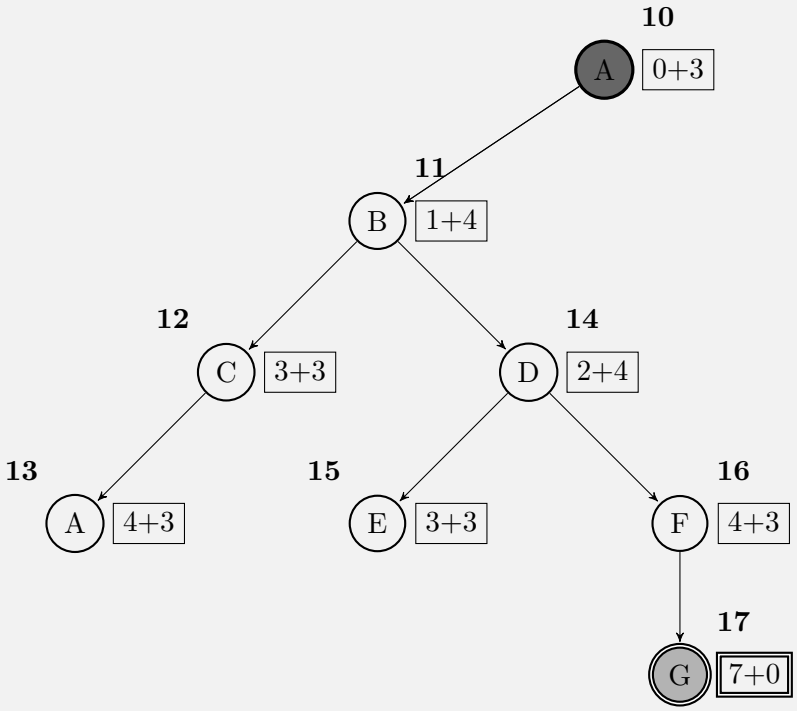
Profundidad 5



Profundidad 6



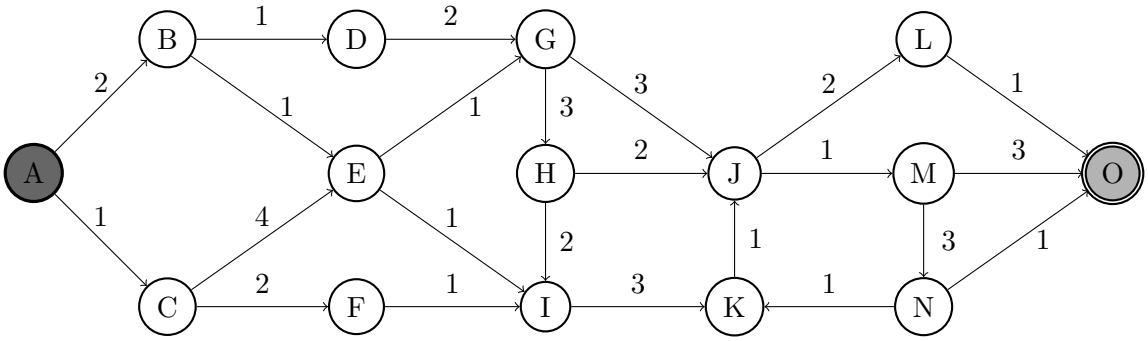
Profundidad 7



La función heurística es admisible. Se puede comprobar que para todos los valores de $h(n)$ que el coste real es siempre superior o igual.

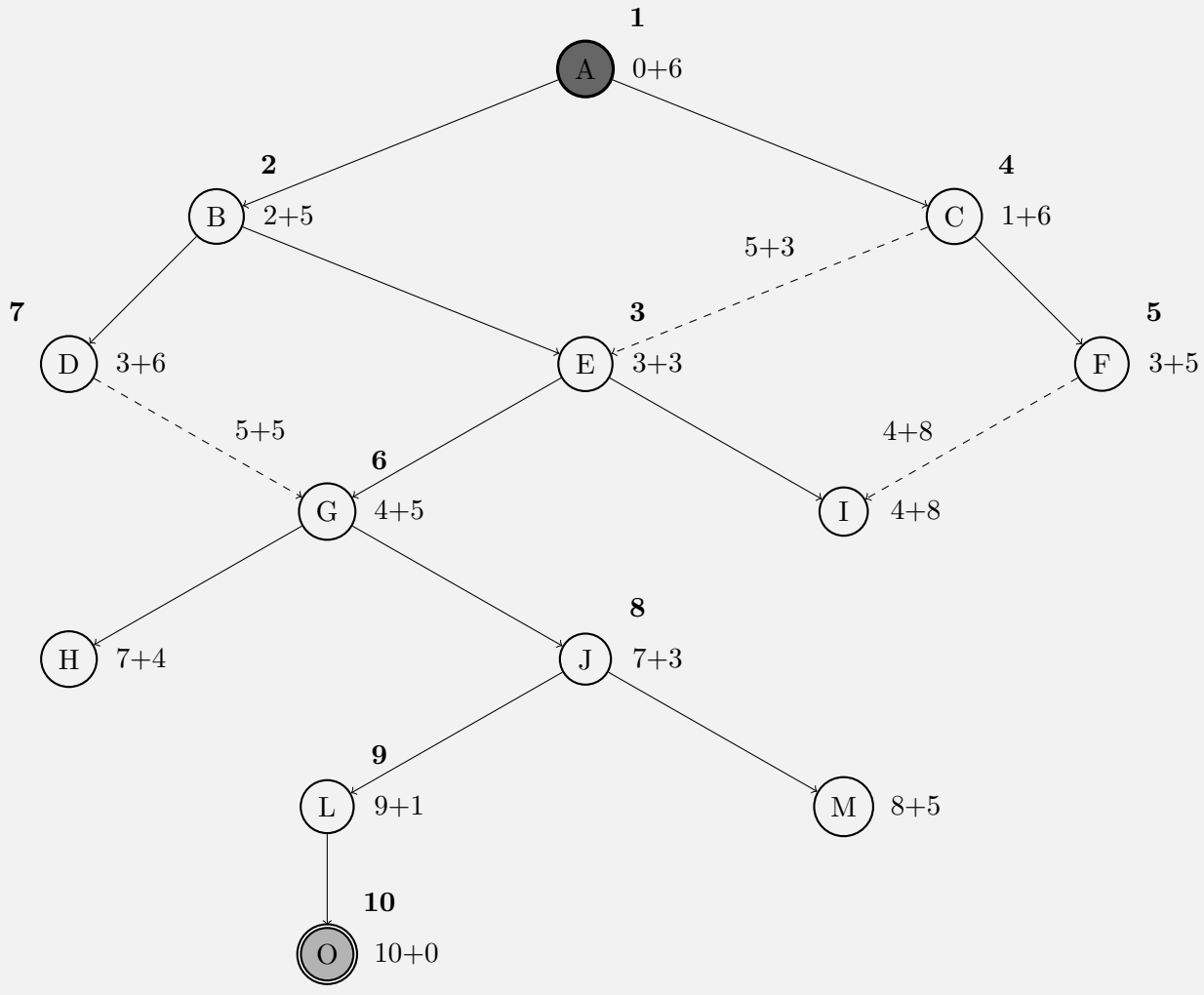
1.1.3 Problema 16

Dado el siguiente grafo donde cada arco indica su coste y la tabla que indica la estimación del coste h hasta la solución, indica cual sería el árbol de búsqueda que se obtendría mediante el algoritmo de A* e IDA* para encontrar el camino entre el nodo A y el nodo O. Haz la generación de los nodos siguiendo el orden alfabético e indica claramente las reexpansiones de los nodos y los cambios de coste que aparezcan. ¿Es la función heurística admisible?

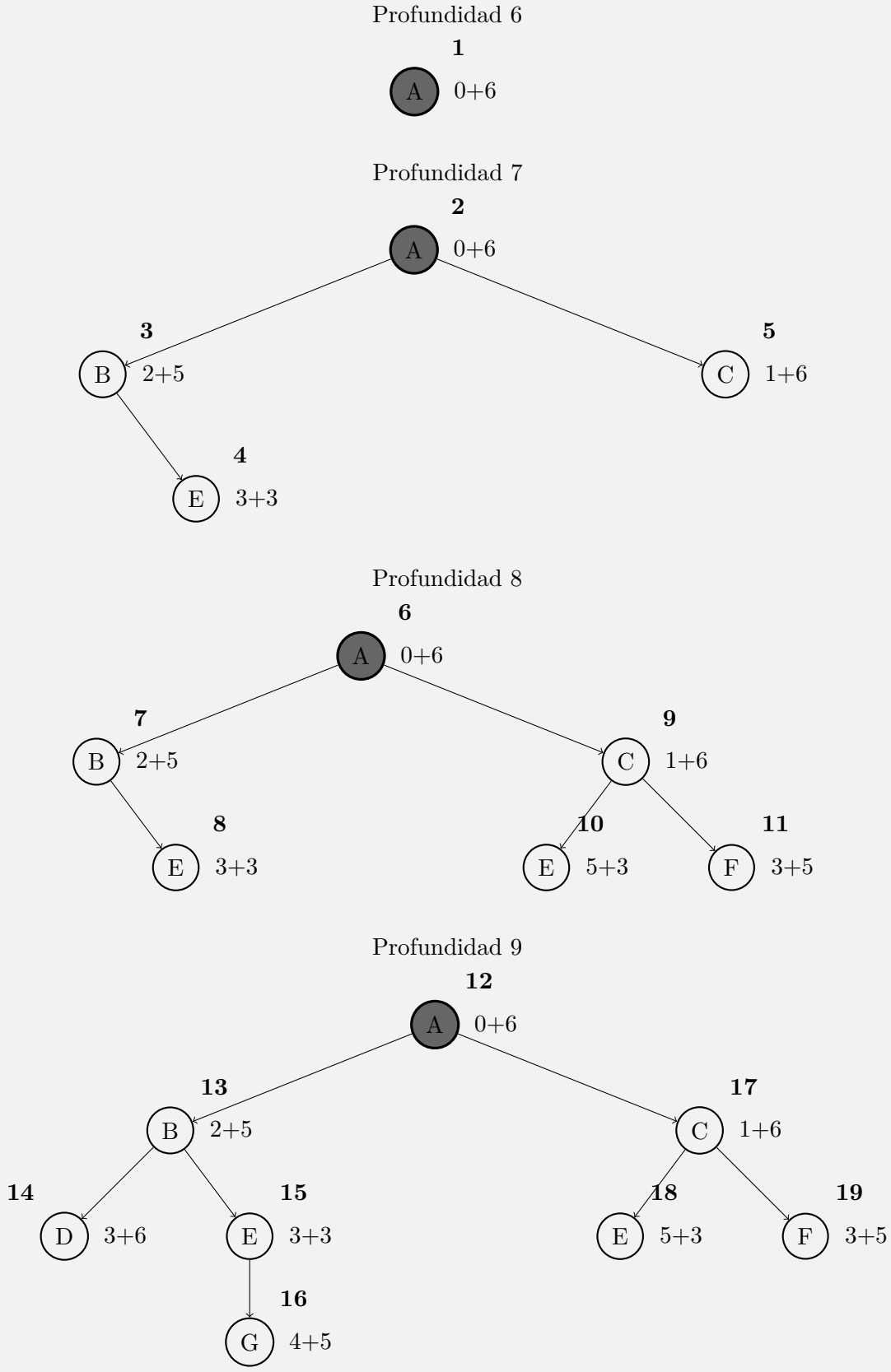


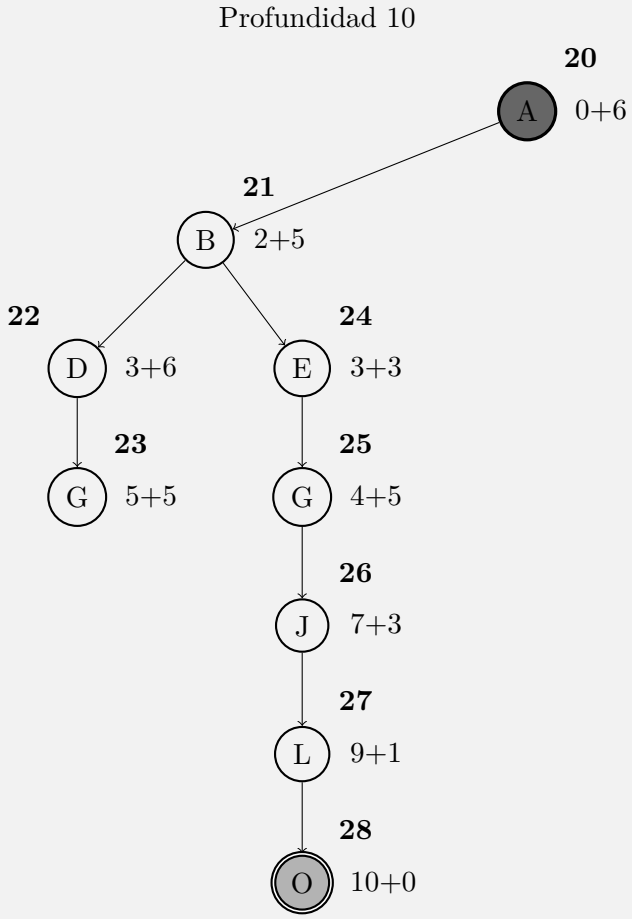
Nodo	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
h(nodo)	6	5	6	6	3	5	5	4	8	3	2	1	5	1	0

Árbol de búsqueda con A*



Árbol de búsqueda con IDA*

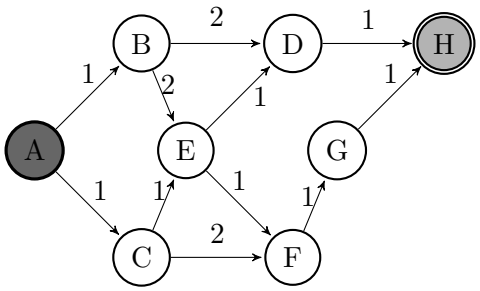




La función heurística no es admisible, se puede ver en el nodo M que tiene una h de 5 y un coste hasta la solución de 3.

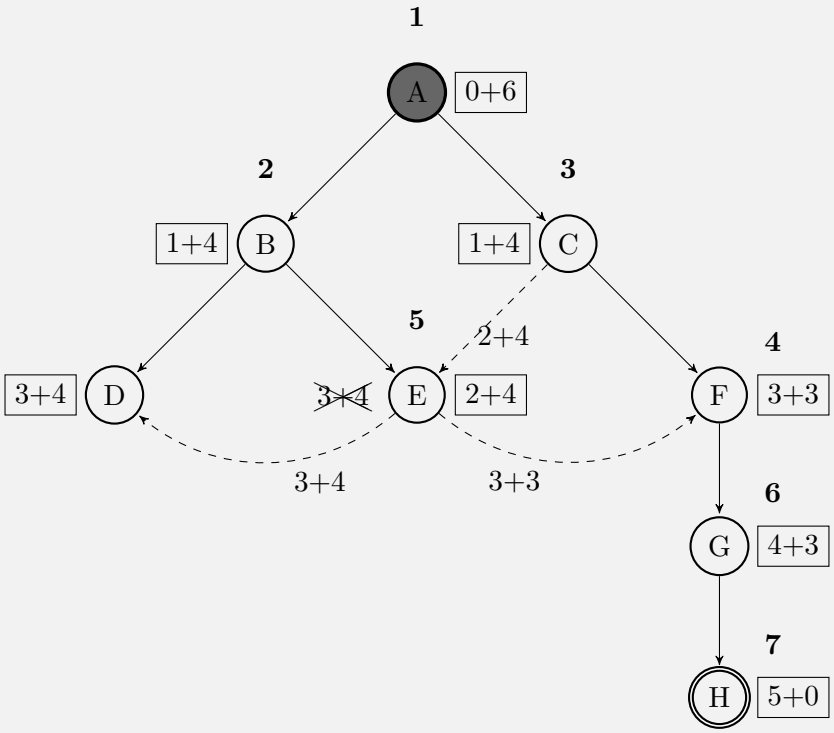
1.1.4 Problema 17

Dado el siguiente grafo, donde cada arco indica su coste, y la tabla que indica la estimación del coste h hasta la solución, indica cual sería el árbol de búsqueda que se obtendría mediante el algoritmo de A* e IDA* para encontrar el camino entre el nodo A y el nodo H. Haz la generación de los nodos siguiendo el orden alfabético e indica claramente las reexpansiones de los nodos y los cambios de coste que aparezcan. ¿Es la función heurística admisible?



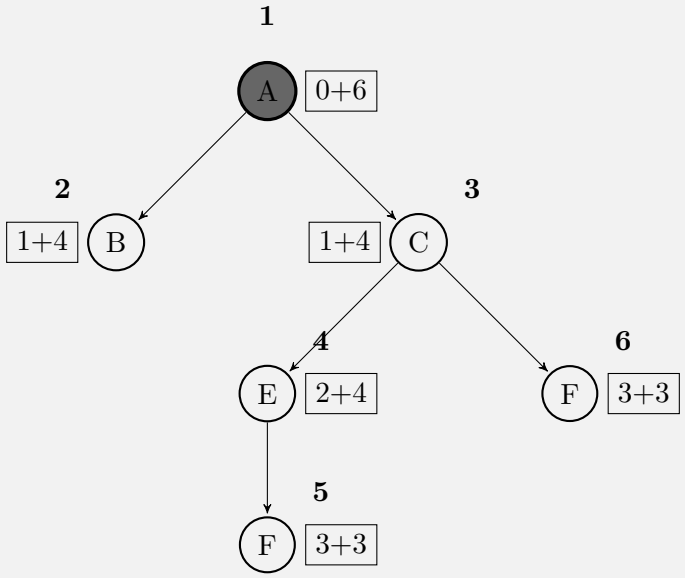
Nodo	A	B	C	D	E	F	G	H
h(nodo)	6	4	4	4	4	3	3	0

Árbol de búsqueda con A*

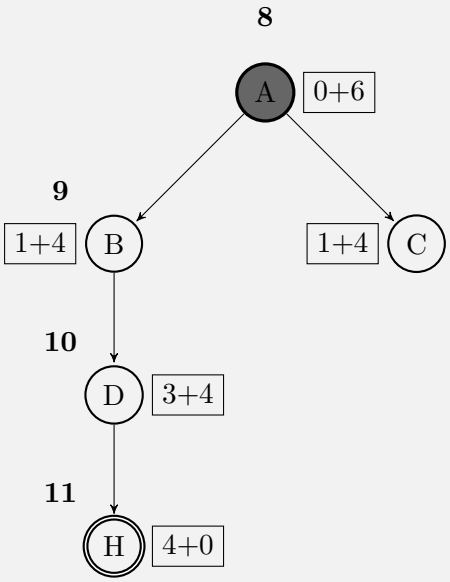


Árbol de búsqueda con IDA*

Profundidad 6



Profundidad 7



La función heurística no es admisible. Se puede ver que varios nodos dan una estimación superior al coste.

2. Cerca Local

2.1 Como plantear los problemas de búsqueda local

El objetivo de estos problemas es comentar las soluciones que se proponen a los diferentes elementos necesarios para resolver el problema de búsqueda que se plantea.

En un problema de búsqueda hay que decidir un conjunto de elementos:

- La representación del estado: Ha de ser válida y completa.
- La solución inicial: Ha de ser solución respecto a los criterios que plantea el problema, se debe evaluar el coste/posibilidad de obtenerla, si es necesario se deben poder obtener soluciones diferentes con el método propuesto, se debe evaluar su calidad.
- Operadores de búsqueda: Han de ser válidos, indicando unas condiciones de aplicabilidad y una función de transformación que genere soluciones, han de explorar correctamente el espacio de búsqueda, se ha de evaluar su factor de ramificación.
- Función heurística: Ha de incluir todos los objetivos del problema a optimizar, todos los objetivos han de ir en la sentido adecuado para optimizarlos, se ha de determinar si la ponderación de los diferentes objetivos es adecuada.

Al solucionar un problema mediante algoritmos genéticos hay que decidir también:

- La codificación del estado: Ha de ser válida y completa, han de poder representarse todos los estados.
- La población inicial: Se ha de poder generar mas de una solución inicial.
- Los operadores genéticos: Han de generar soluciones.

2.2 Problemas solucionados

2.2.1 Problema 6

Se han descubierto A fuentes de contaminación en un parque natural y se quieren colocar B (donde $B < A$) aparatos de descontaminación para mejorar la situación. Para ello se dispone de un mapa del

parque que indica la posición de la estación de trenes donde se han almacenado todos los aparatos y de los A lugares donde se necesita colocar los aparatos de descontaminación. Además también se dispone del nivel de contaminación que hay alrededor de cada fuente, de un mapa de los desplazamientos (dirigidos) posibles de los aparatos en el territorio y del coste de cada desplazamiento. Cada aparato puede eliminar por completo la contaminación de una fuente, independientemente de su nivel.

El objetivo es colocar los aparatos de manera que se minimice la contaminación total en el parque y el coste del recorrido (suma de desplazamientos) que harán los aparatos en el sentido "estación → fuente de contaminación".

En los siguientes apartados se proponen diferentes alternativas para algunos de los elementos necesarios para plantear la búsqueda (solución inicial, operadores, función heurística). Hay que comentar la solución que se propone respecto a si es correcta, es eficiente, y es mejor o peor en comparación con otras alternativas. Y hay que justificar todas las respuestas.

1. Se plantea solucionar el problema mediante Hill-Climbing, partiendo de una solución inicial sin ningún aparato y con un operador que coloca un aparato en una fuente de contaminación determinada, controlando que el número de los aparatos colocados sea como máximo B.

> La solución vacía no debería considerarse solución por que que no cumple las condiciones, ya que una solución debe asignar B aparatos.
>
> No obstante, podriamos obviar este problema dado el operador que tenemos. A cada paso se coloca un aparato en una fuente de contaminación, la única restricción es no superar el número de asignaciones máximo, esto nos garantiza que la exploración podrá pasar del espacio de no soluciones al de soluciones.
>
> Si el heurístico es bueno, partiendo de la solución vacía, este sería un buen operador ya que a cada paso del HC se escogerá la estación que más convenga hasta que se hayan puesto los B aparatos o ya no merezca la pena poner más (porque lo que se minimiza en contaminación no lo vale con el coste de desplazamiento de más que tenemos que hacer). El factor de ramificación del operador será el número de fuentes de contaminación por el número de aparatos disponibles ($O(A \cdot B)$).

2. Se plantea solucionarlo mediante Hill-Climbing, partiendo de una solución inicial con B aparatos colocados aleatoriamente, y utilizando como función heurística la suma de los costes de desplazamiento de la estación a cada una de las B fuentes.

> La solución es correcta puesto que respeta la restricción de que debe haber de B aparatos. También es barata, pero, al ser al azar, no tenemos ninguna garantía sobre su calidad.
>
> La función heurística no es correcta, ya que solo tiene en cuenta el coste del desplazamiento y no tiene en cuenta la contaminación.

3. Se plantea solucionarlo mediante Hill-Climbing, partiendo de una solución inicial con B aparatos colocados aleatoriamente, y utilizando como función heurística la suma de los costes mínimos de los recorridos "estación → fuente de contaminación" multiplicada por la suma de los niveles de contaminación en correspondientes a los B aparatos.

> Solución inicial, mismos comentarios que en el apartado anterior.
>
> La función heurística no es correcta, ya que queremos minimizar el primer multiplicando mientras que queremos maximizar el segundo. Tal como esta la funcion, si la minimizaramos colocariamos los aparatos en los puntos de menor contaminación. Una posible solución sería restar la contaminación al recorrido, pero la diferencia de unidades entre los dos factores podría asignar más peso a un factor que al otro. Otra solución sería pasar el segundo mutiplicando a dividir, de esta forma, cuanto más grande sea el denominador, más pequeño será el resultado.

4. Se plantea solucionarlo mediante Hill-Climbing, partiendo de una solución inicial alcanzada colocando los B aparatos ordenadamente según el coste mínimo "estación → fuente de contaminación" y empezando con el que tiene coste menor. Se plantea como operador mover un aparato a cualquier fuente cuyo producto de "coste mínimo estación → fuente" por "nivel de contaminación" sea menor que el actual.

> Esta solución inicial es mas costosa que las anteriores, ya que requiere ordenar las fuentes, concretamente el coste es $O(A \cdot log(A))$. Dado que el coste de desplazamiento no es el único criterio a optimizar utilizar solo el coste de desplazamiento no nos garantiza nada sobre la bondad de la solución inicial respecto al objetivo.
>
> El operador es correcto respecto a respetar la restricción de colocar B aparatos. La solución inicial parte de B aparatos colocados y solo cambiamos aparatos de sitio, no ponemos ni quitamos ninguno. En este caso el factor de ramificación sería $O(A \cdot B)$. No obstante, el operador estaría optimizando la suma de los productos para cada fuente de contaminación de su distancia y nivel de contaminación, ya que prefiere movimientos en los que el producto de estos dos valores es menor, esto minimizaría los recorridos, pero también escogería los puntos de menos contaminación, que es precisamente lo contrario de lo que queremos.

5. Se plantea solucionarlo mediante algoritmos genéticos: se usan individuos de A bits y como población inicial se generan n individuos donde en cada uno hay exactamente B bits a 1. La función de idoneidad es la suma de los costes mínimos "estación → fuente de contaminación" más la contaminación total residual del parque multiplicada por una constante. Como operadores se usan los habituales de cruce y mutación.

> La población inicial es correcta ya que cada individuo representa una solución diferente. El coste de hallar la solución inicial es $O(n)$, siendo n el número de individuos.
>
> La función de idoneidad es correcta ya que queremos minimizar el coste por desplazamiento y también la contaminación del parque, en este caso usamos la contaminación que no se limpia. El hecho de multiplicar la contaminación por una constante dará más peso a la contaminación, que ya parece lo adecuado.
>
> El operador Cruce no es correcto, ya que podría pasar que al cruzar dos individuos, el individuo resultante tuviera más o menos de B bits a 1, y eso sería una solución no válida.
>
> El operador Mutación tampoco es correcto ya que siempre nos dará soluciones con un bit mas o un bit menos que B.

2.2.2 Problema 8

La International Telecomunications Union (ITU) es un agencia de Naciones Unidas que regula el uso de las tecnologías de la telecomunicación y entre otras cosas coopera en la asignación de órbitas para satélites. El incremento del uso de satélites de todo tipo ha llevado a buscar métodos automáticos para asignar posiciones en órbitas que mantengan la seguridad de los satélites y la calidad de su funcionamiento.

Para solucionar el problema, se ha dividido la órbita geosincrónica en regiones donde se asignarán las posiciones orbitales de los satélites. Obviamente, una región del espacio tiene un volumen. Se quiere resolver el problema para cada región.

Hay S diversos tipos de satélites especializados que se desean poner en órbita (televisión, telefonía, científicos, militares, ...). Para cada satélite se tienen dos informaciones, el espacio de seguridad (volumen necesario para que el satélite pueda navegar con el mínimo riesgo de colisión y pueda alimentar sus paneles solares) y el coste de alquiler de la órbita que cobra la ITU al dueño del satélite (este precio depende de cada satélite y no es una función del espacio de seguridad).

Las restricciones que tiene la ITU son que la suma del volumen que han de ocupar los espacios de seguridad de los satélites ha de cubrir como mínimo 1/3 del volumen de la región si se quieren

cubrir las necesidades existentes, pero no puede superar los 2/3 si no se quieren tener problemas con otros lanzamientos que deben atravesar la órbita geosincrónica o con la basura espacial. También se ha de cubrir un cupo mínimo m_i para cada tipo de satélites. El objetivo del problema es maximizar la ganancia que obtiene la ITU con el alquiler de órbitas.

En los siguientes apartados se proponen diferentes alternativas para algunos de los elementos necesarios para plantear la búsqueda (solución inicial, operadores, función heurística, ...). El objetivo es comentar la solución que se propone respecto a si es correcta, es eficiente, o es mejor o peor respecto a otras alternativas posibles. Justifica tu respuesta.

1. Se plantea usar Hill-climbing, como solución inicial consideramos que asignamos aleatoriamente al menos la mínima cantidad de satélites para cubrir la cuota de cada tipo hasta ocupar 1/3 del volumen de la región. Disponemos del operador `añadir-satélite()` que comprueba el posible exceso de ocupación en la región y del operador `quitar-satélite()` que comprueba la posible infra-ocupación de la región. Como heurística usamos la suma para todos los satélites en la solución del producto entre el espacio de seguridad del satélite y el coste de alquiler del satélite.

   > La solución inicial es correcta (cumple las restricciones de volumen y de cupo), hacer la selección de manera aleatoria no nos garantiza nada sobre la calidad de la solución inicial. El coste de elaborar la solución es lineal.
   >
   > Los operadores permiten generar todas las soluciones posibles, pero el operador de quitar satélite no comprueba que todavía haya en la solución el número mínimo de satélites del tipo de satélite que quita. El factor de ramificación de los operadores es lineal.
   >
   > La función heurística incluye el espacio de seguridad como criterio cuando solo se pide maximizar la ganancia del alquiler. Esto podría considerarse correcto teniendo en cuenta que dependiendo del encaje de los espacios la ganancia que se puede obtener es diferente. No obstante, multiplicar el espacio por el coste de alquiler daría mayores valores a los satélites que necesitaran mucho espacio y de hecho nos haría falta dar prioridad a las soluciones con satélites que necesiten poco espacio.

2. Se plantea usar Hill-climbing, como solución inicial ordenamos los satélites descendentemente por el coste del alquiler y vamos añadiendo los satélites en ese orden hasta llenar 2/3 del volumen. Como operador utilizamos cambiar un satélite de un tipo por otro del mismo tipo que no esté ya en la solución. Como función heurística usamos la suma del espacio de seguridad de todos los satélites en la solución multiplicado por la suma del coste del alquiler de todos los satélites.

   > La solución inicial podría no respetar los cupos mínimos de satélites de cada tipo. El coste de generar la solución es $O(n\log(n))$ debido a que hay que ordenar los satélites. El generar la solución de este modo podría generar soluciones iniciales buenas, pero no tenemos ninguna garantía ya que el poder colocar los satélites depende también del espacio que necesiten.
   >
   > Con el operador no podemos generar todas las soluciones ya que no podremos cambiar el número de satélites de un tipo que tenemos en la solución inicial. El operador además no comprueba ninguna de las restricciones del problema. El factor de ramificación del operador es cuadrático.
   >
   > La función heurística vuelve a incluir el espacio de seguridad como criterio. Igual que antes podría ser correcto considerando que la ganancia dependerá de que podamos encajar más satélites. El multiplicar los dos criterios dará soluciones que compensen el valor de los dos, pudiendo ser buenas soluciones que combinen los dos valores de maneras diferentes (mucho espacio y poco coste o al revés), cuando el criterio que realmente nos interesa es el coste del alquiler.

3. Se plantea usar algoritmos genéticos. Un individuo es una tira de bits cuya longitud es el número total de satélites que tenemos. Si el bit está a 1 significa que el satélite está en la solución y si

está a 0 es que no lo está. La población inicial se genera creando individuos que tengan a 1 los bits de los m_i satélites con mayor coste de alquiler de cada tipo. Como operadores se usan los habituales de cruce y mutación. Como función heurística usamos la suma del coste de alquiler de todos los satélites en la solución.

> La representación del problema es correcta, podemos representar cualquier solución del problema.
>
> La generación de la población inicial tiene varios problemas, el primero es que no tenemos garantía de cuanto espacio ocupará la solución, podemos pasarnos o quedarnos cortos. El segundo problema es que de la manera que generamos la población todos los individuos serán iguales.
>
> Los operadores habituales de cruce y mutación no nos garantizarán el mantener las restricciones de la solución.
>
> La función heurística optimiza el coste de alquiler de los satélites, que es lo que nos interesa, pero el espacio que necesita un satélite influye en el coste que podemos alcanzar, por lo que sería una buena idea incluirlo de alguna manera.

2.2.3 Problema 9

Tenemos una pequeña flota de C camiones que utilizamos para repartir mercancías y cada día tenemos que determinar qué ruta ha de seguir cada camión para abastecer un conjunto de ciudades por todo el país. El objetivo es que todos los camiones acaben la jornada aproximadamente a la misma hora, por lo que el número de kilometros que ha de recorrer cada camión ha de ser muy parecido, y que recorran en total el mínimo número de kilómetros.

Disponemos de un mapa de carreteras que nos dice la distancia en kilómetros entre cada ciudad donde hemos de dejar nuestra mercancía, suponemos que todos los camiones parten de la misma ciudad y han de volver a ella al final del día, cargan al principio de la jornada todo lo que han de repartir, han de pasar sólo una vez por cada ciudad.

Una posible solución a éste problema se puede obtener mediante el uso de algoritmos de búsqueda local. En los siguientes apartados se proponen diferentes alternativas para algunos de los elementos necesarios para plantear la búsqueda (solución inicial, operadores, función heurística,...). Comenta muy brevemente la solución que se propone respecto a si es correcta y si es mejor/peor respecto a otras alternativas posibles. Justifica tus respuestas.

1. Usar Hill Climbing. Como solución inicial asignamos al azar a cada camión un número aproximadamente igual de ciudades, recorriéndolas en orden también al azar. Como operadores usamos el intercambiar dos ciudades entre los recorridos de dos camiones e intercambiar las posiciones de dos ciudades en el recorrido de un camión. La función heurística es la siguiente:

$$h'(n) = \sum_{i=1}^{C} \left(\frac{LR_i}{\sum_{j=1}^{C} LR_j} - \frac{1}{C} \right)$$

donde LR_i es la longitud del recorrido del camión i.

> Por lo general, la aleatoriedad nos genera una solución inicial de forma rápida, aunque la bondad de esta solución será variable en cada ejecución. El balanceo de carga de los recorridos, respecto del número de ciudades por recorrido se cumple, pero eso no implica necesariamente que se cumpla por longitud de los recorridos, que sería lo ideal. Por lo tanto, es importante tener un buen heurístico que nos permita mejorar este aspecto. El heurístico está calculando lo siguiente:
>
> $$h'(n) = \sum_{i=1}^{C}(\frac{LR_i}{\sum_{j=1}^{C} LR_j} - \frac{1}{C}) = [\sum_{i=1}^{C}(\frac{LR_i}{\sum_{j=1}^{C} LR_j})] - 1 = \frac{\sum_{i=1}^{C} LR_i}{\sum_{j=1}^{C} LR_j} - 1 = 1 - 1 = 0.$$

Este heurístico no está dándonos ninguna información, pese a lo que inicialmente pareciera. Dado que la bondad siempre es cero, ninguna solución sucesora de la inicial podría mejorarlo y por solución final se obtendría la inicial. De esta forma, el resultado es obtenido aleatoriamente.

Con respecto a los operadores de transformación hay que tener en cuenta que *2.* siempre mantendrá los recorridos balanceados (salvo sobre el que se aplique), que es lo deseado por lo general. *1.* mantiene el balanceo en número de ciudades por recorrido, pero no necesariamente en longitud (sobre los dos recorridos implicados).

Factores de ramificación:

a) Si tenemos un recorrido de un camión C_i y otro C_j con longitud del recorrido L_{C_j}, en número de ciudades, el factor de ramificación de *1.* con origen en C_i y destino C_j es de $FR_{Op_1} = N - L_{C_j}$. Para toda ciudad es, entonces, del orden de $O(N^2)$.

b) Si L_{C_i} es la longitud, en cada momento, del recorrido del camión C_i, en número de ciudades, el factor de ramificación es de $FR_{Op_2} = L_{C_i} - 1$. El rango es $0 \leq FR_{Op_2} \leq N - 1$. Para toda ciudad es, entonces, del orden de $O(N^2)$.

El problema que se nos plantea con estos dos operadores es la localidad. Dada una solución inicial, las soluciones derivadas de esta son las derivadas de los posibles intercambios. Esto nos mantiene, siempre, la longitud de los recorridos en número de ciudades, como ya se ha dicho, lo que supone que el resto del espacio de estados queda fuera del potencial de las operaciones e inexplorado.

2. Usar Hill Climbing. Como solución inicial asignamos todas las ciudades a un camión, estableciendo el recorrido inicial mediante una estrategia avariciosa de manera que intentemos minimizarlo. Como operadores usamos el mover una ciudad del recorrido de un camión a otro e intercambiar las posiciones de dos ciudades en el recorrido de un camión. La función heurística es la siguiente:

$$h'(n) = \prod_{i=1}^{C} LR_i$$

El uso de una técnica avariciosa supondrá un coste cuadrático para hallar el primer recorrido, eligiendo, a cada paso, el menor de los valores posibles para las ciudades sin asignar. Al estar todas las ciudades asignadas a un mismo camión, la solución está muy desbalanceada (en todos los aspectos), y eso no se desea. Para encontrar una solución buena debemos alejarnos mucho en el espacio de estados de esta solución inicial, por lo que tendrá que ser destruida completamente, haciendo que el coste de hallarla se puerda completamente.

Para modificar este estado y pasar a uno más balanceado, deberemos usar un heurístico que nos guíe en ese camino, pero inicialmente la solución inicial será, por lo general, bastante mala.

El heurístico está calculando lo siguiente:

$$h'(n) = \textstyle\prod_{i=1}^{C} LR_i$$

Y eso quiere decir que si algún $LR_i = 0 \rightarrow h'(n) = 0$. Y como en la solución inicial todos los recorridos son vacíos salvo uno, eso quiere decir que el heurístico será cero, y ninguna otra solución sucesora puede mejorar ese valor por lo que la solución inicial será la final.

Con toda seguridad la solución de este método será peor que la encontrada por el apartado *a)*, ya que en el apartado *a)* la aleatoriedad da cierta posibilidad a obtener mejores resultados.

Con respecto a los operadores de transformación hay que decir que la aplicación sucesiva del operador *1.* podría desbalancear una solución. Pese a esto, este operador puede resultar útil para contrarrestar un efecto de desbalanceo, y como la solución inicial está muy desbalanceada este operador puede ser muy útil. Con el operador *2.*, que realiza un intercambio sobre el mismo recorrido, no se altera el balanceo del resto.

Factores de ramificación:

a) Si tenemos tenemos C recorridos de camiones y un recorrido de un camión C_i, el factor de ramificación de *1.* con origen en C_i y destino cualquier C_j, $i \neq j$, es de $FR_{Op_1} = C - 1$. Así que para toda ciudad sería del orden de $O(N * C)$.

b) Si L_{C_i} es la longitud, en cada momento, del recorrido del camión C_i, en número de ciudades, el factor de ramificación es de $FR_{Op_2} = L_{C_i} - 1$. El rango es $0 \leq FR_{Op_2} \leq N - 1$. Para toda ciudad es, entonces, del orden de $O(N^2)$.

En esta selección de operadores no aparece el problema de la localidad ya que el operador *1.* varía las longitudes en número de ciudades por recorrido lo que permite explorar todas las posibles combinaciones de ciudades a recorridos. Cabe destacar que el operador *2.* por su parte sigue teniendo el mismo problema.

3. Usar Hill climbing. Como solución inicial escogemos las C ciudades más cercanas a la ciudad origen como la primera ciudad a visitar por cada camión, como segunda ciudad en el recorrido de cada camión escogemos la más cercana a la primera que no esté ya asignada, y así sucesivamente hasta asignar todas las ciudades. Como operadores usamos el mover una ciudad del recorrido de un camión a otro, intercambiar las posiciones de dos ciudades en el recorrido de un camión e intercambiar dos ciudades entre los recorridos de dos camiones. La función heurística es la siguiente:

$$h'(n) = \frac{C \cdot (C-1)}{2} - \sum_{i=1}^{C} \sum_{j=i+1}^{C} \frac{LR_i}{LR_j}$$

La solución inicial estará balanceada en número de ciudades, no esta claro que la forma de generar el recorrido balancee las distancias.

Aplicar este sistema para crear la solución inicial es identico en el caso peor al anterior, la primera asignacion requiere escoger las C ciudades más cercanas a la ciudad origen ($O(nlog(N))$ y luego hay que escoger para cada ciudad que asignemos a cada recorrido su ciudad más cercana ($O(N^2)$)

En el heurístico el primer elemento de la resta es un factor constante por lo que no tiene ningún efecto aparte de establecer el rango. El segundo elemento debería acercarse al valor del primero cuanto más equilibrados estén los recorridos en longitud, que es lo que realmente se desea. Por lo tanto, este heurístico dará cero cuando esté totalmente equilibrada la solución.

El problema es que, la solución obtenida, pese a estar equilibrada no tiene porque ser de longitud mínima, lo que hace el heurístico es intentar equilibrar la proporción de las longitudes de cada recorrido.

Los tres operadores son correctos y generan soluciones, permitiendo generar todo el espacio de búsqueda. El operador intercambiar equivale a dos operaciones de mover, pero puede ser necesario ya que es posible que no se puedan aplicar las dos operaciones separadamente y si como una operacion de intercambio.

Factores de ramificación:

a) Si tenemos tenemos C recorridos de camiones y un recorrido de un camión C_i, el factor de ramificación de *1.* con origen en C_i y destino cualquier C_j, $i \neq j$, es de $FR_{Op_1} = C - 1$. Así que para toda ciudad sería del orden de $O(N * C)$.

b) Si tenemos un recorrido de un camión camión C_i y otro C_j con longitud del recorrido L_{C_j}, en número de ciudades, el factor de ramificación de *1.* con origen en C_i y destino C_j es de $FR_{Op_1} = N - L_{C_j}$. Para toda ciudad es, entonces, del orden de $O(N^2)$.

c) Si L_{C_i} es la longitud, en cada momento, del recorrido del camión C_i, en número de ciudades, el factor de ramificación es de $FR_{Op_2} = L_{C_i} - 1$. El rango es $0 \leq FR_{Op_2} \leq N - 1$. Para toda ciudad es, entonces, del orden de $O(N^2)$.

> En esta selección de operadores no aparece el problema de la localidad ya que el operador *1.* varía las longitudes en número de ciudades por recorrido lo que permite explorar todas las posibles combinaciones de ciudades a recorridos. Cabe destacar que el operador *2. y 3.* por su parte siguen teniendo el mismo problema.

4. Usar algoritmos genéticos. Donde cada ciudad está representada por tantos bits como sean necesarios para codificar el valor C, la tira de bits contiene concatenados los bits de todas las ciudades, es decir, representamos en la tira de bits el número del camión que ha de recorrerla. Como operadores utilizamos los operadores habituales de cruce y mutación.

> La codificación escogida no representa el orden en que un camión va a recorrer las ciudades que se le han asignado. En este problema usar una codificación binaria es difícil si se quiere poder representar el orden de recorrido de las ciudades ya que cada recorrido tendrá una longitud variable.
>
> Además matener la coherencia de la representación al aplicar los operadores genéticos sería bastante difícil.

2.2.4 Problema 10

Los dueños del hotel "MiniPreu" quieren reducir su presupuesto de limpieza para maximizar sus ganancias. Para hacerlo han decidido tener tres tipos diferentes de precios para sus habitaciones. El precio A incluye una limpieza diaria de la habitación y tiene un precio de *pa* euros por día, el precio B incluye una limpieza de la habitación en días alternos y tiene un precio de *pb* euros por día y el precio C solo incluye una limpieza el primer y último día y tiene un precio de *pc* por día. El mínimo número de días que se pueden reservar para una habitación es de dos, y asumiremos que la limpieza de las habitaciones de precio B esta organizada de manera que la última limpieza se hace el día en el que el cliente deja su habitación. Tenemos un total de R habitaciones en el hotel.

Limpiar una habitación de precio A cuesta *ca* euros por día de limpieza, una habitación de precio B cuesta *cb* por día de limpieza y una habitación de precio C cuesta *ca* más el número de días que la habitación ha sido reservada multiplicado por el coste *cc*. Las habitaciones que están vacías no se limpian.

El departamento de sanidad impone como restricción que toda habitación (ocupada o no) ha de limpiarse al menos 10 veces al mes. Esto fuerza a que ninguna habitación pueda quedar sin ocupar muchos días.

El hotel recibe peticiones de reserva para un mes completo, cada reserva incluye el precio que el cliente quiere, el día en el que la reserva comienza y el número de días a reservar. Asumiremos que todas las peticiones comienzan y finalizan dentro del mes. El objetivo es decidir qué reservas aceptar, de manera que los gastos de limpieza se minimicen y las ganancias se maximicen.

En los siguientes apartados se proponen diferentes alternativas para algunos de los elementos necesarios para plantear la búsqueda (solución inicial, operadores, función heurística, ...). El objetivo es comentar la solución que se propone, analizando si la técnica escogida es adecuada para este problema, si cada uno de los elementos de la solución son correctos o no (cada uno por separado y en conjunción los unos con los otros). Incluye un análisis de los costes algorítmicos y/o factores de ramificación allá donde sea necesario. Justifica tu respuesta.

a) Queremos usar Hill-climbing, la solución inicial es la solución vacía. Como operadores de búsqueda usamos `añadir-reserva` que añade una reserva de una habitación, `cambiar-reserva` que cambia la reserva de una habitación por otra reserva que no esté ya en la solución. La función heurística es la suma de todos los costes de limpieza de las habitaciones multiplicado por las ganancias obtenidas por las reservas.

Plantear como solución un Hill Climbing para este problema es adecuado a priori, ya que nos piden encontrar una solución maximizando o minimizando una serie de criterios sin necesidad de obtener el óptimo.

La solución inicial planteada no es solución (las habitaciones se han de limpiar al menos 10 veces al mes y en una solución vacía ninguna habitación es limpiada). El coste de generación es $O(1)$ pero su calidad es mala ya que se está muy lejos de cualquier solución, dejando todo el trabajo de construir una primera solución al algoritmo de búsqueda. Podríamos empezar con esta solución si se asegura que el algoritmo de búsqueda acabará en una solución que cumpla las restricciones.

Los operadores de búsqueda permiten cubrir todo el espacio de soluciones ya que pueden generar cualquier configuración (solución o no) para el problema. De hecho el operador `añadir-reserva` ya nos asegura el poder construir potencialmente cualquier solución a partir de la solución vacia, pero al ser un Hill-Climbing que siempre escoge el mejor sucesor local, el operador `cambiar-reserva` permitirá ajustar la calidad de la solución sobretodo en las últimas iteraciones de la búsqueda. Sin embargo estos operadores pueden pasar de una solución a una no-solución, ya que no mantienen la restricción de que cada habitación se ha de limpiar más de 10 veces al mes, y no comprueban si las reservas se solapan con las que ya están en la solución. Ambos operadores deberían comprobar siempre si es posible añadir una reserva en la habitación específica (no hay otra reserva ya durante alguno de los días de la reserva nueva). La restriccion del mínimo de 10 limpiezas por habitación no se cumple al inicio de la búsqueda, por lo que es mejor tratarla directamente en el heurístico. El factor de ramificación de los operadores en el caso peor es el producto de las reservas y el número de habitaciones (en el caso de `añadir-reserva`) y el cuadrado de las reservas en el otro (en el caso de `cambiar-reserva`).

La función heurística asigna el mismo valor a soluciones que tienen un bajo coste de limpieza y gran beneficio que a soluciones con gran coste de limpieza y bajo beneficio, por lo que no puede distinguir las buenas soluciones de las malas. Minimizar este heurístico no es una buena idea ya que a menores valores menor es el beneficio. Además este heurístico debería incluir una penalización de las no soluciones que permita distinguir entre una no-solución mala y una mejor, de forma que pueda guiar la búsqueda por el espacio de no-soluciones dirigiéndola hacia el espacio de soluciones.

b) Queremos usar Hill-climbing, la solución inicial se obtiene de la siguiente manera: para una habitación elegimos una de las reservas que tenga la fecha más temprana, la siguiente reserva será la que tenga la fecha más temprana que empiece después del final de la última reserva asignada a la habitación, repetimos este procedimiento hasta que no podamos asignar más reservas a la habitación. Hacemos esto para todas las habitaciones. Como operadores de búsqueda usamos `cambiar-reserva` que cambia una reserva de una habitación por otra reserva que no este en la solución. La función heurística es la suma de todos los costes de limpieza de las habitaciones menos las ganancias obtenidas por las reservas.

Como se ha dicho en el apartado anterior, a priori Hill Climbing es adecuado para este tipo de problema.

La solución inicial puede ser no-solución, ya que no hay garantía de que la solución inicial cumpla la restricción de limpiar cada habitación más de 10 veces al mes (si hay pocas reservas podria ocurrir que todas se concentren en las primeras habitaciones, dejando las últimas habitaciones con pocas o ninguna reserva; pero incluso si hay más reservas de las qque se pueden colocar en las habitaciones, si en una habitación se concentran muchas reservas de tipo C de más de 3 días, esa habitación podria no limpiarse 10 veces al mes. No hay tampoco garantía sobre la calidad de la solución, ya que esta solución inicial es un greedy que solo se centra en llenar las habitaciones con el máximo número de reservas, pero no tiene en cuenta los tipos de precio de las habitaciones o los costes de limpieza y por lo tanto puede estar lejos de un máximo local (con respecto al beneficio

neto obtenido). El coste de generación de la solución inicial, asumiendo que ordenamos primero las reservas por fecha de inicio, es $O(N + logN) + N \times R$, siendo N el número de reservas y R el número de habitaciones, ya que una vez ordenadas las reservas por habitación, para cada habitación hemos de recorrer la lista de reservas que quedan para irlas seleccionando en orden de fecha.

El operador no puede generar todas las posibles soluciones ya que el número de reservas de la solución inicial no se puede cambiar. El operador no comprueba si la reserva cabe en el calendario de la habitación o que la habitación sea limpiada más de 10 veces al mes. En el caso extremo de que la solución inicial haya podido colocar todas las reservas en habitaciones, el operador no tendría reservas fuera de la solución para intercambiar. Es por todo ello que sería necesario anadir al menos un operador para `mover` una reserva de una habitación a otra. El factor de ramificación del operador en el peor caso es $O(N^2)$, siendo N el número de reservas.

La función heurística propuesta suma los costes y resta los beneficios. Minimizando la función heurística propuesta podemos obtener el tipo de soluciones que queremos. Sin embargo el heurístico no es correcto ya que debería incluir una penalización de las no soluciones.

c) Se plantea utilizar algoritmos genéticos. Asignamos a cada reserva un número de 0 a R, la concatenación de estos números para todas las reservas en binario es la codificación de una solución. El número 0 significa que la reserva no esta asignada a ninguna habitación, otro número representa el número de habitación asignada a la reserva. Para generar la población inicial obtenemos una solución asignando aleatoriamente una reserva a cada habitación (solo R reservas tienen un número diferente de 0). Como operadores genéticos usamos los operadores habituales de cruce y mutación. La función heurística es la suma para todas las reservas en la solución del cociente entre las ganancias obtenidas por la reserva y los costes de limpieza de la reserva.

Plantear como solución un algoritmo genético para este problema es adecuado a priori, ya que nos piden encontrar una solución maximizando o minimizando una serie de criterios sin necesidad de obtener el óptimo.

La codificación de las soluciones es una representación correcta del problema, es la habitación que ha de asignarse a cada reserva, y requiere solo $N \times logR$ bits, siendo N el número de reservas y R el número de habitaciones. La representación escogida permite representar cualquier solución del problema pero también muchas no-soluciones: identificadores de habitación que no existen, habitaciones con reservas solapadas, habitaciones sin un mínimo de 10 limpiezas.

La estrategia para generar la población obtiene diferentes soluciones iniciales, esto es correcto. Pero su variabilidad es limitada, ya que el número de reservas diferentes en la población inicial depende del tamaño de la población P (si el tamaño de la población es pequeño, solo unas pocas reservas serán utilizadas para obtener nuevos individuos). Otro problema es que las soluciones iniciales pueden no cumplir la restricción de limpiar las habitaciones más de 10 veces al mes ya que cada habitación tiene una sola reserva, y tanto si son reservas A o B de pocos dias como si es una reserva C de cualquier duración no cumpliran las limpiezas mínimas. El coste de generación es $O(P \times R)$, siendo P el tamaño escogido de la población de soluciones iniciales y R el número de habitaciones.

Los operadores habituales pueden generar soluciones y todos los tipos de no-solucuión que permite la representación: habitaciones asignadas a reservas que se solapan en el tiempo, identificadores de habitación que no existen, habitaciones que no cumplen la restricción de limpiar más de 10 veces al mes, etc. En este caso el operador de mutación genera soluciones muy próximas a las originales (solo cambia el identificador de habitación de una reserva en un bit) y es el operador de cruce el que genera sucesores diferentes a los padres (pero no muiy lejos de ellos).

> La función heurística aparentemente generará las soluciones que queremos, ya que cuanto mayor sea el cociente entre ganancia y gasto mejor es una reserva, pero podemos obtener el mismo cociente con diferentes valores y la ganancia neta obtenida no ser la misma, por lo que no nos distingue bien entre soluciones diferentes. Además este heurístico debería incluir una penalización de las no soluciones.

2.2.5 Problema 12

Una empresa de distribución de electricidad necesita calcular cada día qué centrales de producción ha de tener en marcha para abastecer la demanda de sus clientes. La empresa dispone de C centrales de producción eléctrica ubicadas en diferentes puntos de la geografía, con una capacidad de producción CA_i cada una. El coste diario de tener en marcha cada central es CO_i.

El contrato que tiene la empresa con cada uno de sus CL clientes indica la cantidad de electricidad (E_i) que se les debe suministrar cada día. La distancia entre los clientes y las centrales (d_{ij}) supone una pérdida en la eficiencia del suministro, esta pérdida se calcula como un factor fijo P multiplicado por esa distancia.

El objetivo es determinar el conjunto de centrales que se han de tener en marcha y a qué clientes han de servir, de manera que se minimice el coste de tenerlas en marcha y la pérdida debida a la distancia, teniendo en cuenta que la cantidad de electricidad que reciben los clientes de una central no ha de sobrepasar su capacidad de producción y que se ha de servir toda la demanda de los clientes. Asumiremos que los valores asignados al problema siempre permiten hallar una solución.

Una posible solución a este problema se puede obtener mediante el uso de algoritmos de búsqueda local. En los siguientes apartados se proponen diferentes alternativas para algunos de los elementos necesarios para plantear la búsqueda (solución inicial, operadores, función heurística,...). Comenta muy brevemente la solución que se propone respecto a si es correcta y si es mejor/peor respecto a otras alternativas posibles. Justifica tus respuestas.

a) Se plantea solucionarlo mediante Hill-climbing utilizando como solución inicial asignar a cada cliente al azar una central. Como operadores se utiliza el mover un cliente de una central a otra e intercambiar dos clientes entre dos centrales. La función heurística es la siguiente:

$$h(n) = \sum_{i=1}^{C} (CO_i \times marcha(i)) - \sum_{i=1}^{C} \sum_{j=1}^{CL} asig(i,j) \times d_{ij}$$

donde $asig(i,j)$ es una función que retorna uno si el cliente i esta asignado a la central j y cero en otro caso y $marcha(i)$ retorna uno si la central tiene clientes asignados y cero si no.

> El asignar los clientes al azar no nos asegura que obtengamos una solución, ya que la suma de las demandas puede superar la capacidad de alguna central. No obstante, si los demás elementos nos aseguran poder converger a una solución podría no ser un problema. Además el coste de esta asignación es lineal, que es el menor coste que podemos tener en este problema.
>
> Los operadores de mover e intercambiar deberían comprobar que no se superara la capacidad para mantenernos en el espacio de soluciones. La ramificación del operador mover es $O(clientes \times centrales)$, para intercambiar $O(clientes^2)$.
>
> La función heurística es incorrecta ya que la contribución de la perdida con la distancia tiene el signo contrario y además no tiene en cuenta el coste P asignado.

b) Se plantea solucionarlo mediante Hill-climbing utilizando como solución inicial un algoritmo avaricioso que ordena ascendentemente las centrales según su coste y los clientes según su

demanda, y va asignando clientes a las centrales según ese orden. Cuando se sobrepasa la capacidad de una central se pasa a la siguiente. El operador utilizado es intercambiar dos clientes entre dos centrales siempre que no se supere la capacidad de suministro de alguna de ellas. La función heurística es:

$$h(n) = \left(\sum_{i=1}^{C} CO_i\right) \times \left(\sum_{i=1}^{C}\sum_{j=1}^{CL} P \times asig(i,j) \times d_{i,j}\right)$$

La generación de la solución nos asegura que la solución cumpla la restricción de capacidad de las centrales. El coste de generar la solución viene dado por la ordenación de los clientes que es en el mejor caso $O(clientes \times \log(clientes))$, la asignación de clientes a las centrales se puede hacer en $O(clientes)$. No tenemos ninguna garantía sobre la calidad de la solución, ya que el coste de las centrales es lo único que intentará optimizar el algoritmo avaricioso.

El operador intercambiar no nos permite generar todo el espacio de búsqueda ya que el número inicial de clientes asignado a una centrar no se podrá cambiar con este operador. Al menos comprueba la restricción que mantiene la búsqueda en el espacio de soluciones.

La función heurística incluye los dos elementos del problema. Al problema principal es que no se tiene en cuenta las centrales que no se usan, asi que el primer termino es constante. Aunque solo se tuvieran en cuenta las centrales usadas, el minimizar esta función puede llevar a puntos de equilibrio entre los dos criterios donde no se minimice uno de los dos, ya que un valor grande de un criterio puede ser compensado con una cantidad pequeña del otro. También el factor multiplicativo P es totalmente irrelevante en esta función, ya que es común a todas las distancias y esta multiplicando.

c) Se plantea utilizar algoritmos genéticos donde la representación de la solución es una tira de bits. Esta está compuesta por la concatenación de C grupos de CL bits, donde cada grupo representa la asignación o no de cada cliente a la central como un 1 o un 0. Las población inicial la obtenemos mediante el mismo mecanismo que en el apartado anterior. Se utilizan los operadores habituales de cruce y mutación. La función heurística es:

$$h(n) = \left(\sum_{i=1}^{C} CO_i \times marcha(i)\right) + \left(P \times \sum_{i=1}^{C}\sum_{j=1}^{CL} asig(i,j) \times d_{i,j}\right)$$

La representación de la solución no es la más eficiente, ya que podríamos representar el problema asignando un número a cada central y usar ese valor para representar en que central esta asignado cada usuario. Representado en binario, el coste espacial seria $O(clientes \times \log(centrales))$.

El mecanismo de hallar una solución del apartado anterior es totalmente determinista, por lo que solo da una solución, lo que nos impediría generar una población de soluciones diferentes, que es lo que nos hace falta para usar los algoritmos genéticos.

Los operadores de cruce y mutación habituales generaran no soluciones ya que podríamos tener clientes asignados a la vez a varias centrales o usuarios no asignados. El operador de mutación sera especialmente problemático, ya que independientemente del bit que cambiemos dejaremos un cliente sin asignación o duplicaremos su asignación.

La función heurística es correcta.

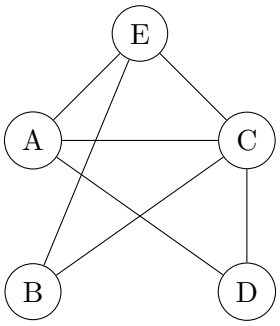
3. Satisfacció de restriccions

3.1 Problemas solucionados

3.1.1 Problema 19

Dado el siguiente grafo de restricciones donde cada restricción es una condición de desigualdad y los siguientes dominios para las variables:

A={1,2}
B={2,3}
C={1,2}
D={1,2,3}
E={1,2,3}



Haz la ejecución del forward checking hasta encontrar la primera solución

```
1) Asignación A= 1

Variable A restringe variable C
C={2 }
Variable A restringe variable D
```

```
D={2 3 }
Variable A restringe variable E
E={2 3 }

2) Asignación B= 2

Variable B restringe variable C

C={}

3) Asignación B= 3

Variable B restringe variable E
E={2 }

4) Asignación C= 2

Variable C restringe variable D
D={3 }
Variable C restringe variable E
E={}

Backtracking a A

5) Asignación A= 2

Variable A restringe variable C
C={1 }
Variable A restringe variable D
D={1 3 }
Variable A restringe variable E
E={1 3 }

6) Asignación B= 2

7) Asignación C= 1

Variable C restringe variable D
D={3 }
Variable C restringe variable E
E={3 }

8) Asignación D= 3

9) Asignación E= 3

 Solución 1
  A : 2
  B : 2
  C : 1
  D : 3
  E : 3
```

4. Anàlisi de mètodes de cerca

4.1 Como plantear los problemas cuestiones de búsqueda

En estos problemas se plantean diferentes alternativas que pretenden solucionar el problema planteado. Estas alternativas utilizan los tres mecanismos de resolución de problemas que se han visto en teoría: algoritmos de búsqueda heurística, algoritmos de búsqueda local y algoritmos de satisfacción de restricciones.

Para criticar cada alternativa se han de tener claras las características de cada uno de estos tres mecanismos:

Búsqueda heurística El problema se ha de plantear como la construcción de un camino, se definirá un estado inicial y un estado final que cumpla las características del problema.

Serán los operadores los que partiendo del estado inicial vayan construyendo el camino completo que llegue al estado final. Estos operadores han de comprobar las restricciones del problema y han de dar estados válidos. El factor de ramificación ha de ser el adecuado y no ha de haber mas o menos operadores de los necesarios.

La función heurística ha de dirigir la búsqueda hacia el tipo de solución que pide el problema. También ha de ser admisible y suficientemente informada como para garantizar una solución en un tiempo razonable. Si no hay posibilidad de encontrar un heurístico adecuado este tipo de resolución de problemas no sería viable debido a su coste a pesar de que permita encontrar una solución.

Búsqueda local El problema se ha de plantear como la mejora de una solución inicial. La solución incial por lo general deberá cumplir las restricciones a no ser que sea demasiado costoso. Si no se cumplen las condiciones de solución, se ha de garantizar que la búsqueda acabará en el espacio de soluciones.

Los operadores han de comprobar las restricciones del problema y generar soluciones correctas. Los operadores han de ser los necesarios y su factor de ramificación debe ser el justo para permitir la exploración del espacio de búsqueda.

La función heurística ha de incluir todos los elementos que se quieren optimizar y todos han de ir en la dirección adecuada. Si existen ponderaciones entre los diferentes elementos estas deberían estar justificadas.

Satisfacción de restricciones El problema se ha de plantear como la asignación de un conjunto de valores a unas variables. La elección de que son variables y sus dominios ha de ser correcta.

El conjunto de restricciones ha de ser completo y correcto.

El problema no ha de buscar la optimización de ningún parámetro ya que estos algoritmos no lo permiten.

4.2 Problemas solucionados

4.2.1 Problema 8

El ayuntamiento de una pequeña ciudad quiere ofrecer a sus ciudadanos un servicio similar al Bicing barcelonés, pero a menor escala. Uno de los problemas a resolver es como mover las bicicletas de un punto de recogida a otro para intentar que en toda estación haya algunos puestos libres para dejar bicicletas, y que en toda estación haya alguna bicicleta disponible. El ayuntamiento ya tiene un sistema que decide que bicicletas hay que mover de una estación a otra, y ha subcontratado los servicios de un camión, que puede llevar un máximo de B bicicletas a la vez. Cada hora el camión ha de recoger y dejar bicicletas en diferentes estaciones del servicio distribuidas por la ciudad, siguiendo un listado del ayuntamiento de pares (estación_origen, estación_destino), y ha de hacerlo realizando el recorrido más corto posible, sin que se sobrepase en ningún momento el máximo de bicicletas que el camión puede cargar. Cada hora partimos de cierto punto de origen y volvemos a él, habiendo movido todas las bicicletas que el ayuntamiento ha solicitado de su estación origen a su estación destino. Para obtener el recorrido se dispone de un mapa de la ciudad que indica la distancia mínima entre cada par de estaciones por las que ha de pasar el camión.

Puedes resolverlo mediante:

1. El algoritmo de A*. El estado es el camino recorrido. Utilizamos como coste la longitud del camino actual. La función heurística vale infinito si el camión en el estado actual supera el número B de bicicletas que puede transportar y, en caso contrario, es la suma de las distancias de las estaciones por recorrer al origen. El operador aplicable es pasar de la estación actual a otra no visitada. Para evitar la necesidad de otro operador, el punto de origen y final del camión se modela como una estación más.

> Se plantea la busqueda como un camino, lo que es adecuado para resolverlo mediante A*. La representación del estado es correcta, pero añadir el número de bicicletas que lleva el camión en cada momento facilitaria la comprobación de las rtestricciones.
>
> Utilizar la longitud del camino como coste de la solución es adecuado, ya que nos permite minimizarla.
>
> La función heurística no es admisible ya que la suma de las distancias de las estaciones por recorrer al punto de origen será más grande que la distancia del camino que queda por recorrer, por lo que no tenemos garantía de obtener el óptimo.
>
> El hacer infinito el heurístico cuando se supere el número B de bicicletas que puede transportar es una alternativa válida a hacer esa comprobación como condición de aplicabilidad del operador. Pero también se deberia hacer infinito el heurístico cuando el número de bicicletas transportadas sea menor que cero (se han dejado más bicicletas de las que tenia el camión).
>
> El operador de busqueda también es adecuado, dado lo que se propone para la función heurística. Con otra función heurística habría que comprobar las restricciones del problema
>
> En este caso para solucionar el problema correctamente se debería encontrar un heurístico admisible. En este problema concreto es difícil encontrar un heurístico admisible que sea una buena estimación del camino.

2. Satisfacción de restricciones, donde las variables son todas las aristas del grafo de conexiones entre las estaciones a recorrer, éstas son variables booleanas e indican si pertenecen al camino a

recorrer o no. Las restricciones son que debe haber exactamente dos aristas de un mismo vértice en la solución y que no se sobrepase el número B de bicicletas que el camión puede llevar a la vez en el recorrido formado por las aristas.

> La elección de variables y dominios para representar el problema es aparentemente correcta aunque un poco excesiva (un numero cuadrático de variables). Un problema de la representación es que nos representa que una arista entre dos estaciones forma parte del camino solución, pero no dice en que orden se visitan (y el orden puede ser importante cuando algunas de las bicicletas de una estación tienen como destino la estación siguiente, y no al revés).
>
> Cumplir las restricciones nos impondría que las aristas que elijamos formen un camino y que en ningún punto del recorrido se supere el número B de bicicletas. No obstante es imposible cumplir las restricciones explorando las asignaciones mediante ningún algoritmo de satisfacción de restricciones que hemos visto, no podríamos encontrar una asignación valida para la primera variable, es imposible tener dos aristas para un mismo vértice cuando estamos asignando la primera arista.
>
> Además, como el objetivo del problema es minimizar el camino, plantearlo como un problema de satisfacción de restricciones es incorrecto ya que no encontraremos el camino de longitud mínima sino uno cualquiera. Esta circunstancia no se puede corregir, por lo que no podríamos resolver el problema (encontrar el camino mínimo) de esta manera.
>
> Sin embargo, este problema se puede arreglar añadiendo una variable de camino recorrido (C), de dominio numérico entre 0 y la suma de todos los caminos, junto a una restricción n-aria codificando que C es igual a la suma de toda arista con su variable a '1' (es decir, parte del camino). Esta variable tiene un dominio grande, pero con una búsqueda con el algoritmo general de restricciones es factible encontrar la solución óptima en un tiempo razonable.

Comenta cada una de las posibilidades indicando si resuelven o no el problema, qué errores te parece que tiene cada solución y cómo se podrían corregir, y qué ventajas e inconvenientes tienen cada una de ellas. Justifica la respuesta.

4.2.2 Problema 14

Tras las numerosas críticas y movilizaciones de los taxistas por el mecanismo de turnos que ha implantado recientemente el *Ajuntament de Barcelona* para el sector, un grupo de trabajo formado por representantes de los taxistas y del *Ajuntament* nos ha pedido crear un sistema que adapte el número de taxis ciculando por el área metropolitana a la demanda estimada y al nivel de contaminación diaria (medida en gr. de CO_2 emitidos a la atmósfera.

Según nos cuentan, en estos momentos hay un número T de taxis registrados en el área metropolitana. Nos han contado que en vez de controlar el número de pasajeros que lleva cada taxi, el sistema controlará el número de servicios (un servicio es un viaje del taxi que recoge a un cierto número de pasajeros en un lugar de origen y los deja en un lugar de destino). Para simplificar el problema consideraremos que cada servicio dura unos 15 minutos de media. Se establece que el sistema asignará a cada taxi como mínimo NS servicios a la semana. Para cada taxi t disponemos del valor CO_t que mide el grado de contaminación que produce (medido en gr. de CO_2 emitidos por minuto). El *Ajuntament* nos dice que en ningun momento los taxis en servicio han de superar la cota máxima MAX_CO_H de gramos de CO_2 generados en una hora, y que en un día la cantidad de gramos de CO_2 emitidos no puede superar la cota MAX_CO_D. Con el número T de taxis disponibles es fácil superar ambas cotas. zona, tenemos una tabla semanal en la que, para cada zona z, dia d y hora h se tiene una estimación del número de servicios ($S_{z,d,h}$) que se suelen originar en esa zona, para cada hora del día. Para modelar la demanda esperada de taxis tenemos una tabla semanal en la que, para cada día d y hora h se tiene una estimación del número de servicios ($S_{d,h}$) que se suelen realizar.

El objetivo es decidir de forma dinámica los taxis que han de circular por el área metropolitana, de forma que se minimicen los niveles de contaminación diaria (nunca se pueden superar los límites) y se

maximice el número de servicios realizados (pero podemos dejar usuarios sin servir, si añadir servicios implica superar los límites de contaminación). Se nos plantean las siguientes alternativas:

1. Queremos utilizar A*. Definimos el estado como la asignación de taxis a horas del día. El estado inicial consiste en asignar de forma ordenada a cada taxi un número mínimo de servicios NS distribuidos de forma aleatoria en las horas de la semana. Tenemos un operador que asigna un nuevo servicio a un taxi t en un dia d y hora h determinados, el coste serán los gramos CO_t emitidos en los 15 minutos del servicio. Como función heurística usamos la suma del numero de servicios estimados en la tabla de demanda y que aun no hemos servido, o infinito si no se supera la cota $S_{d,h}$.

   Se plantea correcamente el problema como un espacio de estados donde el estado es la asignación de servicios a taxistas en un día y hora determinados. Sin embargo la elección del estado inicial es totalmente erronea: A* siempre ha de iniciarse con un estado vacio e irlo rellenando hasta llegar a una solución. Nunca podemos empezar en un estado que ya es solución o una solución parcial, porque entonces el algoritmo no puede asegurar que encuentre el óptimo.
   Lo diferente respecto a otros problemas planteados con A* es que en este caso el estado final no es aquel en que se cubre toda la demanda, ya que se da prioridad a no superar los niveles de contaminación por hora y por día, por encima de servir toda la demanda. Como veremos más adelante esto tiene un impacto importante en el heurístico.
   El operador es adecuado, asigna un servicio a un taxi en un día y hora determinados partiendo de un conjunto de taxis por asignar y un número estimado de servicios por realizar. El factor de ramificación será bastante grande, ya que en cada estado tenemos tantos posibles sucesores como posibles asignaciones de servicios se puedan hacer a taxis, pero en este caso es dificil reducir este coste ya que es inherente a la naturaleza del problema. No necesitamos operadores adicionales para mover asignaciones de un taxi a otro o para eliminar asignaciones, ya que eso lo hace el propio algoritmo de A* explorando otras ramas del espacio de búsqueda en paralelo. Sin embargo, para que el operador propuesto sea correcto, se deberia comprobar que no se superan los límites de contaminación en la hora y en el día. La otra restricción que no se controla en la propuesta, el número mínimo de servicios NS que se le han de asignar a cada taxi, no la podemos controlar en la condición de aplicabilidad del operador, ya que se ha de mirar cuando la solución esta (casi) construida.
   La función de coste en principio parece correcta, ya que tiene en cuenta el incremento de contaminación que aporta la asignación del servicio al taxi. Sin embargo un problema es que la funcion de coste g y el heurístico h no estan alineados, cada uno mide cosas diferentes en unidades dferentes, y esto como mínimo tendrá impacto en la consistencia de la heurística.
   El problema principal de la solución propuesta es la función heurística. El primer problema es que solo tiene en cuenta uno de los parámetros a maximizar (el número de servicios que quedan por servir) pero no tiene en cuenta que se quiere minimizar la contaminación y que, en caso de llegar al límite de contaminación, puede que dejemos demanda sin servir. Esto hace que el heurístico propuesto sea no admisible, ya que en caso de llegar al límite de contaminación sobreeestima el camino que queda por recorrer (el heurístico dice que aun queda demanda por servir pero en realidad no nos quedan asignaciones posibles a realizar). Un problema adicional del heurístico propuesto es el uso de la cota $S_{d,h}$ que, si se supera, hace que el heuristico tome el valor infinito. Esto no tiene sentido, ya que $S_{d,h}$ no es una cota máxima que no podamos superar, al contrario, es una cota mínima de servicios que querriamos servir y que incluso en ciertos casos no podremos servir.
   El heuristico correcto para este problema es complicado, ya que ha de combinar la demanda por servir, la contaminación acumulada y los servicios que le faltan a algunos taxis para llegar a la cota mínima semanal NS. La solución para combinar todos estos factores seria trabajar los tres aspectos en unidades de contaminación (los servicios que nos faltan por colocar los traduciremos en gr. de CO_2 que aun hemos de emitir usando el valor de de contaminación por minuto del taxi más ecológico.) La formula resultante seria el mínimo entre los gramos de CO_2 que nos quedan por emitir y los gramos de CO_2 que podemos emitir ante de llegar a los límites por hora y día, o infinito en el caso de que ya no podemos asignar más servicios a

> taxis sin superar los límites de contaminación pero existe algun taxi con un numero de servicios asignados menor que NS.

2. Queremos utilizar Hill Climbing, donde se genera una solución inicial asignando al azar suficientes taxis a servicios hasta llegar a servir toda la demanda estimada en cada hora. Los operadores de modificación de la solución consisten en intercambiar dos taxis entre servicios que pertenezcan a horas diferentes, y eliminar un servicio asignado a un taxi (siempre que no quedemos por debajo del límite NS de servicios mínimos asignados a ese taxi). Como función de evaluación usaremos el sumatorio de los gramos de CO_2 emitidos por todos los taxis asignados a servicios en la solución actual.

> La solución inicial que se plantea es una solución de máximos en la que se asignan servicios para cubrir toda la demanda. Es una solución poco costosa de calcular, pero como no comprueba ni las restricciones de contaminación ni el número mínimo de servicios por taxi, es muy probable que nos de no-soluciones, lo cual no seria grave si los operadores y la funcion de evaluación nos movieran lo antes posibles al espacio de soluciones, pero este no es el caso. Seria mejor que, en vez de hacer toda la asignación al azar, se empezara por asignar de forma secuencia tantos servicios a cada taxi como haga falta para cubrir la cota mínima NS y luego continuar con asignaciones pseudo-aleatorias hasta llegar a cubrir toda la demanda.
> Los dos operadores planteados son correctos ya que intentan reducir el nivel de contaminación de la solución o bien reasignando los taxis entre sí o bien eliminando servicios asignados, pero no comprueban ninguna de las restricciones, y podrian llevarnos a estados que no son solución. Además, con la solución inicial propuesta en el enunciado, los operadores no son capaces de explorar todo el espacio, ya que si por el azar algun taxi no ha recibido ninguna asignación en la solución inicial los operadores son incapaces de arreglarlo. Una opción a esto seria incluir un operador para añadir asignaciones de servicios a taxis.
> La función de evaluación que se propone es incompleta, ya que deberia de incluir el número de servicios realizados como factor a maximizar, e incluir unos factores de ponderación para controlar el peso de cada uno de los criterios en la función.

Comenta cada una de las posibilidades indicando si resuelven o no el problema y qué ventajas e inconvenientes tiene cada una de ellas. Justifica la respuesta.

4.2.3 Problema 16

Tenemos un sistema P2P que utiliza un mecanismo centralizado para asignar a cada cliente qué otros clientes son los que le envían las partes del fichero que le faltan. Cada cliente calcula una lista con los retardos medios de transmisión a cada uno de los clientes que conoce (en milisegundos). El mecanismo centralizado conoce el ancho de banda disponible de cada cliente tanto de subida como de bajada (en Kb/s) para el fichero que se quiere transmitir. Cada cierto tiempo el mecanismo centralizado distribuye a los clientes con qué otros clientes debe conectarse para recibir partes del fichero y qué ancho de banda dedicar. Para cada cliente conocemos qué partes del fichero tiene, por lo que podemos saber si puede enviar o no a un cliente. La idea es que minimicemos el tiempo de retardo total de las transmisiones y utilicemos el máximo ancho de banda de bajada disponible de cada cliente.

Comenta cada una de las posibilidades indicando si resuelven o no el problema, qué errores te parece que tiene cada solución y cómo se podrían corregir, y qué ventajas e inconvenientes tienen cada una de ellas. Justifica la respuesta.

1. Queremos utilizar A* de manera que recorremos la lista de clientes en un orden preestablecido. El estado es la asignación que hemos hecho de clientes y sus anchos de banda a los clientes recorridos. Utilizamos como operador asignar a un cliente uno de los que conoce (siempre que tenga partes del fichero que el cliente actual no tenga) y su máximo ancho de banda de subida al cliente actual, cuando el ancho de banda de bajada del cliente actual es superado por la suma de los anchos de banda de subida de los clientes asignados pasamos al siguiente cliente. Evidentemente una vez asignado un cliente para transmitir partes del fichero no lo podemos asignar más veces. El coste

del operador es el retardo del cliente asignado. La función heurística es la suma para los clientes que quedan por recorrer de los retardos a los clientes que conocen.

> Planteamos la busqueda como un camino en el que cada paso es la asignacion a un cliente a otro, correcto.
>
> Tenemos dos operadores, uno es asignar cliente al cliente actual y otro pasar al siguiente cliente, las restricciones de los operadores hacen que el segundo operador solo se pueda aplicar cuando no se puede aplicar el primero.
>
> En este caso el orden en que se hacen las asignaciones lo determina el coste de los operadores, que en este caso es el retardo, esto no nos da ninguna garantia de que podamos maximizar la ocupacion del ancho de banda.
>
> La funcion heuristica nos dara valores superiores de los retardos reales que sumaran las asignaciones que hacemos, por lo que no sera admisible.
>
> El mayor problema de esta solucion es no tener en cuenta como se asignan los anchos de banda en ningun momento.
>
> Otro problema es no poder compartir un cliente entre varios, esto impide que se pudieran encontrar asignaciones mejores.

2. Queremos utilizar búsqueda local generando una solución inicial en la que cada cliente recibe de todos los clientes que conoce que tienen partes del fichero que le faltan con un ancho de banda de 1 Kb/s. Como operadores tenemos aumentar o disminuir el ancho de banda de un cliente que transmite a otro en 1 Kb/s. La función heurística es la suma para cada cliente de los retardos de los clientes que le transmiten con un ancho de banda superior a 0 Kb/s.

> La solucion inicial que se plantea puede no ser solucion si para algun cliente el ancho de banda que soporta es inferior al numero de clientes que conoce. Tampoco tenemos en cuenta que lo que envian los clientes superen su capacidad de transmision. Por lo tanto deberiamos crear una solucion inicial que al menos respete estas restricciones.
>
> Los operadores nos deberian permitir explorar todas las posibles asignaciones de ancho de banda entre clientes siempre que tengamos en cuenta las restricciones de subida y bajada.
>
> La funcion heuristica solo se preocupa del retardo, por lo que explorando de esta manera la mejor solucion es en la que nadie transmite nada, que es la que menos retardo tiene. Para obtener una buena solucion deberiamos incluir restricciones que permitan maximizar el acho de banda que se recibe.
>
> Para ello deberian penalizarse soluciones en las que el ancho de banda de un cliente no este ocupado. Una manera sencilla podria ser calcular todo el ancho de banda que necesita el colectivo de clientes y tomarlo como valor base que se podria restar del ancho de banda realmente ocupado

4.2.4 Problema 17

Se quiere planificar cómo componer S servicios Web en un único servicio de orden superior (meta-servicio). Cada servicio Web usa un conjunto de agentes informáticos que deben ejecutarse en un orden específico para cumplir la tarea que realiza el servicio, estos agentes pueden trabajar en paralelo. Se supone que la acción que realiza cada agente tiene la misma duración (un paso) y hay que tener en cuenta que un servicio puede necesitar un mismo agente en diferentes pasos de su ejecución. Se dispone de un agente de cada tipo, teniendo un total de A agentes. El meta-servicio se considera completo cuando se haya completado cada servicio que lo compone. Se plantean dos siguientes alternativas para minimizar el número total de pasos de ejecución del meta-servicio.

Comenta cada una de las posibilidades indicando si resuelven o no el problema, qué errores te parece que tiene cada solución y cómo se podrían corregir, y qué ventajas e inconvenientes tienen cada

una de ellas. Justifica la respuesta.

1. Queremos utilizar A*. Definiremos el estado como la asignación de pasos de los S servicios individuales a uno de los A agentes en cada paso del meta-servicio. El estado inicial es tener un único paso del meta-servicio donde ninguno de los agentes tiene un servicio asignado.

   Los operadores de cambio de estado consisten en:

   a) Asignar el primer paso no ejecutado de alguno de los servicios a un agente libre en el paso actual del meta-servicio, con coste uno

   b) Añadir un paso nuevo al meta-servicio, con coste uno

   La función heurística es la suma de pasos de los servicios individuales que nos quedan por ejecutar dividida por el número de agentes.

   > Planteamos el problema como la construccion de un camino, que sera la asignacion de pasos de cada servicio del meta servicio a agentes, es el modo correcto de plantear una busqueda con A*.
   >
   > Iniciar desde el estado vacio tambien es la aproximacion correcta, si planetamos la busqueda como un camino este es el estado inicial, el estado final es cuando hemos hecho la asignacion completa.
   >
   > Los operadores van asignado los pasos actuales de los servicios a agentes en el paso actual del metaservicio o crean un nuevo paso del metaservicio, esto conecta los posibles estados adyacentes a uno dado. Seria mas productivo si al añadir un nuevo paso tambien asignaramos un agente a uno de sus pasos, asi evitariamos tener pasos del metasercicio vacios.
   >
   > El coste de estos operadores hace que A* minimice la suma total de pasos mas la suma pasos del metaservicio.
   >
   > Es un problema que el coste de añadir un nuevo paso del metaservicio y el coste de asignar un agente al paso actual sea el mismo ya que el A* explorara primero el añadir nuevos pasos, que no hacen nada, hasta que el coste supere la posibilidad de asignar un agente. Se supone que debemos meter el mayor numero de agentes en cada paso del metaservicio y con estos costes no favorecemos esa exploracion.
   >
   > La exploración de A* acabara minimizándonos la suma total de pasos (que es constante) y los pasos del metaservicio (que es lo que realmente nos interesa).
   >
   > El hacer que el coste del primer operador sea cero tampoco es una solucion ya que nos obligaria a explorar todas las posibles asignaciones de agente al paso actual antes de poder añadir un paso mas. Una posibilidad mejor es que el coste de añadir un nuevo paso fuera igual al numero de servicios.
   >
   > La funcion heuristica dados los costes es admisible, siempre estara por debajo del coste que tienen los caminos, de hecho es una cota del numero ideal de pasos.

2. Queremos utilizar satisfacción de restricciones. Suponemos que el número máximo de pasos del meta servicio (MP) es el número de veces que aparece el agente más utilizado, de manera que usamos $S \cdot MP$ variables para representar qué agente ejecuta un paso de un servicio en la secuencia de pasos del metaservicio. El dominio de cada variable son los A agentes, mas un valor que indica que la variable no esta asignada. Las restricciones son las siguientes:

   a) Para las variables de servicio en un paso, estas no pueden tener el mismo agente.

   b) Para las variables de un mismo servicio, estas no pueden violar la secuencia de acciones del servicio

> Es factible utilizar un solver de satisfacción de restricciones, pero solo si añadimos el objetivo a optimizar. En la propuesta actual, este algoritmo no optimiza, con lo cual no sería efectivo. Aunque no pretendiéramos optimizar el valor, tomar como numero máximo de pasos el numero de veces que aparece el agente mas utilizado no es correcto, el numero de pasos puede ser superior, por lo que no tendríamos variables suficientes para resolver el problema. En el caso de tener variables suficiente, las restricciones que dan para el grafo de restricciones serian correctas. Para poder pasar resolver este problema encontrando el valor óptimo, deberíamos tener una variable de tiempo usado T, con valor entre 0 y TMAX, a minimizar. Para simplificar cómo codificar las restricciones, podríamos tener un conjunto de ocupado[TMAX] variables, cada una asignada a un paso del proceso global y que son 1 en caso de que haya >0 agentes en uso en ese paso. Esto se puede codificar con TMAX*A restricciones binarias. Por último, tendríamos un conjunto de restricciones del tipo si ocupado[t] -> T>t, y con esto conseguiríamos optimizar el problema, e incluso llegar al óptimo.

4.2.5 Problema 20

Una empresa pesquera y conservera envía sus N barcos $\{b_i\}$ con capacidades para $C(b_i)$ pescadores a sus D destinos internacionales $\{d_i\}$ $(N > D)$ para 180 dias de pesca. Algunos de sus M pescadores $\{p_i\}$ $(M > \sum C(b_i))$ son asignados a los barcos y son enviados a los destinos en el barco asignado. El coste por marinero del viaje en el barco b_i a un destino d_j es $K_b(d_j, b_i)$. Cada pescador p_i proporciona a la empresa una ganancia de $g(p_i)$ euros/día en media y cuesta a la empresa $k(p_i)$ euros/día en media. La empresa requiere una planificación que cubra todos sus destinos sin pérdidas.

En los siguientes apartados se proponen diferentes alternativas para algunos de los elementos necesarios para plantear la búsqueda (solución inicial, operadores, función heurística). Comenta la solución que se propone respecto a si es correcta, es eficiente, y es mejor o peor en comparación con otras alternativas. Justifica todas las respuestas.

1. Aplicar búsqueda local. La solución inicial consiste en: 1) asignar secuencialmente $NdivD$ barcos a cada destino, repartiendo los $NmodD$ restantes equitativamente entre los primeros $NmodD$ destinos; 2) asignar aleatóriamente $C(b_i)$ pescadores a cada barco b_i. Los operadores son: cambiar el destino de un barco (y el de todos sus pescadores) e intercambiar dos pescadores entre dos barcos de destinos diferentes. Como función heurística usamos:

$$h'(n) = \sum_{i=1}^{D} \sum_{\forall b_j \in Asig(d_i))} K_b(d_i, b_j) + 180 \times \sum_{\forall p_i \in O(b_j)} (k(p_i) - g(p_i)))$$

donde $O(b_j)$ es la asignación de marineros del barco b_j $(|O(b_j)| \leq C(b_j))$ y $Asig(d_i)$ son los barcos asignados al destino d_i.

> El planteamiento del problema como una búsqueda en el espacio de soluciones es correcto. Aunque no nos piden maximizar o minimizar ningún criterio, dada la importante combinatoria del problema (barcos, pescadores, destinos) se puede considerar que en este caso la función heurística (si fuera correcta) reduce la exploración para conseguir lo antes posible una solución sin pérdidas (cualquier solución que nos dé el algoritmo que de ganancias nos vale).
>
> La solución inicial tiene un coste bajo ($O(P)$, siendo $P = \sum C(b_i)$) y garantiza que se cubren todos los destinos, pero impone que asignemos todos los barcos (no lo pide el enunciado) y que todos los barcos tengan tantos pescadores como sea necesario para llegar a su capacidad máxima (algo que tampoco se pide). La aleatoriedad en la asignación de pescadores hace que no podamos asegurar nada sobre la calidad de la solución inicial. Podría tener sentido ordenar primero los pescadores descendentemente por el valor $g(p_i) - k(p_i)$ y asignarlos en ese orden, así meteriamos en la solución los pescadores que generan mayor ganancia primero (y el incremento del coste sería $O(P \times \log(P))$). Dado que no se controla si la solución inicial genera pérdidas

tampoco podemos asegurar que sea una solución válida, y por ello tendremos que trabajar en el espacio de soluciones parciales, algo que no sería un problema si los operadores junto con el heurístico nos movieran lo antes posible hacia soluciones válidas (pero no es el caso).

Los operadores propuestos son insuficientes y no cubren todo el espacio de búsqueda (ni podemos alterar el número de barcos en la solución ni los pescadores asignados en la solución inicial). El operador de cambio de destino del barco puede convertir una solución en no solución ya que puede dejar alguno de los destinos sin barco (el operador no lo comprueba y el heurístico no lo penaliza), y su factor de ramificación sería $N \times D - 1$ (asumendo una representación optima donde el destino del barco no está replicado en la representación de los pescadores que van en él). El operador de intercambio de pescadores entre barcos de destinos diferentes tiene un factor de ramificación elevado (P^2) y es directamente inutil, ya que tanto el coste de un pescador como la ganancia que genera son independientes del destino que se les asigna y no se altera el número de pescadores asignados a cada destino (el coste $K_b(d_j, b_i)$). Seria mejor intercambiar pescadores en la solución por otros que no estén la solución, o mejor aún substituirlo por operadores para añadir y eleminar marineros de la solución. Si además se añaden operadores para añadir y eliminar barcos de la solución entonces si que podríamos explorar todo el espacio de búsqueda.

El heurístico es inadecuado porque aunque penaliza no-soluciones en las que haya pérdidas no penaliza de ninguna forma no-soluciones en las que haya destinos sin cubrir (de hecho pueden tener mejores valores que soluciones con todos los destinos cubiertos, se deberia hacer que valga infinito es esos casos para evitarlo). Aunque en general está bien planteado al usar las mismas unidades (costes) la primera parte de la formula solo considera el coste de un pescador a un destino (en vez de considerar el de los $\sum C(b_i)$ pescadores asignados).

2. Aplicar satisfacción de restricciones. Las variables son todos los posibles pares $< p_i, d_j >$ (pescador, destino). Todas las variables tienen como dominio el conjunto $\{b_i, undef\}$. El primer valor denota que el barco b_i se asigna al destino y que el pescador llegará en él. El segundo valor denota que el pescador no viaja al destino. Las restricciones son:

$$R1 : \forall_i |O(b_i)| \leq C(b_i) \qquad R2 : \sum_{j=1}^{N} \sum_{p_i \in O(b_j)} (k(p_i) - g(p_i)) \geq 0$$

El planteamiento del problema como un problema de satisfacción de restricciones es muy correcto ya que no se nos pide maximizar o minimizar ningú criterio, solo se nos pide una solución que cumpla unas propiedades.

La representación como una matriz de $M \times D$ variables $< p_i, d_j >$ (pescador, destino) es correcta ya que si reducimos los dominios de todas las variables a un solo valor (b_i o bien $undef$) y se cumplieran todas las restricciones del problema tendríamos una solución válida (el problema es que faltan restricciones por representar, como veremos a continuación). Lo que si que es criticable es que la representación es algo ineficiente en espacio, ya que toda solución válida tendrá la mayoría de las variables con valor $undef$.

Sin embargo la representación de variables escogida permite tener un pescador asignado a varios barcos, y un barco asignado a varios destinos, por loq ue se deberían añadir restricciones que lo impidan. La restricción $R1$ es correcta y solo se puede criticar que es algo costosa de comprobar. La restricción $R2$ sufre varios problemas: no tiene en cuenta el coste por marinero de los barcos a destino $K_b(d_j, b_i)$, y comprueba que las pérdidas por pescador sean superiores a las ganancias, cuando lo que tiene sentido es lo contrario (por ello habría que girar la resta). Un problema adiciónal de $R2$ es que solo se puede comprobar al final del proceso de satisfacción de restricciones, por lo que guia poco el algoritmo y potencialmente puede producir muchos backtrackings.

II Sistemes Basats en el Coneixement

5. Enginyeria del Coneixement

5.1 Problemas solucionados

5.1.1 Problema 8

Debido al creciente número de plataformas de streaming de contenido audiovisual, escoger dónde gastar el dinero para acceder a contenido de nuestro agrado es cada vez más complicado. Para ayudar en este propósito, queremos crear una herramienta llamada QuéVeo que permitirá a los usuarios obtener una recomendación sobre las plataformas que deberían contratar. Para el desarrollo de nuestra herramienta queremos utilizar un sistema basado en el conocimiento que, en base al poder adquisitivo del usuario, sus preferencias personales y con quién suele visionar productos audiovisuales, nos permita recomendarle varios packs de contenidos seleccionados de diferentes plataformas.

En QuéVeo tenemos un registro con todas las plataformas de contenidos que existen. Cada plataforma tiene asociados uno o varios packs de contenido (para reflejar el hecho de que algunos contenidos pueden ser ofrecidos como compra o alquiler). Un pack tiene un coste mensual y un catálogo, que puede ser distinto para cada país. Cada catálogo incluye un conjunto de obras. Cada obra tiene un formato determinado: película, serie o documental. De cada obra queremos guardar su título, el nombre y apellidos de sus directores, sus géneros narrativos, su edad mínima recomendada y sus nacionalidades (lista de países de las entidades productoras). Los géneros que puede tener una película o una serie son acción, ciencia ficción, comedia, drama, fantasía, melodrama, musical, romántica, thriller y/o terror. Los géneros que puede tener un documental son sociopolítico, artístico, divulgativo, de naturaleza y/o falso documental. De cada película, además, queremos guardar su duración. Las series se componen de temporadas, y cada temporada se compone de episodios. De cada episodio también queremos almacenar su duración.

Para QuéVeo también son relevantes las bases de datos que existen en Internet con información de las obras audiovisuales que se producen en todo el mundo. Cada base de datos contiene registros para un conjunto de obras, y para cada una de ellas se almacena la puntuación de los usuarios, la puntuación de la crítica y si la obra cumple con ciertas categorías especiales: si la obra es de culto, si es un clásico, si es un *blockbuster*, si es de autor y/o si es de serie B. Cada obra de los catálogos de las plataformas estará relacionada con la información de las obras almacenadas en los registros de las bases de datos.

Para utilizar QuéVeo necesitamos del usuario su nombre, su primer apellido, su país de residencia, su número de teléfono, su correo electrónico, su año de nacimiento (para poder estimar su edad) y cuánto dinero está dispuesto a gastarse en plataformas cada mes. Además queremos guardar sus géneros

favoritos, el formato de obra que prefiere ver y las categorías especiales en las que tiene especial interés.

Sabemos que cuando un usuario contrata una plataforma de streaming es muy probable que no sea el único consumidor del contenido. Por lo tanto del usuario se pide que nos diga si vive solo, en pareja, en familia o en un piso compartido. Si el usuario vive en familia también nos interesa guardar el número de niños menores de 10 años, y si vive en un piso compartido queremos guardar el número de convivientes y la diferencia de edades entre la edad más alta y la edad más baja. En cualquier caso queremos almacenar la media de horas totales de contenido audiovisual que se consume en su casa cada semana.

Debido a la complejidad del dominio de QuéVeo buscamos simplificar, identificando un conjunto de características que nos permitan categorizar a cada usuario del sistema:

- la **diversidad de los espectadores**, que puede ser *baja*, *media* o *alta*. La diversidad será *baja* si el número de convivientes en casa del usuario (incluyendo el propio usuario) es uno o dos, *media* si el número de convivientes es más de dos y el usuario vive en familia con como mucho un niño o en un piso compartido con una diferencia de edades menor de 10, y *alta* en cualquier otro caso.

- el **nivel de cinefilia**, que dependerá de cuántas preferencias ha configurado el usuario sumando sus géneros preferidos y las categorías especiales seleccionadas. El nivel puede ser *nulo* (sin preferencias), *poco-cinéfilo* (como máximo 5 preferencias), *algo-cinéfilo* (entre 5 y 10 preferencias) o *cinéfilo* (más de 10 preferencias).

- el **volumen de consumo**, que dependerá del número de horas de consumo audiovisual proporcionado por el usuario y que puede ser *escaso* (menos de 10 horas), *normal* (entre 10 y 40) o *alto* (más de 40).

- el **gasto potencial**, que puede ser *bajo*, *normal* o *elevado*. El gasto será *bajo* si el dinero que el usuario está dispuesto a gastarse por mes es menor de 10 euros, *medio* si está dispuesto a gastarse menos de 50 euros y *elevado* en el resto de casos.

Estas categorías nos permiten identificar el tipo de recomendaciones que podemos hacer al usuario. Para clasificar los diferentes tipos de recomendaciones definimos las siguientes características categóricas:

- la **variedad**: con esta característica queremos identificar si es una prioridad que haya variedad de contenidos. Si la diversidad de los espectadores es *alta* o el usuario es *cinéfilo* la prioridad de recomendar packs de contenido con mucha variedad será *prioridad-muy-alta*. Si la diversidad es *baja* y además el nivel de cinefilia es *nulo* o *poco-cinéfilo* asignaremos *sin-prioridad*. En el resto de usuarios nos guiaremos por el volumen de consumo: si es *alto*, asignaremos *prioridad-alta* y en cualquier otro caso *prioridad-media*.

- el **número de packs de contenido** a recomendar: recomendaremos sólo *un-pack* de contenido si el volumen de consumo es *escaso*, el gasto potencial es *bajo* o la diversidad es *baja*. Alternativamente, recomendaremos *dos-a-cuatro* packs si ni el volumen de consumo es *escaso*, ni el gasto potencial es *bajo* ni la diversidad es *baja*, excepto si el gasto potencial es *elevado* y el volumen de consumo es *alto* en cuyo caso el número de packs a recomendar será *cinco-o-más*.

- si hay que **considerar categorías especiales**: con esta característica identificaremos si las categorías especiales que el usuario especificó como preferencias serán relevantes a la hora de construir la recomendación. Su valor será *sí* si el nivel de cinefilia es *algo-cinéfilo* o *cinéfilo* y, o bien la diversidad de los espectadores es *baja*, o bien el volumen de consumo es *alto*, y será *no* para cualquier otro usuario.

El objetivo es construir un sistema capaz de generar una recomendación de packs de contenidos a cada usuario de QuéVeo.

1. (2,5 puntos) Diseña la ontología del dominio descrito, incluyendo todos los conceptos que aparecen en la descripción y todas las relaciones necesarias. Identifica qué conceptos forman parte de los datos de entrada del problema y qué conceptos forman parte de la solución. Justifica las decisiones de diseño que has realizado. En el caso de las subclases de una clase, justifica con un texto breve la necesidad de crear dichas subclases. Para cada concepto de la ontología lista todos sus atributos (para cada atributo se ha de indicar su nombre y su tipo). [Nota: tened en cuenta que la ontología puede necesitar modificaciones para adaptarla al apartado siguiente.]

Para resolver este problema, la ontología ha de incluir todos los conceptos que forman parte de la entrada del problema:

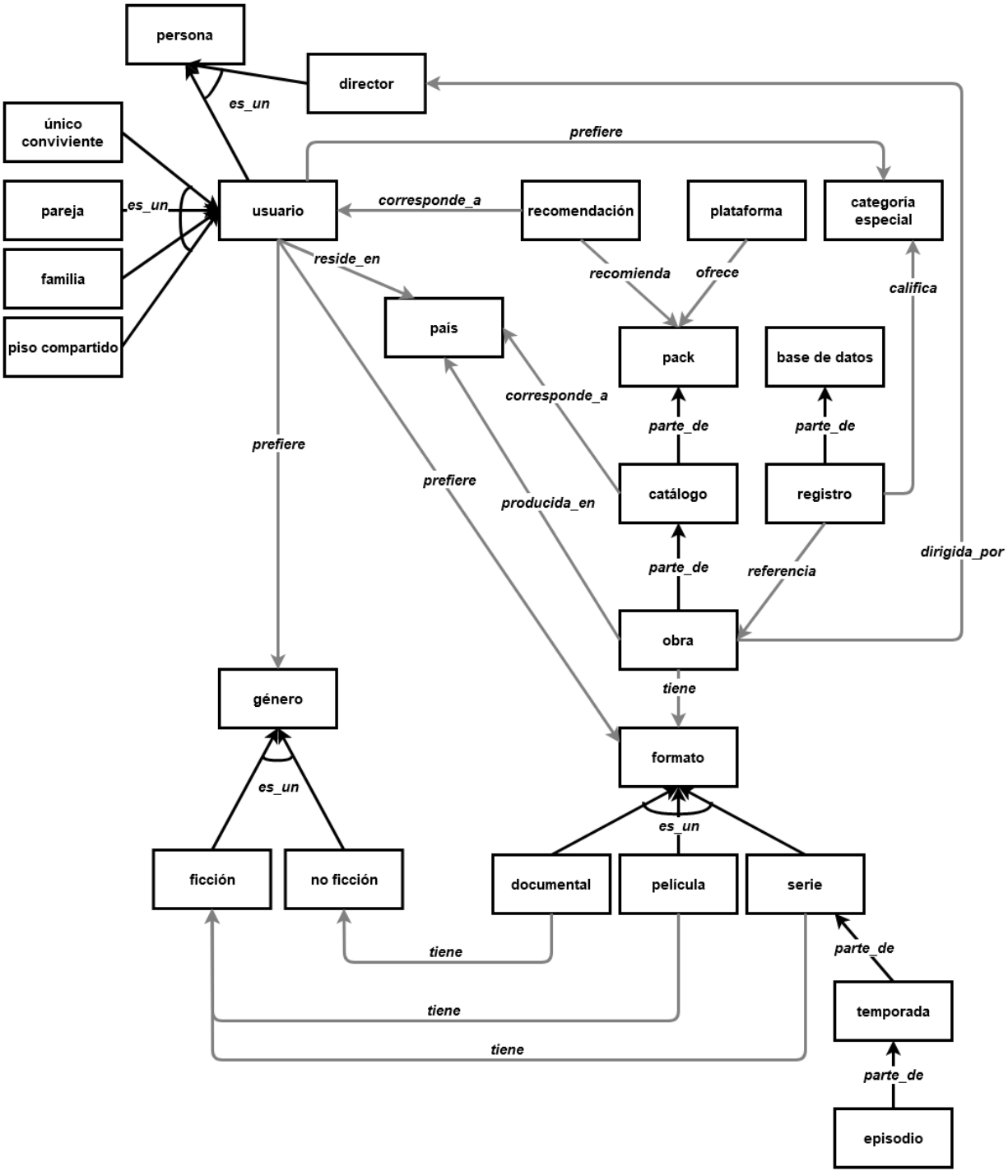
- La información sobre el usuario y con quién convive.
- La información relacionada con los contenidos ofrecidos por las plataformas.
- Los elementos que forman parte de las preferencias del usuario y que a su vez se utilizan para categorizar los contenidos: géneros, formatos, categorías especiales.

La ontología también tiene que incluir los conceptos que se utilizarán para proporcionar la salida del sistema:

- La recomendación basada en un conjunto de packs de contenido.

A la hora de decidir qué conceptos se deben materializar como clases o como atributos, y qué superclases hay que crear más allá de las explícitamente definidas en el enunciado, debemos fijarnos principalmente en aquellos conceptos que comparten propiedades. Una regla general que hemos seguido es la de crear subclases cuando una o varias tienen atributos y/o relaciones diferentes. En esta ontología, encontramos el caso de `Documental`, `Película` y `Serie` que han de ser clases debido a que, por un lado, `Serie` y `Película` tienen una relación con los géneros que es diferente a la que tiene `Documental`, y por el otro lado, `Serie` tiene una relación con `Temporada` que no tienen ni `Documental` ni `Película`. Deberá haber una superclase para las tres (`Formato`), ya que será la que tenga relación con el `Usuario` para especificar sus preferencias.
Algo parecido pasa en el caso de los géneros: el conjunto {acción, ciencia ficción, comedia, . . . , terror} y el conjunto {sociopolítico, artístico, . . . , falso documental} tienen relaciones diferentes. Sin embargo, dentro de cada conjunto los géneros no tienen relaciones diferentes. Una manera de modelar esto es creando nuevas superclases para cada conjunto: `Ficción` (con la cual estarán relacionadas `Serie` y `Película`) y `No_Ficción` (con `Documental`), respectivamente. Como ya hemos mencionado, los géneros que corresponden o bien a `Ficción` o bien a `No_Ficción` no tienen atributos o relaciones diferentes y por lo tanto no es necesario añadirlas como subclases y se pueden modelar añadiendo un atributo `tipo` a estas dos superclases nuevas. Debido a que del `Usuario` obtenemos un conjunto de géneros preferidos, deberemos añadir `Género` como superclase de `Ficción` y `No_Ficción`, con la cual estará relacionada `Usuario`.
En el enunciado se nos dice que del `Usuario` necesitamos guardar el nombre y el primer apellido, mientras que de los directores de las obras necesitamos almacenar el nombre y los apellidos. Debido a estos atributos comunes, decidimos crear una clase `Director` y una superclase de ambas, `Persona`, que tendrá como atributos el nombre y el primer apellido. Análogamente, hemos definido los conceptos `Único_Conviviente`, `Pareja`, `Familia` y `Piso_Compartido` como subclases de `Usuario`, debido a que dos de las suibclases tienen atributos diferentes, y las otras dos se han creado siguiendo el heurístico de mantener el mismo nivel de detalle en las diferentes ramas de la ontología.
Otra cosa importante a destacar es que siempre que el enunciado nos dice que un concepto `A` se compone de `B` hemos usado la relación taxonómica `B parte_de A`, que modela correctamente cómo un concepto se compone de sus partes: `Episodio parte_de Temporada`, `Temporada parte_de Serie`. Además hemos usado esta relación para `Registro parte_de Base_de_Datos`, `Catálogo parte_de Pack` y `Obra parte_de Catálogo`, debido a que, aunque se podrían modelar con relaciones no taxonómicas, se puede interpretar limitándose al enunciado que los segundos conceptos se componen de los primeros.

A continuación se listan los atributos para representar la información mencionada explícitamente en el enunciado y las modificaciones necesarias para el apartado b) del problema. Hemos decidido también representar las características abstractas del problema y las de la solución abstracta (son los que tienen el nombre en *Cursiva* o *CURSIVA* respectivamente). De esta manera todos los conceptos que aparecerán en las reglas están soportados por la ontología. El enunciado no propone características específicas de la solución concreta por lo que no es necesario añadirlas ni distinguirlas visualmente.



Atributos:

- `Plataforma`: nombre (string)
- `Pack`: coste mensual (entero positivo)
- `Obra`: título (string), nombre_completo_directores (string), edad_mínima_recomendada (entero positivo)

- `Pais`: nombre (string)
- `Ficción`: tipo (enumeración: {acción, ciencia_ficción, comedia, drama, fantasía, melodrama, musical, romántica, thriller, terror})
- `No_Ficción`: tipo (enumeración: {sociopolítico, artístico, divulgativo, de_naturaleza, falso_documental})
- `Película`: duración (entero positivo)
- `Episodio`: duración (entero positivo), num_episodio (entero positivo)
- `Temporada`: num_episodio (entero positivo)
- `Base_de_Datos`: nombre (string)
- `Registro`: puntuación_usuarios (entero positivo), puntuación_crítica (entero positivo)
- `Categoría_Especial`: tipo (enumeración: {de_culto, clásico, blockbuster, de_autor, serie_B})
- `Persona`: nombre (string), primer_apellido (string)
- `Usuario`: número_teléfono (string), correo_electrónico (string), año_nacimiento (entero positivo), gasto_mensual (real positivo), media_consumo_semanal (real positivo), *Diversidad* (enumeración: {baja, media, alta}), *Nivel_Cinefilia* (enumeración: {nulo, poco-cinéfilo, algo-cinéfilo, cinéfilo}), *Volumen_Consumo* (enumeración: {escaso, normal, alto}), *Gasto_Potencial* (enumeración: {bajo, normal, elevado})
- `Director`: segundo_apellido (string)
- `Familia`: número_niños (entero positivo)
- `Piso_Compartido`: número_convivientes (entero positivo), diferencia_edades (entero positivo)
- `Recomendación`: *VARIEDAD* (enumeración: {sin-prioridad, prioridad-media, prioridad-alta, prioridad-muy-alta}), *NÚMERO_PACKS* (enumeración: {un-pack, dos-a-cuatro, cinco-o-más}), *CONSIDERAR_CATEGORÍAS_ESPECIALES* (booleano)

2. (2,5 puntos) El problema descrito es un problema de análisis. Explica cómo lo resolverías usando clasificación heurística, justificando si el tipo de solución que se pide requiere de refinamiento o no. Lista las variables que forman parte del Problema Abstracto (nombre y valores posibles). Lista las variables que forman parte de la Solución Abstracta (nombre y valores posibles). Da al menos 4 ejemplos de reglas (suficientemente variadas, utilizando diferents conceptos) para cada una de las fases de esta metodología, usando los conceptos de la ontología desarrollada en el apartado anterior. Para las reglas podéis usar la notación de alto nivel que usamos en los ejercicios de problemas (si ... entonces ...) o una notación más próxima a la del lenguaje CLIPS (siempre que queden claro los elementos del antecedente de la regla y del consecuente).

Para resolverlo mediante clasificación heurística debemos identificar en el problema las diferentes fases y elementos de esta metodología. El sistema require que la solución sea una recomendación de un número de packs de contenido seleccionados a partir de un conjunto de preferencias y de información sobre quién compartirá la cuenta con el usuario. Por lo tanto, el sistema nos pide una solución concreta y hará falta refinamiento.

Una vez sabemos cuántas fases requiere nuestra solución ya podemos enunciar la solución completa. El primer elemento son los problemas concretos que hay que tratar. En este dominio los problemas concretos están definidos por la información obtenida del usuario así como por la información detallada de los catálogos de las plataformas y de las bases de datos de obras. Supondremos la existencia de funciones que nos permitan acceder a las relaciones, calcular la cardinalidad, sumar, comparar y comprobar la igualdad.

El segundo elemento son los problemas abstractos, que estarán definidos a partir de las cuatro características que menciona el enunciado (*Diversidad*, *Nivel_Cinefilia*, *Volumen_Consumo* y *Gasto_Potencial*), todas ellas pertenecientes a `Usuario` y con rangos de valores discretos:

- `Usuario`.*Diversidad* (enumeración: {baja, media, alta})
- `Usuario`.*Nivel_Cinefilia* (enumeración: {nulo, poco-cinéfilo, algo-cinéfilo, cinéfilo})
- `Usuario`.*Volumen_Consumo* (enumeración: {escaso, normal, alto})
- `Usuario`.*Gasto_Potencial* (enumeración: {bajo, normal, elevado})

Para conectar los problemas concretos con los abstractos necesitamos definir las reglas de abstracción de datos, por ejemplo:

- si cardinalidad(`Único_Conviviente`) > 0 entonces `Usuario`.*Diversidad* = baja;
- si `Piso_Compartido`.número_convivientes > 2 y `Piso_Compartido`.diferencia_edades >= 10 entonces `Usuario`.*Diversidad* = alta;
- si (cardinalidad(`Usuario`.prefiere.Género) + cardinalidad(`Usuario`.prefiere.Categoría_Especial)) <= 5
  y (cardinalidad(`Usuario`.prefiere.Género) + cardinalidad(`Usuario`.prefiere.Categoría_Especial)) > 0
  entonces `Usuario`.*Nivel_Cinefilia* = poco-cinéfilo;
- si (cardinalidad(`Usuario`.prefiere.Género) + cardinalidad(`Usuario`.prefiere.Categoría_Especial)) > 10
  entonces `Usuario`.*Nivel_Cinefilia* = cinéfilo;
- si `Usuario`.media_consumo_semanal < 10 entonces `Usuario`.*Volumen_Consumo* = escaso;
- si `Usuario`.media_consumo_semanal <= 40 y `Usuario`.media_consumo_semanal >= 10 entonces `Usuario`.*Volumen_Consumo* = normal;
- si `Usuario`.gasto_mensual < 10 entonces `Usuario`.*Gasto_Potencial* = bajo;
- si `Usuario`.gasto_mensual >= 50 entonces `Usuario`.*Gasto_Potencial* = elevado;

El tercer elemento son las soluciones abstractas. El enunciado define tres características que definirán las soluciones abstractas: *VARIEDAD*, *NÚMERO_PACKS* y *CONSIDERAR_CATEGORÍAS_ESPECIALES*.

- `Recomendación`.*VARIEDAD* (enumeración: {sin-prioridad, prioridad-media, prioridad-alta, prioridad-muy-alta}),
- `Recomendación`.*NÚMERO_PACKS* (enumeración: {un-pack, dos-a-cuatro, cinco-o-más}),
- `Recomendación`.*CONSIDERAR_CATEGORÍAS_ESPECIALES* (booleano: {si, no})

Para ligar los problemas abstractos con las soluciones abstractas necesitaremos reglas de asociación heurística, como por ejemplo:

- si `Usuario`.*Diversidad* == alta o `Usuario`.*Nivel_Cinefilia* == cinéfilo
  entonces `Recomendación`.*VARIEDAD* = prioridad-muy-alta;
- si `Usuario`.*Diversidad* == baja y `Usuario`.*Nivel_Cinefilia* == poco-cinéfilo
  entonces `Recomendación`.*VARIEDAD* = sin-prioridad;
- si `Usuario`.*Volumen_Consumo* == escaso o `Usuario`.*Gasto_Potencial* == bajo
  o `Usuario`.*Diversidad* == baja
  entonces `Recomendación`.*NÚMERO_PACKS* = un-pack;
- si `Usuario`.*Gasto_Potencial* == elevado y `Usuario`.Volumen_Consumo == alto
  entonces `Recomendación`.*NÚMERO_PACKS* = cinco-o-más;
- si `Usuario`.*Volumen_Consumo* == alto
  entonces `Recomendación`.*CONSIDERAR_CATEGORÍAS_ESPECIALES* = cierto;

El cuarto y último elemento son las soluciones concretas. En este caso corresponde a la creación de una lista de packs de contenido concretos a partir de la solución abstracta y de los datos concretos del problema. En este enunciado no hay detalles concretos sobre cómo guiar el proceso de refinamiento, pero podemos suponer la existencia de funciones que combinan la solución abstracta con datos concretos del problema para obtener los packs de contenido. Algunos ejemplos de reglas podrían ser:

- si `Recomendación`.*CONSIDERAR_CATEGORÍAS_ESPECIALES* == no
  y `Recomendación`.*VARIEDAD* == sin-prioridad
  entonces recomienda_packs_por_alta_puntuación_usuarios(`Recomendación`.*NÚMERO_PACKS*)
- si `Recomendación`.*VARIEDAD* == sin-prioridad
  y `Usuario`.prefiere.`Formato` == `Documental`
  y `Recomendación`.*NÚMERO_PACKS* == un-pack
  entonces recomienda_pack_plataforma_especializada_docus(`Usuario`.prefiere.`Género`)
- si `Recomendación`.*CONSIDERAR_CATEGORÍAS_ESPECIALES* == sí
  y `Recomendación`.*NÚMERO_PACKS* == un-pack y `Recomendación`.*VARIEDAD* == nula
  entonces recomienda_pack_especial(`Usuario`.prefiere.`Categoría_Especial`)
- si `Recomendación`.*CONSIDERAR_CATEGORÍAS_ESPECIALES* == no
  y `Recomendación`.*NÚMERO_PACKS* == un-pack
  y `Recomendación`.*VARIEDAD* == prioridad-muy-alta
  entonces recomienda_pack_variedad(`Usuario`.prefiere.`Género`)
- si `Recomendación`.*CONSIDERAR_CATEGORÍAS_ESPECIALES* == no
  y `Recomendación`.*NÚMERO_PACKS* == dos-a-cuatro
  y `Recomendación`.*VARIEDAD* == prioridad-muy-alta
  entonces recomienda_packs_genéricos(calcula_número_convivientes(`Usuario`),
  `Usuario`.prefiere.`Género`)
- si `Recomendación`.*CONSIDERAR_CATEGORÍAS_ESPECIALES* == sí
  y `Recomendación`.*NÚMERO_PACKS* == cinco-o-más
  y `Recomendación`.*VARIEDAD* == sin-prioridad
  entonces recomienda_muchos_packs_especiales(calcula_número_convivientes(`Usuario`),
  `Usuario`.prefiere.`Categoría_Especial`)

5.1.2 Problema 9

Una empresa del sector de los videojuegos quiere ir mucho más allá en la introducción de la Inteligencia Artificial en sus productos. Para su próximo videojuego quieren que la IA del juego adapte la historia, el ritmo y el tipo de juego al tipo de jugador que lo usa, de forma que tenga más o menos acción, más o menos aventura, más o menos misterio, centrar la historia en el jugador individual o modificarla para que sea un juego de estrategia por equipos, todo ello dentro del mismo juego. Para hacer esta adaptación el sistema dispone del historial de jugador, tanto dentro del juego como en otros juegos de la misma compañia, extrayendo de ahí el perfil de jugador.

De cada juego del catálogo de esta empresa tiene asociado un título, un año de publicación, la edad recomendada (+0 años, +3 años, +5 años, +8 años, +12 años, +18 años), si es un juego on-line o no, si es un juego multi-jugador y la saga de juegos a la que pertenece (de la que solo se guarda el título y los juegos que la componen). La empresa solo ofrece en estos momentos ocho tipos de juego: arcade, juegos de estrategia en tiempo real (más conocidos por las siglas RTS en inglés), juegos de lucha, de deportes, aventuras gráficas, puzzles, simuladores de mundo y simuladores de vehículos, y de todos ellos se guarda la misma información excepto en el caso de los simuladores de vehículos, en los que se guarda además el conjunto de vehículos que los jugadores pueden pilotar. Cada juego esta compuesto por uno o más niveles, y de cada nivel se guarda su nombre, la dificultad (de 1 a 10), el conjunto de niveles desde los que se puede acceder a ese nivel, y el tiempo máximo para poder superar el nivel (en segundos, si vale 0 significa qe no hay límite de tiempo). Cada nivel está compuesto por uno o más espacios, y de cada uno se guarda su nombre, si es un espacio interior o exterior, el conjunto de espacios desde los que se puede acceder a ese espacio y el conjunto de objetos que estan situados dentro de ese espacio. De cada objeto se guarda su nombre y su precio en créditos (en moneda ficticia, el usuario puede gastar esos créditos en el mismo juego o en otros juegos de la misma compañia). Hay cinco tipos de objetos: las llaves, de las que se guarda además el espacio al que pueden dar acceso; las armas, de las que se guarda nivel de daño que pueden generar por ataque y la energia disponible (en %); los vehículos, que pueden ser aereos, espaciales, marinos, anfibios o terrestres, y de los que se

guarda el nombre, el nivel de energía y el número de plazas; las herramientas, de las que se guarda solo el nombre y el precio, y los contenedores, de los que se guarda el conjunto de objetos que tienen dentro.

La participación de los jugadores en los juegos se guarda por partidas, y de cada partida se guarda la fecha y hora de inicio, la duración (en minutos), los niveles del juego completados por el usuario, los personajes que han participado y los equipos que han participado. En un juego hay uno o varios personajes, que pueden ser personajes jugador (personajes que corresponden a usuarios reales) y personajes no jugador (más conocidos por el acrónimo NPC, personajes que corresponden a jugadores virtuales que crea el juego). De los personajes jugador se guarda el nombre del personaje (texto), el usuario real asociado a este personaje su rol (texto), su salud (en %), los puntos obtenidos (entero), los creditos obtenidos (en la moneda fictícia) y el conjunto de objetos del juego que tiene (ya sea porque los encontró o porque los compró). De los personajes no jugador se guarda el nombre del personaje (texto), su rol (texto), su salud (en %), si es enemigo, y el nivel de inteligencia artificial (de 1 a 10). Tanto los personajes jugador como los no jugador pueden formar un equipo, del que se guarda su nombre y todos sus integrantes. De cada usuario se guarda el nombre de usuario, todas las partidas que ha jugado y el crédito total acumulado (en la moneda fictícia).

Los personajes pueden realizar acciones, y cada acción tiene asociado un nombre, la fecha y hora en la que se completó la acción, el tiempo de reacción del personaje a la hora de ejecutar la acción (en milisegundos), el conjunto de acciones de las que esta acción depende para poder ejecutarse, el espacio en el que se ha realizado la acción y el conjunto de objetos que requiere la acción para poder hacerla. Los personajes también pueden participar en un diálogo, y de los diálogos se guarda el identificador de diálogo y el conjunto de personajes que ha participado. Todos los juegos (incluso los puzzles de habilidad más simples) tienen una historia asociada, ligada a lo que el jugador va haciendo en el juego. La historia está compuesta por hitos argumentales, es decir puntos importantes dentro de la historia del juego a los que el jugador ha de llegar. Cada hito argumental ocurre en un nivel del juego (un nivel puede tener varios hitos asociados) y tiene asociada una acción objetivo que el jugador ha de conseguir realizar. La información que se guarda para cada hito incluye el tema (una misión individual, una misión grupal, una relación personal, un misterio), si el hito es central (es decir, imprescindible para el juego) o si es opcional (se puede añadir o eliminar sin que el juego pierda sentido), la acción a realizar, la fecha y hora límite para realizar la acción (todo a ceros si no hay límite temporal). Todos los hitos argumentales tienen asociadas tres escenas pre-grabadas de animación 3D: la escena de inicio (un cortometraje donde se narra parte de la historia y le dan pistas al jugador de la acción objetivo que ha de realizar) la escena de objetivo cumplido (se confirma al jugador que ha conseguido el objetivo del hito, y se cuenta un trozo de historia que es consecuencia del éxito) y la de objetivo no cumplido (un trozo de historia consecuencia del fracaso), de estas escenas solo se guarda el identificador del video y la duración de la escena. Además un hito argumental puede incluir uno o más dialogos (que se insertan en el juego para guiar al usuario en cierta dirección, o darle pistas).

Los servidores de la empresa recopilan toda esta información y la complementan con otros estadísticos generados de forma automática, como el tiempo medio de reacción del jugador (en milisegundos) en las acciones de la partida actual / en el día actual / en la última semana / en el último mes; o el tiempo extra (en segundos) que el usuario dedica a explorar los espacios en la partida actual / en el día actual / en la última semana / en el último mes, para las escenas animadas el porcentaje de video (en media) que ve el usuario en la partida actual / en el día actual / en la última semana / en el último mes (hay usuarios que veran normalmente los videos enteros, otros que se saltarán el video para ir a la acción, pero esto puede variar por dias, según si el usuario tiene tiempo para verlos), etc.

Por lo que nos dicen los expertos se suelen usar varias características para construir un perfil de jugador:

- La **salud** del personaje en la partida actual, que puede ser muy baja ($< 10\,\%$) , baja ($< 40\,\%$), normal ($< 80\,\%$) o elevada ($\geq 80\,\%$).
- El **tiempo de reacción** medio de las acciones en la partida actual, que puede ser rápido (< 0.4 seg), normal (< 0.9 seg) o lento (≥ 0.9 seg).
- El **tiempo de exploración** medio en las partidas del día actual, que puede ser mínimo (< 5 seg), normal (< 50 seg) o elevado (≥ 50 seg).

- El **tiempo de narración** medio que admite el usuario en las escenas animadas del día actual, que puede ser mínimo ($< 3\%$), bajo ($< 20\%$), normal ($< 80\%$) o elevado ($\geq 80\%$).
- El **nivel de trabajo en equipo**, que está asociado al porcentaje de partidas jugadas e el último mes en las que el usuario ha participado dentro de un equipo, y que puede ser muy bajo ($< 10\%$) , bajo ($< 40\%$), normal ($< 80\%$) o elevado ($\geq 80\%$).
- La **interacción con otros personajes**, que está asociada al número de diálogos en los que el usuario ha participado en el último mes, y que puede ser muy baja (< 5) , baja (< 20), normal (< 30) o elevada (≥ 30).
- La **tipología de juegos** que el usuario ha jugado en el último año, que puede ser variada (un poco de todos los tipos de juego) mayormente de acción (si la mayoría de partidas jugadas son de juegos de tipo arcade o lucha), mayormente de ingenio (si la mayoria de juegos son de tipo puzzle, aventura gráfica o simulador de mundo), o mayormente competitivo (si la mayoría de juegos son de tipo deporte o simulador de vehículo).

A partir de estas características queremos adaptar la partida actual modificando algunos de sus parámetros para ajustar la experiencia de juego a lo que el usuario parece necesitar o preferir:

- *Ajustar el número de enemigos*: si la salud del personaje es alta o el tiempo de reacción es rápido y la tipología de juegos es mayormente de acción o mayormente competitivo entonces hay que incrementar el número de NPCs enemigos; si la salud es normal y el tiempo de reacción es normal o alto y la tipología de juegos es mayormente de acción entonces hay que incrementar el número de NPCs enemigos; si la salud es muy baja o si el tiempo de reacción es lento entonces hay que reducir el número de NPCs enemigos; en los otro casos no modificar los enemigos.
- *Ajustar el nivel de la IA de los enemigos*: si la salud del personaje es alta o el tiempo de reacción es rápido y la tipología de juegos es mayormente de aventuras o mayormente competitivo entonces incrementar el nivel de la IA de los NPCs enemigos; si la salud es normal y el tiempo de reacción es normal o alto y la tipología de juegos es mayormente competitivo entonces hay que incrementar el nivel de la IA de los NPCs enemigos; si la salud es muy baja o si el tiempo de reacción es lento entonces hay que reducir el nivel de la IA de los NPCs enemigos; en los otro casos no modificar la IA de los enemigos.
- *Ajustar misiones de equipo*: si el nivel de trabajo en equipo es bajo y el tiempo de narración es normal o elevado entonces introducir hitos argumentales con tema de misión grupal; si el nivel de trabajo en equipo es elevado y el tiempo de narración es elevado entonces introducir hitos argumentales con tema misión individual; si el tiempo de narración es mínimo entonces eliminar todos los hitos argumentales con tema misión individual o misión grupal que no sean centrales; en el resto de casos no variar.
- *Ajustar misterios*: si la tipología de juegos es mayormente de ingenio y el tiempo de narración es elevado o el tiempo de exploración es elevado entonces introducir hitos argumentales con tema misterio; si el tiempo de narración es bajo o el tiempo de exploración es bajo entonces reducir hitos argumentales con tema misterio; si el tiempo de narración es mínimo entonces eliminar todos los hitos argumentales con tema misterio que no sean centrales; en el resto de casos no variar.
- *Ajustar relaciones*: si el nivel de interacción con otros personajes es bajo o muy bajo y el tiempo de narración es normal o elevado y el tiempo de exploración es normal o elevado entonces introducir hitos argumentales con tema relación personajes; si el nivel de interacción con otros personajes es normal y el tiempo de narración es bajo entonces reducir hitos argumentales con tema relación personajes; si el tiempo de narración es mínimo entonces eliminar todos los hitos argumentales con tema relación personajes que no sean centrales; en el resto de casos no variar.

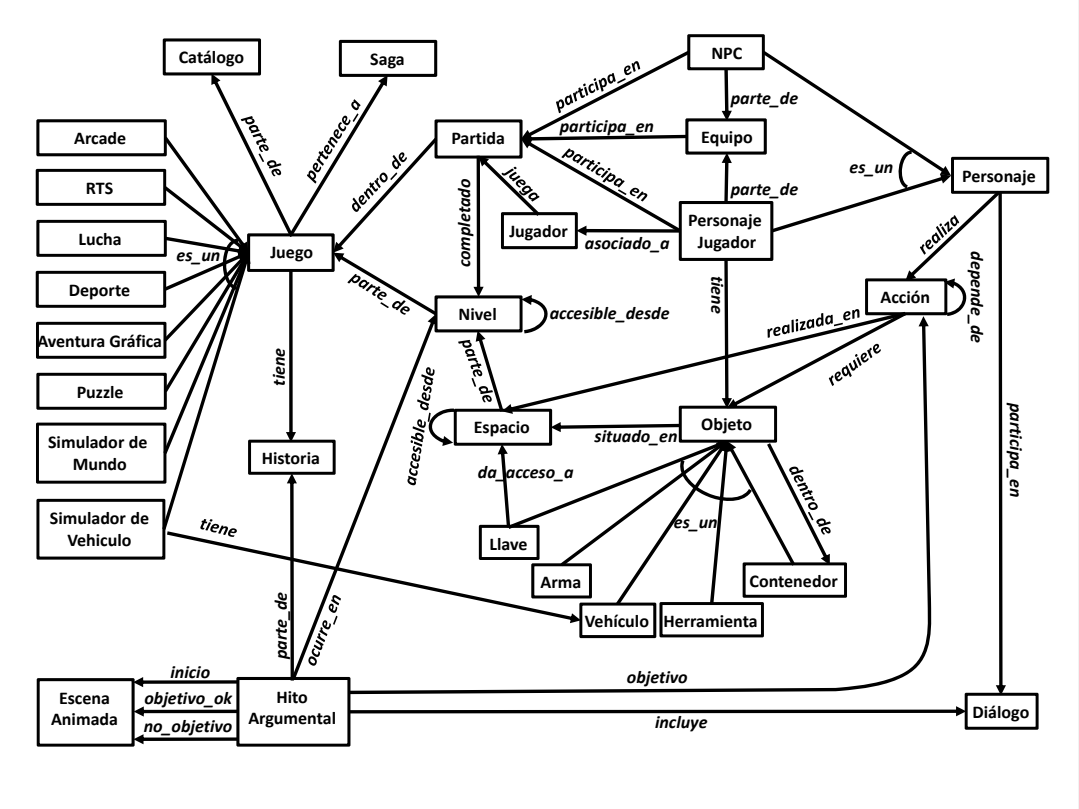
La salida de este sistema de adaptación ha de sugerir exactamente que modificaciones hay que introducir en el juego (número de enemigos a añadir/eliminar, niveles de IA que aumentar/disminuir, que hitos hay que añadir o eliminar, etc...).

a. Diseña la ontología del dominio descrito, incluyendo todos los conceptos que aparecen en la descripción e identificando los atributos más relevantes. Lista que conceptos forman parte de los datos de entrada del problema y que conceptos forman parte de la solución. (Nota: tened en

cuenta que la ontología puede necesitar modificaciones para adaptarla al apartado siguiente).

En este problema la ontología ha de incorporar, como mínimo, todos los conceptos que forman parte de la entrada del problema: la información que se obtiene directamente de la base de datos de la empresa de videojuegos y la información derivada. Todos los conceptos del diagrama forman parte de la entrada del problema, y algunos de ellos se ven afectados o modificados por la solución concreta final (`NPC` e `Hito Argumental` son los conceptos afectados directamente por la solución, pero es posible que todo concepto que esté directamente relacionado (`Equipo`, `Dialogo`, `Acción`, `Objeto`, `Escena` ...) también se vean afectados (por ejemplo, si hay que añadir más NPC's enemigos eso puede afectar a uno o más equipos, al introducir más personajes puede que se creen nuevos dialogos y nuevas acciones, que requieran espacios y objetos nuevos, etc).

De la ontología resultante cabe remarcar la distinción entre `Jugador` (el usuario o persona del mundo real que juega con los juegos de la compañia) y `PersonajeJugador` (uno de los personajes con los que ek jugador participa dentro del juego). A la hora de perfilar el usuario separaremos la información de la partida actual o bien en la partida actual o en el/los personaje/s que tenga el usuario con información del perfil que cubre varias partidas, que estará dentro del concepto `Jugador`. Tambiñen cabe remarcar que existen varios conceptos que comparten atributos y/o relaciones, y que esto ha sido algo que se ha tenido muy en cuenta a la hora de crear super-clases con las características comunes, o para decidir entre crear subclases o solo añadir un atributo `tipo`. La regla general es que hemos creado subclases cuando una o varias tienen atributos diferentes y/o relaciones diferentes. Ese es el caso de las subclases de `Juego`, `Personaje` y `Objeto`. Siguiendo ese criterio no se crearon subclases de `Hito Argumental` o de `Vehículo`. También se ha intentado seguir el heurístico del diseño de ontologías que recomienda que ramas hermanas de la taxonomía tengan un nivel de granularidad similar, y por ello, aunque `Vehículo` y `Arma` comparten un atributo de energía, no se ha creado una superclase `ObjetoConEnergía` para no tener una jerarquia de objetos con diferents niveles en las ramas.Otra cosa importante a destacar es que siempre que el enunciado nos dice que un concepto A se compone de B hemos usado la relación taxonómica B `parte_de` A, que modela correctamente como un concepto se compone de sus partes. Y que hemos colocado exactamente tres relaciones entre `Hito Argumental` y `Escena` que establece las tres posibles animaciones asociadas a la escena.



Atributos: a continuación se listan los atributos mínimos para representar la información mencionada explícitamente en el enunciado y la necesaria para el apartado b) del problema.

- `Juego`: título (string), año_publicación (entero), edad_recomendada (enumeración: {+0años, +3años, +5años, +8años, +12años, +18años}), online? (booleano), multijugador (booleano);
- `Saga` título (string);
- `Partida`: fecha_inicio (fecha), hora_inicio (hora), duración (minutos), AJUSTAR_NUM_ENEMIGOS (enumeración: {incrementar, mantener, reducir}), AJUSTAR_NIVEL_IA_ENEMIGOS (enumeración: {incrementar, mantener, reducir}), AJUSTAR_MISIONES_EQUIPO (enumeración: {incrementar_grupal, incrementar_individual, mantener, reducir_grupal, reducir_individual, eliminar_extras}), AJUSTAR_MISTERIOS (enumeración: {incrementar, mantener, reducir, eliminar_extras}), AJUSTAR_RELACIONES (enumeración: {incrementar, mantener, reducir, eliminar_extras});
- `Jugador`: username (string), crédito_total (real), *Tiempo_Exploración* (enumeración: {mínimo, normal, elevado}), *Tiempo_Narración* (enumeración: {mínimo, normal, elevado}), *Nivel_Trabajo_Equipo* (enumeración: {muy bajo, bajo, normal, elevado}) *Interacción_con_Otros_Personajes* (enumeración: {muy baja, baja, normal, elevada}) *Tipología_Juegos* (enumeración: {variada, may_acción, may_ingenio, may_competitivo});
- `Personaje`: nombre (string), rol (string), salud (%);
- `NPC`: enemigo? (booleano), nivel_IA (enumeración: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10});
- `PersonajeJugador`: puntos_obtenidos (real), crédito_obtenido (real), *Nivel_Salud* (enumeración: {muy bajo, bajo, normal, elevado}), *Tiempo_Reacción* (enumeración: {rápido, normal, lento});
- `Equipo`: nombre (string);
- `Acción`: nombre (string), fecha_inicio (fecha), hora_inicio (hora), tiempo_reacción (milisegundos);

- `Diálogo`: id_diálogo (string);
- `Nivel`: nombre (string), dificultad (enumeración: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}), tiempo_máximo (minutos);
- `Espacio`: nombre (string), interior? (booleano);
- `Objeto`: nombre (string), precio_en_créditos (real);
- `Arma`: nivel_daño (real), energia_disponible (%);
- `Vehículo`: tipo (enumeración: {aereos, espaciales, marinos, anfibios, terrestres}), número_plazas (entero), energia_disponible (%);
- `HitoArgumental`: tema (enumeración: {misión_individual, misión_grupal, relación_personal, misterio}), central? (booleano), fecha (fecha_max_objetivo), hora_max_objetivo (hora);
- `Escena`: id_video (string), duración (segundos);

Como se puede ver, hemos decidido también representar las características abstractas del problema, las de la solución abstracta y la solución concreta (son los que tienen el nombre en *Cursiva*, *CURSIVA* o MAYUSCULAS, respectivamente). De esta manera todos los conceptos que aparecerán en las reglas están soportados por la ontología.

b. El problema descrito es un problema de análisis. Explica cómo lo resolverías usando clasificación heurística, usando los conceptos de la ontología desarrollada en el apartado anterior. Da al menos 4 ejemplos de reglas para cada una de las fases de esta metodología.

Para resolverlo mediante clasificación heurística debemos identificar en el problema las diferentes fases y elementos de esta metodología. En este caso hay solo una opción posible: la solución que pide el enunciado solo puede ser una solución concreta, ya que es imposible poder calcular en el nivel de solución abstracta exactamente cuantos enemigos hay que añadir al juego, o que HitoArgumental en concreto vamos a añadir o eliminar sin acceder a los datos concretos como el porcentaje exacto de salud del personaje jugador o cuantos enemigos hay ya cerca del Personaje.

Una vez sabemos cuantas fases requiere nuestra solución ya podemos enunciar la solución completa.

El primer elemento son los problemas concretos que hay que tratar. En este dominio los problemas concretos están definidos por toda la información detallada que la compañia tiene del jugador en todos los juegos de su catálogo y que se ha identificado en la ontología del apartado anterior. Tal y como dice el enunciado, supondremos la existencia de funciones (como tiempo_medio_reacción_partida) que o bien calculan bajo demanda cierto estadístico o van a buscar el valor precalculado en alguna base de datos.

El segundo elemento son los problemas abstractos, estos estarán definidos a partir de las siete características que menciona el enunciado (*Nivel_Salud*, *Tiempo_Reacción*, *Tiempo_Exploración*, *Tiempo_Narración*, *Nivel_Trabajo_Equipo*, *Interacción_con_Otros_Personajes* y *Tipología_Juegos*), todas ellas con rangos de valores discretos.

Para conectar los problemas concretos con los abstractos necesitamos definir las reglas de abstracción de datos, por ejemplo:

- si PersonajeJugador.salud $< 10\%$ entonces Nivel_Salud = muy bajo;
- si PersonajeJugador.salud $> 80\%$ entonces Nivel_Salud = elevado;
- si tiempo_medio_reacción_partida(PersonajeJugador) < 0.4 seg entonces Tiempo_Reacción = rápido;
- si tiempo_medio_reacción_partida(PersonajeJugador) ≥ 0.9 seg entonces Tiempo_Reacción = lento;
- si tiempo_medio_exploración_hoy(Jugador) < 5 seg entonces Tiempo_Exploración = mínimo;

- si tiempo_medio_exploración_hoy(Jugador) ≥ 50 seg
  entonces Tiempo_Exploración = elevado;
- si tiempo_medio_narración_hoy(Jugador) < 3%
  entonces Tiempo_Narración = rápido;
- si porcentaje_partidas_equipo_jugadas(Jugador) < 10%
  entonces Nivel_Trabajo_Equipo = bajo;
- si num_dialogos_jugador_mes(Jugador) ≥ 30
  entonces Interacción_con_Otros_Personajes = elevada;
- si (porcentaje_partidas(Jugador,arcade) + porcentaje_partidas(Jugador,lucha)) ≤ 60%
  entonces Tipología_Juegos = may_acción;
- si (porcentaje_partidas(Jugador, SimuladorVehículo)
  + porcentaje_partidas(Jugador, Deporte)) ≤ 60%
  entonces Tipología_Juegos = may_competitivo;

El tercer elemento son las soluciones abstractas. En este caso tenemos solo cinco soluciones abstractas: AJUSTAR_NUM_ENEMIGOS, AJUSTAR_NIVEL_IA_ENEMIGOS, AJUSTAR_MISIONES_EQUIPO , AJUSTAR_MISTERIOS y AJUSTAR_RELACIONES.

Para ligar los problemas abstractos con las soluciones abstractas necesitaremos reglas de asociación heurística, como por ejemplo:

- si (Nivel_Salud == alta o Tiempo_Reacción == rápido)
  y Tipología_Juegos==(may_acción o may_competitivo)
  entonces Partida.AJUSTAR_NUM_ENEMIGOS=incrementar;
- si Nivel_Salud == normal y Tiempo_Reacción == normal
  y Tipología_Juegos==may_acción
  entonces Partida.AJUSTAR_NUM_ENEMIGOS=incrementar;
- si Nivel_Salud == muy bajo y Tiempo_Reacción == lento
  entonces Partida.AJUSTAR_NUM_ENEMIGOS=reducir;
- si (Nivel_Salud == alta o Tiempo_Reacción == rápido)
  y Tipología_Juegos==(may_aventuras o may_competitivo)
  entonces Partida.AJUSTAR_NIVEL_IA_ENEMIGOS=incrementar;
- si Nivel_Salud == normal y Tiempo_Reacción == normal
  y Tipología_Juegos==may_competitivo
  entonces Partida.AJUSTAR_NIVEL_IA_ENEMIGOS=incrementar;
- si Nivel_Salud == muy bajo y Tiempo_Reacción == lento
  entonces Partida.AJUSTAR_NIVEL_IA_ENEMIGOS=reducir;
- si Nivel_Trabajo_Equipo == muy bajo y Tiempo_Reacción == (normal o elevado)
  entonces Partida.AJUSTAR_MISIONES_EQUIPO=incrementar_grupal;
- si Tipología_Juegos == may_ingenio y
  (Tiempo_Narración == elevado o Tiempo_Exploración == elevado)
  entonces Partida.AJUSTAR_MISTERIOS=incrementar;
- si Tiempo_Narración == mínimo entonces
  (Partida.AJUSTAR_MISIONES_EQUIPO=eliminar_extras
  y Partida.AJUSTAR_MISTERIOS=eliminar_extras
  y Partida.AJUSTAR_RELACIONES=eliminar_extras)

El cuarto y último elemento son las soluciones concretas. En este caso corresponde al cálculo de los ajustes exactos a aplicar en la partida actual, echando mano de la solución abstracta y de los datos concretos del problema. Supondremos la existencia de funciones que combinan la solución abstracta con datos concretos del problema y aplican directamente cambis a la partida actual. Lo que sigue son algunos ejemplos de reglas:

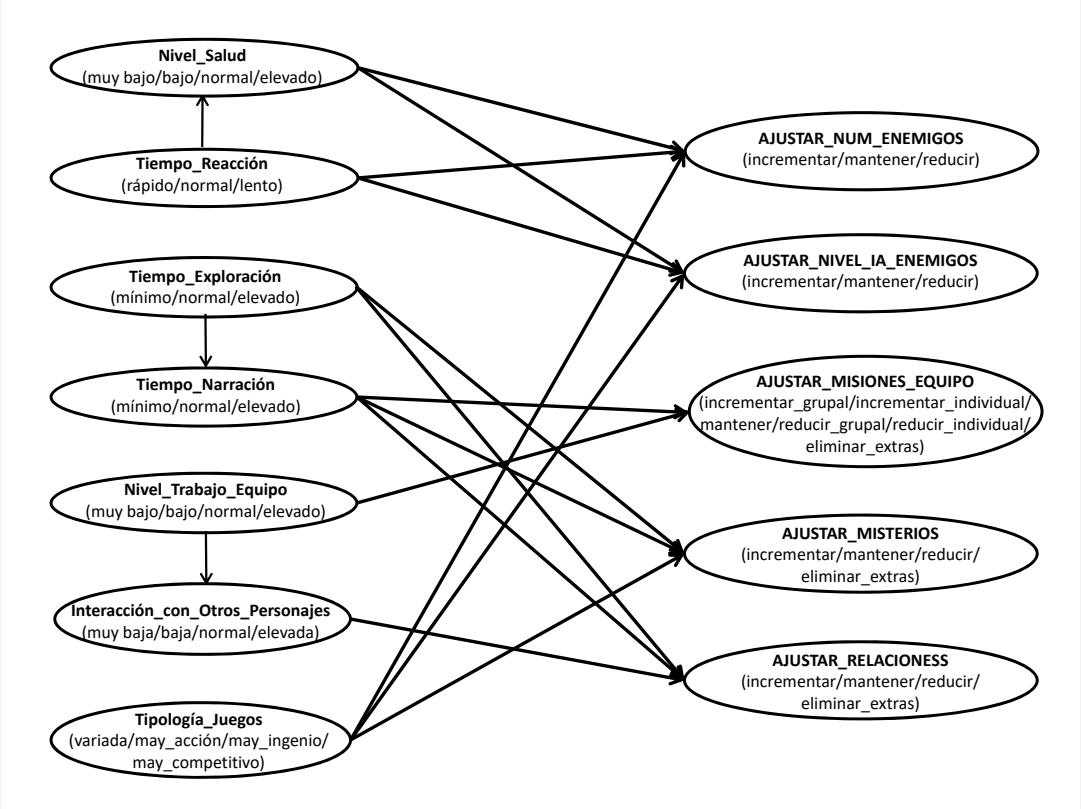
- si Partida.AJUSTAR_NUM_ENEMIGOS==reducir entonces reduce_num_Enemigos(PersonajeJugador.salud, PersonajeJugador.tiempo_medio_reacción);
- si Partida.AJUSTAR_NUM_ENEMIGOS==incrementar entonces aumenta_num_Enemigos(PersonajeJugador.salud, PersonajeJugador.tiempo_medio_reacción);
- si Partida.AJUSTAR_NIVEL_IA_ENEMIGOS==incrementar entonces aumenta_IA_Enemigos(PersonajeJugador.salud, PersonajeJugador.tiempo_medio_reacción);
- si Partida.AJUSTAR_MISIONES_EQUIPO==incrementar_grupal entonces añade_Hito_Argumental(PersonajeJugador, Nivel);
- si Partida.AJUSTAR_RELACIONES==eliminar_extras entonces elimina_Hitos_Argumentales_extra(PersonajeJugador, Historia);

c. Las características que se usan durante el proceso de adaptación del videojuego no son independientes entre sí. Los tiempos de reacción suelen influir en la salud del personaje (con tiempos de reacción lentos la salud suele empeorar); el tiempo de exploración suele influir en el tiempo de narración (un usuario que dedica mayor tiempo a explorar el entorno suele tener más paciencia o curiosidad por ver los videos enteros, y por ello el tiempo de narración es mayor, mientras que un tiempo de exploración menor suele asociarse a un jugador con menos tiempo disponible o menos curiosidad, y por lo tanto tener un tiempo de narración menor); el nivel de trabajo en equipo influye en la interacción con otros personajes (a mayor trabajo en equipo, mayor interacción con otros personajes). Define el problema de asociación heurística como una red bayesiana expresando en ella al menos las relaciones indicadas no solo en este apartado sinó en el resto del enunciado, de forma que todas las características abstractas del problema que hayas definido en el apartado anterior tengan algun tipo de influencia en la solución. Separa bien en el diagrama que variables describen características de problema y cuales describen soluciones. Lista de forma clara los diferentes valores que puede tomar cada variable. Da un ejemplo de tabla de probabilidad de algún nodo, inventándote las probabilidades, pero expresando como influyen los valores de los nodos padre en las probabilidades de los valores de los nodos hijo.

Esta claro que en el gráfico deberíamos tener al menos dos grupos de nodos, los que corresponden a las características abstractas del jugador que se obtienen en la fase de abstracción, y los que corresponden a las características de la solución abstracta. Nuestro objetivo es construir una red que conecte características del problema abstracto a características de la solución abstracta.

La figura a continuación corresponde a la solución propuesta, que está alineada con las características abstractas del apartado anterior. Los nodos de la solución serán los nodos finales de la red bayesiana (en este caso cinco nodos con los 5 tipos de ajuste de los que consta la solución abstracta). Los nodos a su izquierda son las 7 características abstractas que modelan al jugador. En este caso no ha hecho falta hacer ninguna adaptación especial de las características del apartado anterior, ya que todas ellas tenian ya un rango de valores discretos.

Respecto a las dependencias, representaremos las dependencias entre nodos, ya sean de la solución o del problema abstracto. Es muy importante representar en la red cómo dependen los nodos solución de los nodos del problema abstracto, ya que ese es el objetivo principal de la fase de asociación heurística.



Una tabla de probabilidad simplemente ha de asignar más probabilidad a valores más correlacionados entre grupos de variables. Por ejemplo, si escogemos una variable como Ajustar_Nivel_IA_Enemigos (que depende de Nivel_Salud, Tiempo_Reacción y Tipología_Juegos), la tabla de probabilidad podría ser algo como lo siguiente:

			Ajustar_Nivel_IA_Enemigos		
Nivel_Salud	Tiempo_Reacción	Tipología_Juegos	incrementar	mantener	reducir
muy baja	lento	may_ingenio	0,0	0,01	0,99
muy baja	lento	variado	0,0	0,03	0,97
muy baja	lento	may_competitivo	0,0	0,04	0,96
muy baja	lento	may_acción	0,0	0,05	0,95
⋮	⋮	⋮	⋮	⋮	⋮
baja	lento	may_acción	0,0	0,2	0,8
⋮	⋮	⋮	⋮	⋮	⋮
normal	normal	may_ingenio	0,1	0,9	0,0
⋮	⋮	⋮	⋮	⋮	⋮
alto	rápido	may_competitivo	0,95	0,05	0,0
alto	rápido	may_acción	0,99	0,01	0,0

La tabla intenta reflejar, por ejemplo, que en igualdad de nivel de salud y tiempo de reacción, el nivel de IA del juego será algo mayor en jugadores cuya tipología de juego sea mayormente de acción, y va decreciendo progresivamente en el caso de mayormente competitivos, variados y

> mayormente ingénio. También refleja que a mayor salud y tiempo de reacción más alto, maror el incremento de la IA, y a menor salud y peor tiempo de reacción, la reducción del nivel de IA aumenta. Es importante asegurarse que la suma de los valores de cada fila de la tabla sumen 1.

5.1.3 Problema 13

La cadena de supermercados *HIPERFUR* nos pide que diseñemos un SBC orientado a la creación de folletos de ofertas de productos personalizados para cada cliente. Cada folleto consta de 8 páginas y en cada página se pueden colocar 9 ofertas diferentes, que provienen de una base de ofertas a la que tendrá acceso el SBC. Toda oferta tiene una fecha de inicio y de fin, y un producto o productos asociados. Hay dos tipos de ofertas:

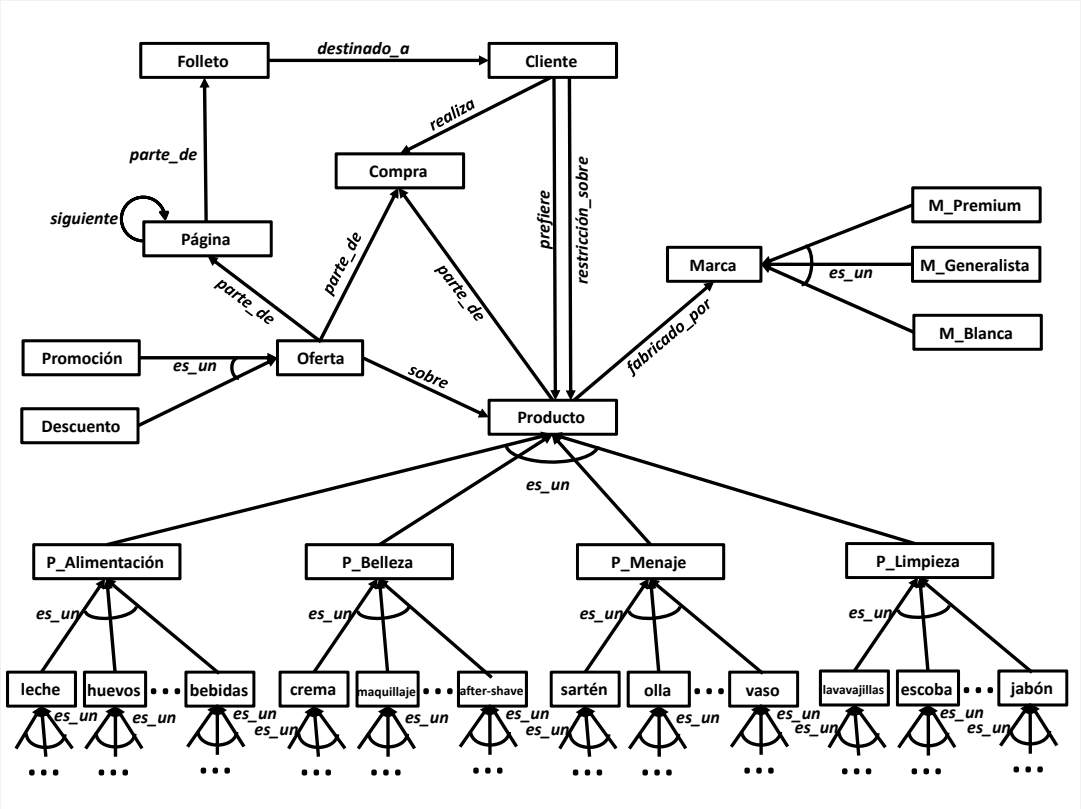
- *descuentos*, que tienen entre otros datos el porcentaje de descuento, e información sobre si son descuentos directos (el cliente paga el precio rebajado) o indirectos (el cliente recibe un vale de compra).
- *promociones*, que pueden ser packs de productos diferentes (por ejemplo, un queso y un jamón) u ofertas al llevarse varias unidades del mismo producto (2x1, 3x2, 4x3).

Los productos pueden ser de diferente tipologia: alimentación (leche, huevos, pescado, carne, bebidas...), belleza (cremas, maquillaje, after-shave...), menaje (sartenes, ollas, vasos, cuberteria...), limpieza del hogar (jabon de ropa, lavavajillas, quitamanchas, mocho, escoba...). Además los productos pueden pertenecer a una marca blanca (precio muy bajo), a una marca generalista (precio medio) o a una marca premium (precios muy altos). Este sistema obtendrá la información de los hábitos de compra de un cliente (una lista de las ofertas y los productos individuales que ha comprado en los últimos 6 meses donde, para cada oferta y para cada producto individual, se dice cuantas unidades ha comprado. Además los clientes pueden (a través de la página web) añadir una lista de productos preferidos , y/o una lista de restricciones que se pueden aplicar a un producto concreto (ej: no quiero el jabón "Pixan plus") o a un subtipo de producto (ej: no me gustan las leches de soja). Se quiere construir este SBC de manera que, en base a toda la información disponible sobre el cliente, seleccione un conjunto de ofertas adecuadas para ese tipo de cliente y las asigne a alguna de las 8 páginas del folleto. El director de marketing de *HIPERFUR* nos recomienda que incluyamos las siguientes características abstractas como base de la selección: el nivel de **gasto mensual** (alto, medio, bajo) que servirá para calcular el importe total de los productos que se coloquen en el folleto; la **calidad** de los productos (mayoria premium, mayoria normal, mayoria marca blanca) que servirá para seleccionar, dentro de un (sub)tipo de producto, el producto adecuado al patrón de compra del cliente; la **cantidad_alimentación** (exagerada, mucha, normal, poca, limitada), que servirá para decidir que porcentaje del folleto se dedica a productos alimenticios (y en los casos extremos -exagerada, limitada- se colocarán preferentemente productos de alto valor nutricional);la **cantidad_belleza** (narcicista, normal, dejado/a) que servirá para decidir que porcentaje del folleto se dedica a productos de belleza; la **cantidad_menaje** (exagerado/a, normal, minimalista) que servirá para decidir que porcentaje del folleto se dedica a productos de menaje; y la **cantidad_limpieza** (obseso/a, normal, sucio/a) que servirá para decidir que porcentaje del folleto se dedica a productos de limpieza.

1. Diseña la ontología del dominio descrito, incluyendo todos los conceptos que aparecen en la descripción e identificando los atributos más relevantes. Lista qué conceptos forman parte de los datos de entrada del problema y qué conceptos forman parte de la solución. (Nota: tened en cuenta que la ontología puede necesitar modificaciones para adaptarla al apartado siguiente).

> En este problema la ontología ha de incorporar, como mínimo, todos los conceptos que forman parte de la entrada del problema, así como conceptos que formen parte de la solución.
> Los conceptos que forman parte de la entrada del problema son todos referidos a características del cliente al que queremos generarle el folleto personalizado:
> - las compras que ha realizado
> - el tipo de los productos que ha comprado
> - la marca y calidad de los productos que ha comprado
> - si ha sacado provecho de descuentos u ofertas anteriores

Los conceptos que forman parte de la salida del problema son el folleto, las páginas del folleto y toda la información relevante sobre los descuentos y promociones que se incluirán en el folleto. En este caso, como se ha de tener la información de los productos que se incluyen en el folleto, todos los conceptos excepto `Compra` pertenecen a la solución.



No hay mucho que remarcar sobre la ontología resultante, ya que el enunciado daba bastante detalle. Se ha optado por indicar en el diagrama que de algunos conceptos (como `P_Alimentación`, `P_Belleza`, `P_Limpieza` o `P_Menaje`) seria necesario especificar sus subclases, que en el diagrama solo indicamos con los puntos suspensivos. También hemos reducido el valor nutritivo de un producto de alimentación a un número, pero para hacerlo más realista deberiamos tener la tabla nutricional completa.

Atributos:

- `Cliente`: nombre (string), dirección (string)
- `Producto` : nombre (string), referencia (string), importe (real positivo),
- `Oferta`: fecha_inicio (fecha), fecha_fin (fecha)
- `Descuento`: porcentaje (entero positivo), directo (booleano)
- `P_Alimentación`: valor_nutritivo (real positivo)

2. El problema descrito es un problema de análisis. Explica cómo lo resolverías usando clasificación heurística, usando los conceptos de la ontología desarrollada en el apartado anterior. Da al menos 3 ejemplos de reglas para cada una de las fases de esta metodología.

En este caso el enunciado ya nos propone un conjunto finito de características abstractas para describir el problema. Para resolverlo mediante clasificación heurística, deberiamos tener además un conjunto enumerable de soluciones candidatas. Por ello plantearemos la solución partiendo de la hipótesis que existe un conjunto enumerable a priori de posibles proporciones de los diferentes tipos de producto en el folleto, dependiendo de los valores discretos de las características abstractas que nos ha dado el director de marketing (por ejemplo, podemos suponer que nos han dado una tabla con todas las combinaciones de valores para los cinco

criterios y un reparto de porcentajes de folleto ocupados por cada tipo de producto).
A partir de este planteamiento lo que nos queda es describir las fases y elementos necesarios para aplicar la clasificación heurística al problema.
El primer elemento son los problemas concretos que hay que tratar. En este dominio los problemas concretos están definidos por los datos concretos del cliente y su historial de compra que son la entrada del problema.

El segundo elemento son los problemas abstractos, que describen al cliente mediante las 5 características abstractas (**gasto mensual**, **calidad**, **cantidad_alimentación**, **cantidad_belleza**, **cantidad_menaje** y **cantidad_limpieza**).
Para conectar los problemas concretos con los abstractos necesitamos definir las reglas de abstracción de datos, por ejemplo:

- si 400 > $\sum$ Producto.importe > 200 entonces gasto_mensual=medio;
- si $\sum$ P_Belleza.importe < 10 entonces cantidad_belleza=dejado_a;
- si $\sum$ P_Limpieza.importe >= 400 entonces cantidad_limpieza=obseso_a;

El tercer elemento son las soluciones abstractas. En este caso el enunciado nos dice que el director de marqueting nos sugiere calcular unos porcentajes del espacio del folleto que se dedica a cada tipo de producto, y eso es lo que haremos. Además iremos seleccionando el tipo de ofertas que pueden ser más adecuadas para el cliente (descuentos, promociones).
Ejemplos de reglas de asociación heurística que ligan los problemas abstractos con las soluciones abstractas serian las siguientes:

- si cantidad_alimentación=poca y cantidad_belleza=narcicista
  y cantidad_limpieza=normal y cantidad_menaje=minimalista
  entonces porcentaje(alimentación, 10 %) y porcentaje(belleza, 50 %)
  y porcentaje(limpieza, 25 %) y porcentaje(menaje, 15 %)
- si gasto_mensual=alto y calidad=mayoria_premium)
  entonces tipo_ofertas=pack
- si gasto_mensual=bajo y calidad=mayoria_marca_blanca)
  entonces tipo_ofertas=desc_directo

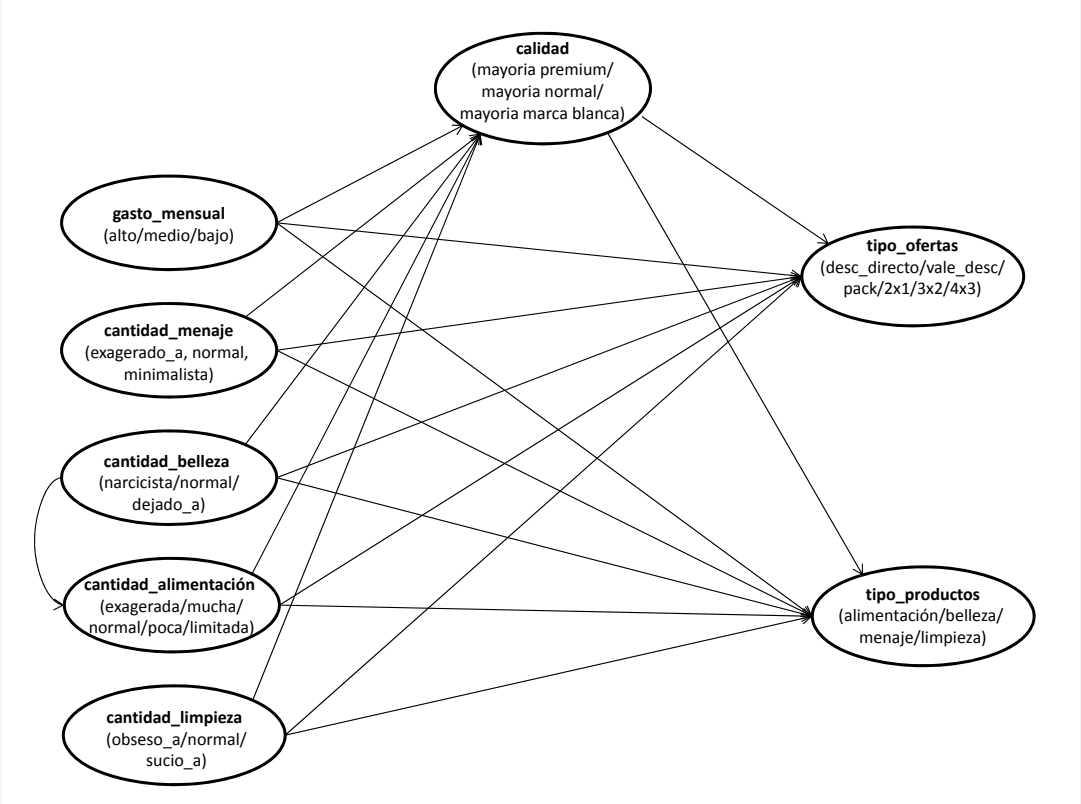
El cuarto elemento son las soluciones concretas. En este caso son soluciones completas en las que se tiene una serie de productos asignados a las páginas del folleto, teniendo en cuenta no solo las proporciones que vienen de la fase anterior sino también las preferencias y restricciones del cliente que nos vienen como entrada.
Algunos ejemplos de como refinar la solución son los siguientes:

- poner en portada los productos preferidos del usuario
- eliminar de la lista de productos posibles todos los que estan en las restricciones del usuario
- colocar primero los productos del tipo que tiene mayor proporción, y luego el siguiente y asñi succesivamente
- cuando haya varios productos posibles del mismo tipo a escoger en una página, seleccionar
  aquellos que el usuario compra más.
- en el caso de usuarios con un volumen de compras de alimentación muy bajo añadiremos primero
  productos de alimentación con un alto valor nutricional.

3. Las características que se utilizan para determinar las ofertas a incluir en el folleto no son independientes entre si. Por ejemplo: la cantidad de productos de belleza suele ser inversamente proporcional a la cantidad de alimentación (cuanto más cuida su aspecto, menos come), la calidad esta influida por el nivel de gasto mensual y las cantidades de cada tipo de producto. El tipo de ofertas a incluir viene influenciado por el nivel de gasto y la cantidad (por ejemplo, en caso de niveles de gasto bajos o cantidades bajas suele ser mejor incluir descuentos, mientras que para clientes que compran grandes cantidades o con niveles de gasto elevado lo mejor es recomendar

promociones multi-producto, etc).
Define el problema de asociación heurística como una red bayesiana expresando en ella al menos las relaciones indicadas y alguna más. Da un ejemplo de tabla de probabilidad de algún nodo, inventándote las probabilidades, pero expresando como influyen los valores de los nodos padre en las probabilidades de los valores de los nodos hijo.

Esta claro que tenemos dos grupos de nodos, los que corresponden a las características abstractas del cliente que se obtienen en la fase de abstracción, y los que corresponden a las características de la solución abstracta. Nuestro objetivo es construir una red que conecte características del problema abstracto a características de la solución abstracta. Los nodos de la solución serán los nodos finales de la red bayesiana (tipo_ofertas y tipo_productos). Los nodos iniciales de la red serán los del problema abstracto (gasto_mensual, calidad, cantidad_alimentación, cantidad_belleza, cantidad_menaje y cantidad_limpieza). En la solución se ha decidido tener una sola variable tipo_producto con cuatro posibles valores (alimentación, belleza, menaje, limpeza) en vez de tener una variable por cada tipo porque de esta forma aprovecharemos al máximo la capacidad de razonamiento aproximado de la red bayesiana: los porcentajes que nos dé la red para cada valor de la variable tipo_producto serán ya los porcentajes que necesitamos para organizar el espacio en el folleto. Por una razón similar hemos agrupado todos los descuentos y las promociones como valores de una sola variable tipo_oferta: la red nos calculará en que proporción ha de aparecer cada uno de los tipos de oferta en el folleto.
Respecto a las dependencias, representaremos las dependencias entre nodos, ya sean de la solución o del problema abstracto. Es muy importante representar en la red cómo dependen los nodos solución de los nodos del problema abstracto, ya que ese es el objetivo principal de la fase de asociación heurística. Pero también es bueno representar dependencias entre nodos del problema abstracto si la intuición (y el enunciado) nos dicen que son dependientes (por ejemplo, la cantidad_alimentación depende de la cantidad_belleza, la calidad depende del gasto_mensual y de todas las cantidades) ya que eso permite calcular en casos en los que faltan datos el valor de una de esas variables respecto a las otras, haciendo el sistema más robusto.



Una tabla de probabilidad simplemente ha de asignar más probabilidad a valores más co-

rrelacionados entre grupos de variables. Por ejemplo, si escogemos una variable como cantidad_alimentación (que depende de cantidad_belleza), la tabla de probabilidad podría ser algo como lo siguiente:

	cantidad_alimentación				
cantidad_belleza	exagerada	mucha	normal	poca	limitada
narcicista	0,03	0,07	0,2	0,3	0,4
normal	0,05	0,25	0,45	0,2	0,05
dejado	0,25	0,45	0,2	0,07	0,4

La tabla intenta reflejar que la probabilidad de consumir gran cantidad de productos de alimentación es inversamente proporcional a la cantidad de productos de belleza que se consume.
Si se escoge una variable con más dependencias, como tipo_productos (que depende de gasto_mensual, calidad y las cuatro variables de cantidad) tendriamos una tabla muy grande. Por cuestiones de espacio solo pondremos parte de la tabla:

gasto_ mensual	calidad	cantidad_ alimentación	...	cantidad_ menage	tipo_productos alimen tación	be lleza	lim pieza	me nage
⋮	⋮	⋮	...	⋮	⋮	⋮	⋮	⋮
alto	premium	poca	...	minimalista	0,1	0,5	0,25	0,15
⋮	⋮	⋮	...	⋮	⋮	⋮	⋮	⋮
medio	normal	normal	...	normal	0,25	0,25	0,25	0,25
⋮	⋮	⋮	...	⋮	⋮	⋮	⋮	⋮
bajo	dejado_a	limitada	...	minimalista	0,4	0,25	0,25	0,1
⋮	⋮	⋮	...	⋮	⋮	⋮	⋮	⋮

5.1.4 Problema 14

La empresa de pirotécnica FantasIA quiere diseñar un sistema inteligente de gestión de pedidos de material a través de su página web que ayude al cliente a escoger el material pirotécnico que desea recomendándole el más adecuado según sus preferencias, el evento y el lugar donde se celebrará. Coincidiendo con las fiestas de San Juan de este año quiere probar un primer prototipo del sistema y ver si gusta a sus clientes.

Los usuarios de este sistema podran ser empresas o ayuntamientos. De las empresas podemos obtener información sobre si es una empresa *pequeña*, *mediana* o *grande*, y de los ayuntamiento podemos saber si se trata de una *metrópoli* (una ciudad grande), una *ciudad pequeña*, una *villa* o un *pueblo* (cuanto más grande es la empresa o el ayuntamiento más puede invertir en material pirotécnico). Tanto en el caso de las empresas como de los ayuntamientos el sistema puede acceder al histórico de pedidos para saber que tipo de material han solicitado en el pasado y usarlo en la recomendación. Del evento obtendremos información sobre el número de asistentes que se preveen y el tipo de espacio en el que se va a usar el material (*espacio abierto*, *espacio forestal*, *parque*, *plaza urbana*, *calle*...). Para todos los espacios sabremos los m^2 de superficie libre de los que se dispone, y en el caso de los espacios forestales y los parques sabremos además el *grado de humedad* de la vegetación. También interesa que el usuario haga un listado de los objetos y estructuras en las cercanias del lugar que podrian recibir algun daño (*edificios*, *naves industriales*, *mobiliario urbano*, *coches*) dando el número de objetos de cada tipo, su distancia al lugar en el que se usará el material y, en el caso de las naves industriales, si contienen productos altamente inflamables en su interior. El sistema preguntará si se tiene contratada

una *dotación de bomberos* o no, y si la persona que va a usar el materia l es un *especialista pirotécnico* certificado o no. Accediendo a un servicio web el sistema obtendrá además una *previsión metereológica* de la zona en la que se celebrará el evento (temperatura máxima y mínima, humedad relativa, velocidad del viento, probabilidad de lluvia). Y obviamente en todo pedido tendremos el *presupuesto máximo* que quiere gastar al cliente.

A partir de toda esta información el sistema ha de recomendar una serie de productos de su catálogo de material pirotécnico: *material básico* (pequeños cohetes y petardo que puede manipular cualquier persona), *truenos* (petardos de una cierta potencia), *cohetes* (solo suben y explotan), *luces de color* y *luces fantasia* (lo más avanzado, al abrirse dibujan formas en el aire). El material básico es lo más barato y menos peligroso, mientras que las luces fantasia son lo más caro, y ambos tipos de luces son el material más peligroso. Para facilitar la recomendación, la empresa ya tiene preparados unos surtidos o packs de productos: el **pack fantasia** (donde predominan las luces fantasia y de colores), el **pack color** (donde predominan las luces de color y los cohetes), el **pack de cohetes** (con cohetes que hacen diferentes tipos de explosión, muy usados en las fiestas de poblaciones pequeñas), el **pack de truenos** (con mucho éxito en la Comunidad Valenciana) y el **pack básico** (muy solicitado en fiestas organizadas por pequeñas empresas donde se deja que los trabajadores 'quemen' material). También tienen un **surtido variado** con un poco de todo (excepto luces fantasia y, según el presupuesto, luces de color).

Además, los expertos del departamento de ventas nos cuentan que, desde siempre, para realizar sus recomendaciones a los clientes se evalua en cada pedido cada una de las siguientes características:

- **presupuesto**, que se basa en el tipo de cliente y en el número de asistentes para clasificar el presupuesto en *generoso*, *correcto*, *justito*, *rácano*. Por poner algunos ejemplos, un presupuesto de 4000 euros será considerado generoso para un pueblo de 20 habitantes, correcto para una villa de 2000 habitantes, justito para una ciudad de 100000 habitantes y rácano para una metrópoli de 1200000 habitantes.

- **preferencias**, que basándose en el histórico de pedidos indica el tipo de producto que el cliente pide más: *de_todo_un_poco*, *más_básico*, *más_cohetes*, *más_truenos*, *más_color*, *más_fantasia*.

- **peligro_incendio**, que mide el grado de peligro (*extremo*, *importante* o *menor*) de que se produzca un incendio en base a la proximidad de bosques, coches, estaciones de servicio o naves industriales con productos muy inflamables. El grado de peligro se incrementa con la presencia de viento en la zona y grados de humedad bajos, y se decrementa con la presencia en el evento de una dotación de bomberos.

- **peligro_daños_materiales**, que mide el grado de peligro (*extremo*, *importante* o *menor*) de que se produzcan daños a objetos u estructuras en base a la proximidad de edificios, coches ... al lugar donde se usará el material pirotécnico. Este grado de peligro disminuye si se ha contratado a un especialista pirotécnico.

- **peligro_daños_personales**, que mide el grado de peligro (*extremo*, *importante* o *menor*) de que se produzcan daños a personas en base al número de personas que asistirán al evento y a los m^2 de espacio de los que se dispone. Este grado de peligro disminuye si se ha contratado a un especialista pirotécnico.

Es importante aclarar que los niveles de peligro antes mencionados no dependen del producto que finalmente se va a recomendar al cliente. Se trata de niveles de peligro *a priori* computados únicamente a partir de las características del cliente, el evento y el lugar.

A partir de la evalución de estos criterios queremos obtener una recomendación que diga al usuario de la web que tipo de pack es el más adecuado para su evento y la cantidad de productos de cada tipo que habrá finalmente en el pack.

Sabemos que las características abstractas tienen diferentes relaciones con estos productos. Por ejemplo, las preferencias marcan de una forma abstracta el tipo de pack que suele escoger el cliente, los pack color y fantasia suelen estar contraindicados si hay niveles extremos de peligro en alguno de los

criterios, niveles generosos de presupuesto suelen llevar la recomendación hasta el máximo nivel de pack que sea adecuado para las características del evento, ...

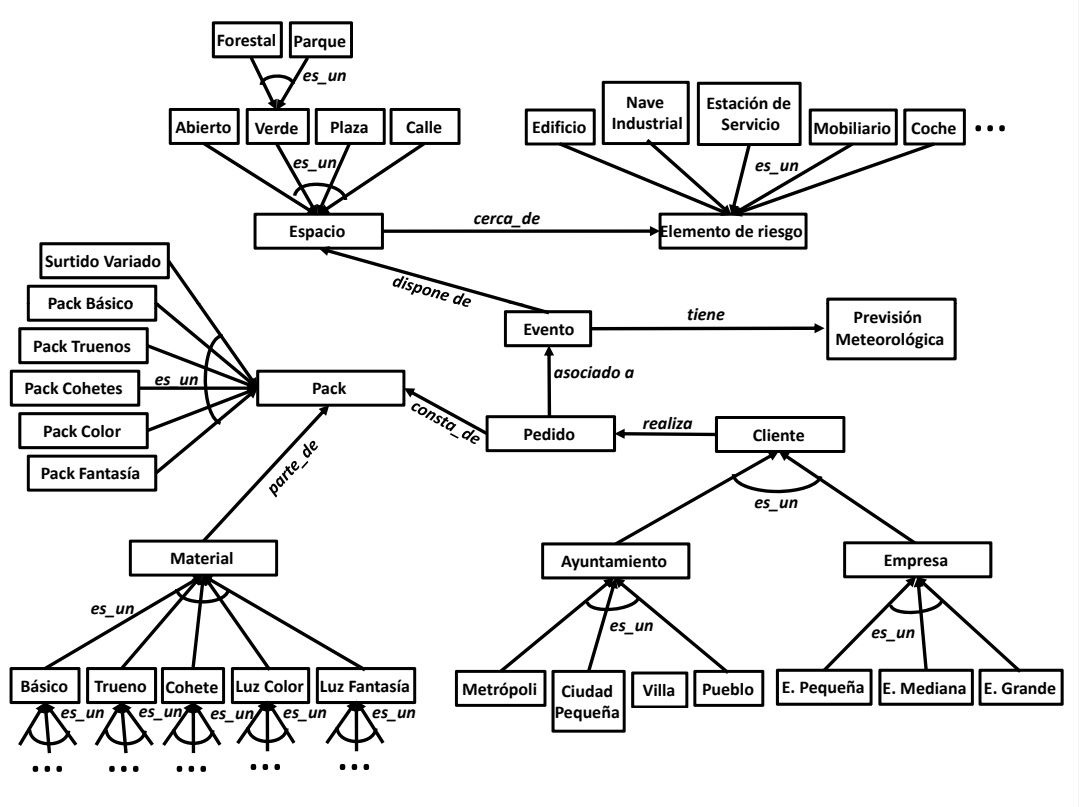
1. Diseña la ontología del dominio descrito, incluyendo todos los conceptos que aparecen en la descripción e identificando los atributos más relevantes. Lista que conceptos forman parte de los datos de entrada del problema y que conceptos forman parte de la solución. (Nota: tened en cuenta que la ontología puede necesitar modificaciones para adaptarla al apartado siguiente).

En este problema la ontología ha de incorporar, como mínimo, todos los conceptos que forman parte de la entrada del problema, así como conceptos que formen parte de la solución.

Los conceptos que forman parte de la entrada del problema son todos referidos a características del evento que el cliente nos da más el historial de pedidos anteriores del cliente al que queremos generarle la recomendación del pack:

- los pedidos anteriores que ha realizado
- el tipo de packs que ha comprado y el material que incluian
- el evento, el espacio en el que se realiza y los elementos de riesgo que pueden recibir daño.

Los conceptos que forman parte de la salida del problema son el pedido final, los packs que contiene y el material que incluyen.



No hay mucho que remarcar sobre la ontología resultante, ya que el enunciado daba bastante detalle. Se ha optado por indicar en el diagrama que de algunos conceptos (como el material `Básico`, `Cohete`, `Trueno`, etc.) seria necesario especificar sus subclases, que en el diagrama solo indicamos con los puntos suspensivos. También hemos decidido no tener un concepto aparte para la recomendación del sistema, ya que hemos considerado que el pedido de material se basa en la recomendación, que finalmente acaba personalizada para el usuario. Pero también sería correcto incluir un concepto `Recomendación` asociado a `Pedido` para distinguir entre lo que se recomendó al cliente y lo que éste acabó comprando. No hemos añadido el concepto `Historial` (de pedidos) a la ontología porque podemos obtener directamente el historial de

pedidos recorriendo todas las relaciones entre `Cliente` y `Pedido`. Tampoco hemos creado un concepto `Preferencias` entre `Cliente` y los conceptos `Pack` o `Material` porque podemos obtener las preferencias analizando el historial de pedidos.

Atributos:

- `Cliente`: nombre (string), *Preferencias* (enumeración: {de_todo_un_poco, más_básico, más_cohetes, más_truenos, más_color, más_fantasía})
- `Ayuntamiento`: número_habitantes (entero)
- `Evento`: número_asistentes (entero), bomberos (booleano), experto_pirotécnico (booleano), *Peligro_Incendio* (enumeración: {extremo, importante, menor}), *Peligro_Daños_Materiales* (enumeración: {extremo, importante, menor}), *Peligro_Daños_Personales* (enumeración: {extremo, importante, menor})
- `Espacio`: m2_libres (real positivo)
- `Verde`: humedad (porcentaje)
- `Elemento_de_Riesgo`: distancia (real positivo)
- `Nave_Industrial`: inflamable (booleano)
- `Previsión Meteorológica`: temp_max (real), temp_min (real), humedad_relativa (porcentaje), vel_viento_km_h (real), probabilidad_lluvia (porcentaje)
- `Material` : unidades (entero), importe (real positivo),
- `Pedido`: presupuesto_max (real positivo), *Presupuesto* (enumeración: {generoso, correcto, justito, rácano})

Como se puede ver, hemos decidido también representar las características abstractas como atributos (son los que tienen el nombre en *cursiva*). De esta manera todos los conceptos que aparecerán en las reglas están soportados por la ontología.

2. El problema descrito es un problema de análisis. Explica cómo lo resolverías usando clasificación heurística, usando los conceptos de la ontología desarrollada en el apartado anterior. Da al menos 3 ejemplos de reglas para cada una de las fases de esta metodología.

En este caso el enunciado ya nos propone un conjunto finito de características abstractas para describir el problema. Para resolverlo mediante clasificación heurística, deberiamos tener además un conjunto enumerable de soluciones candidatas. Por ello plantearemos la solución usando los seis tipos de pack como conjunto enumerable a priori de posibles cmbinaciones de material pirotécnico, dependiendo de los valores discretos de las características abstractas que nos ha dado la empresa.

A partir de este planteamiento lo que nos queda es describir las fases y elementos necesarios para aplicar la clasificación heurística al problema.

El primer elemento son los problemas concretos que hay que tratar. En este dominio los problemas concretos están definidos por los datos concretos del cliente, su historial de pedidos y los datos del evento que son la entrada del problema.

El segundo elemento son los problemas abstractos, que describen al cliente mediante las 5 características abstractas (**Preferencias**, **Presupuesto**, **Peligro_Incendio**, **Peligro_Daños_Materiales** y **Peligro_Daños_Personales**).

Para conectar los problemas concretos con los abstractos necesitamos definir las reglas de abstracción de datos, por ejemplo:

- si presupuesto_max > 1000000 entonces Presupuesto=generoso;
- si (número_habitantes * 200) ≥ presupuesto_max > (número_habitantes * 100) entonces Presupuesto=generoso;
- si existe Elemento_de_riesgo tal que Elemento_de_riesgo.distancia < 150m entonces hay_elementos_en_peligro;
- si existe Nave_Industrial tal que (Nave_Industrial.inflamable y Nave_industrial.distancia < 350m) entonces hay_elementos_en_peligro;
- si existe Estación_de_servicio tal que Estación_de_servicio.distancia < 1000m entonces hay_elementos_en_peligro;
- si velocidad_viento > 60 y (Forestal o hay_elementos_en_peligro) entonces Peligro_Incendio=extremo;
- si especialista y velocidad_viento < 50 y no(hay_elementos_en_peligro) entonces Peligro_Daños_Materiales=menor;

El tercer elemento son las soluciones abstractas. En este caso el enunciado nos dice que la empresa ya ofrece a sus clientes unos packs como base de la recomendación, y lo que haremos es seleccionar el tipo de pack que pueda ser más adecuado para el cliente.

Ejemplos de reglas de asociación heurística que ligan los problemas abstractos con las soluciones abstractas serian las siguientes:

- si Peligro_Incendio=extremo y Peligro_Daños_Materiales=extremo y Peligro_Daños_Personales=extremo entonces Pack_Básico
- si Presupuesto=generoso y Preferencias=más_fantasía y Peligro_Incendio=menor y Peligro_Daños_Materiales=menor y Peligro_Daños_Personales=menor entonces Pack_Fantasía
- si Presupuesto=correcto y Preferencias=más_fantasía y Peligro_Incendio=menor y Peligro_Daños_Materiales=importante y Peligro_Daños_Personales=menor entonces Pack_Color
- si Presupuesto=justito y Preferencias=de_todo_un_poco y (Peligro_Incendio=importante o Peligro_Daños_Materiales=importante o Peligro_Daños_Personales=importante) entonces Pack_Básico

El cuarto elemento son las soluciones concretas. En este caso son soluciones completas en las que se asigna al pack recomendado el número exacto de material pirotécnico de cada tipo, teniendo en cuenta no solo el tipo de pack sino también los datos concretos del cliente y del evento que nos vienen como entrada.

Algunos ejemplos de como refinar la solución son los siguientes:

- en el caso del pack básico poner como mínimo 10 unidades de material básico por persona que asiste, y ajustar el número de unidades al presupuesto máximo.
- en el caso del surtido variado, si no hay experto pirotécnico o bomberos repartir el presupuesto máximo solo en unidades de material básico, cohetes y truenos.
- en el caso del pack color, si se trata de una empresa catalana o de un ayuntamiento catalan colocar un 10 % de unidades de las luces de color "rojas y amarillas".

3. Las características que se utilizan para caracterizar el pedido y el pack recomendado no son independientes entre si. Por ejemplo: el peligro de incendio influye sobre el peligro de daños materiales, y éste sobre el peligro de daños personales; las preferencias influyen directamente sobre la elección de cada tipo de pack, etc.

   Define el problema de asociación heurística como una red bayesiana expresando en ella al menos las relaciones indicadas y alguna más, de forma que todas las características tengan algun tipo de influencia en la solución. Da un ejemplo de tabla de probabilidad de algún nodo, inventándote las probabilidades, pero expresando como influyen los valores de los nodos padre en las probabilidades

de los valores de los nodos hijo.

Esta claro que tenemos dos grupos de nodos, los que corresponden a las características abstractas del evento que se obtienen en la fase de abstracción, y los que corresponden a las características de la solución abstracta. Nuestro objetivo es construir una red que conecte características del problema abstracto a características de la solución abstracta. Los nodos de la solución serán los nodos finales de la red bayesiana (en este caso un único nodo tipo_pack). Los nodos iniciales de la red serán los del problema abstracto (Preferencias, Presupuesto, Peligro_Incendio, Peligro_Daños_Materiales y Peligro_Daños_Personales). En la solución se ha decidido tener una sola variable tipo_pack con seis posibles valores (básico, truenos, cohetes, color, fantasía, surtido) en vez de tener una variable por cada tipo porque de esta forma aprovecharemos al máximo la capacidad de razonamiento aproximado de la red bayesiana: los porcentajes que nos dé la red para cada valor de la variable tipo_pack nos dirán el tipo de pack más adecuado para el evento.

Respecto a las dependencias, representaremos las dependencias entre nodos, ya sean de la solución o del problema abstracto. Es muy importante representar en la red cómo dependen los nodos solución de los nodos del problema abstracto, ya que ese es el objetivo principal de la fase de asociación heurística. Pero también es bueno representar dependencias entre nodos del problema abstracto si la intuición (y el enunciado) nos dicen que son dependientes (por ejemplo, Peligro_Daños_Materiales depende de Peligro_Incendio, Peligro_Daños_Personales depende de Peligro_Daños_Materiales) ya que eso permite calcular en casos en los que faltan datos el valor de una de esas variables respecto a las otras, haciendo el sistema más robusto.

Una tabla de probabilidad simplemente ha de asignar más probabilidad a valores más correlacionados entre grupos de variables. Por ejemplo, si escogemos una variable como Peligro_Daños_Materiales (que depende de Peligro_Incendio), la tabla de probabilidad podría ser algo como lo siguiente:

	Peligro_Daños_Materiales		
Peligro_Incendio	extremo	importante	menor
extremo	0,7	0,2	0,1
importante	0,35	0,4	0,15
menor	0,05	0,15	0,8

La tabla intenta reflejar que la probabilidad de daños materiales es directamente proporcional al peligro de incendio.

III

Planificació

6. Planificació

6.1 Problemas solucionados

6.1.1 Problema 3

Un equipo de investigadores en robótica necesita dotar a su robot de la capacidad de planificar acciones para poder probar su habilidad de reconocimiento de objetos y colores. Para empezar las pruebas quieren hacer un planificador simple que le diga al robot cómo apilar y desapilar vasos en una mesa.

Los vasos pueden ser de dos colores: blanco y negro. En la mesa se han pintado cuatro casillas, dos blancas y dos negras. Las restricciones que se imponen son las siguientes:

- El robot solo puede usar uno de sus brazos, y coger un vaso a la vez.
- Un vaso negro solo se puede poner encima de un vaso blanco o una casilla blanca.
- Un vaso blanco solo se puede poner encima de un vaso negro o una casilla negra.

1. Describir el dominio (incluyendo predicados, acciones, etc...) usando PDDL. Dad una explicación razonada de los elementos que habéis escogido. Tened en cuenta que el modelo del dominio ha de funcionar para tamaños de problema grandes (como el descrito en el apartado c)

> Existen muchas posibles formas de modelar este dominio en PDDL (distinguir o no distinguir entre vasos individuales, distinguir o no distinguir entre vasos y casillas...) y por ello tomaremos decisiones que ya vayan encaminadas en la dirección correcta para el apartado c).
>
> Algunas cosas que tendremos en cuenta en la solución propuesta:
>
> - intentaremos minimizar el número de operadores y el factor de ramificación de los mismos, de forma que se reduzca la exploración de que operadores son aplicables en cada momento;
> - evitaremos operadores con parámetros similares, de forma que la existencia de unos objetos u otros dirija la instanciación de operadores;
> - usaremos tipos en las variables para reducir en lo posible la cantidad de objetos que el planificador comprobará para cada parámetro del operador;

- intentaremos evitar el uso de `exists` y `forall` en las precondiciones y efectos, ya que tienen un impacto negativo en el tiempo de cómputo y, en la práctica, aumentan el factor de ramificación por la existencia de variables ocultas (variables a instanciar que no son parámetros) dentro del operador;
- será más importante que la ejecución sea eficiente, aunque la representación sea más compleja (es decir, añadiremos predicados, operadores y tipos si eso puede facilitar la labor del planificador).

Una solución muy equilibrada sería la siguiente:

```
(define (domain vasos)
  (:requirements :adl :typing)
  (:types vaso casilla - object)

  (:predicates (encima ?x - vaso ?y - object)
               (libre ?x - object)
               (blanco ?x - object)
  )

  (:action mover-vasoA-a-destinoB
   :parameters  (?vasoA - vaso ?bajoA - object ?destinoB - object)
   :precondition (and (libre ?vasoA) (libre ?destinoB) (encima ?vasoA ?bajoA)
                      (not (and (blanco ?vasoA) (blanco ?destinoB)))
                 )
   :effect (and (encima ?vasoA ?destinoB) (libre ?bajoA)
                (not (libre ?destinoB)) (not (encima ?vasoA ?bajoA))
           )
  )
)
```

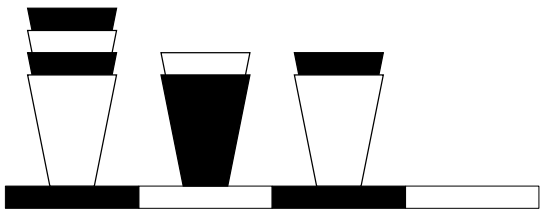
En esta solución se distingue entre vasos y casillas, pero no siempre (para no duplicar predicados y acciones muchas veces se usa el tipo `object`). Se han creado tres predicados:

- `encima`, que nos dice si un vaso esta encima de otro objeto (vaso o casilla). Es importante ver que se ha definido de la forma más general posible (para evitar tener un predicado para vaso encima de vaso y otro para vaso encima de casilla) que aumente la complejidad de las precondiciones y los efectos. Al restringir que el primer parametro de este predicado sea un vaso, automáticamente el planificador no moverá casillas (ni siquiera lo explorará).
- `libre`, que nos dice si un vaso o una casilla esta libre (no tiene nada encima). Aunque pueda parecer que este predicado es redundante, no lo es: la alternativa (usar un `exists` para ver si hay algun vaso encima de un cierto objeto) es mucho más compleja algorítmicamente (lo ha de comprobar para todos los vasos). Añadir este predicado (y actualizarlo adecuadamente en la precondición y los efectos de los operadores) reduce considerablemente el tiempo de cómputo.
- `blanco`, que nos dice si un objeto (vaso o casilla) es de color blanco. Si no es blanco es negro, así que no hace falta tener otro predicado `negro`.

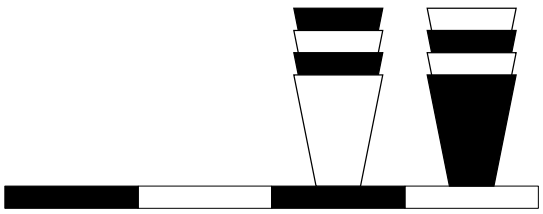
Se ha decidido solucionar el problema con un solo operador (mover) en vez de dos (coger y dejar), ya que no nos piden que modelemos estados intermedios en los que el robot tiene el vaso en el brazo, y de esta forma reducimos la profundidad del espacio de búsqueda. Otra importante decisión ha sido añadir como parámetro de la acción el objeto que está debajo del vaso que queremos mover (`?bajoA`): esto evita usar un `exists` para buscar que objeto está por debajo del que vamos a mover, para así poder dejarlo libre en los efectos. La última cosa remarcable es que el usar un solo predicado para controlar el color (`blanco`) nos permite escribir de forma

sencilla una expresión que controla que un vaso negro solo puede ir encima de un objeto blanco y viceversa, y así hacer un único operador para todos los casos: `(not (and (blanco ?vasoA) (blanco ?destinoB)))`.

2. Describir el problema siguiente usando PDDL. Dad una breve explicación de cómo modeláis el problema.



Estado inicial



Estado final

En este caso el modelado del problema está totalmente marcado por el modelado del dominio del apartado anterior. El resultado es el siguiente:

```
(define (problem vasosproblem1) (:domain vasos)
(:objects
   B1 B2 B3 B4 N1 N2 N3 N4 - vaso
   C1 C2 C3 C4 - casilla
)
(:init
  (blanco B1)(blanco B2)(blanco B3)(blanco B4)
  (blanco C2)(blanco C4)
  (encima B1 C1)(encima N1 B1)(encima B2 N1)(encima N2 B2)(libre N2)
  (encima N3 C2)(encima B3 N3)(libre B3)
  (encima B4 C3)(encima N4 B4)(libre N4)
  (libre C4)
)
(:goal
  (and
    (libre C1)
    (libre C2)
    (encima B2 C3)(encima N2 B2)(encima B1 N2)(encima N1 B1)(libre N1)
    (encima N4 C4)(encima B4 N4)(encima N3 B4)(encima B3 N3)(libre B3)
)
))
```

Se ha creado un objeto por cada vaso y casilla del problema. Para facilitar la escritura se le han dado nombres diferentes a casillas, vasos blancos y negros (`Cx`, `Bx` y `Nx` respectívamente), aunque al planificador le da lo mismo el nombre que le demos a los objetos. Lo importante para facilitar el trabajo del planificador es que hemos etiquetado los objetos con el tipo adecuado (`vaso` ó `casilla`).

El estado inicial y el estado objetivo son simplemente una fotografia de las configuraciones que aparecen en el enunciado usando los predicados. Para facilitar su lectura cada casilla es una linea, aunque al planificador le és indiferente. En el caso del estado inicial también añadimos que elementos son blancos (por omisión el resto son negros).

3. Ahora supongamos que hemos pintado en la mesa un tablero de 24×24 casillas, la mitad blancas y la mitad negras, y que tenemos 600 vasos blancos y 600 vasos negros. ¿Cual seria el factor de ramificación de vuestros operadores? ¿Se os ocurre una forma de mejorar el modelo que habéis hecho? Razonad vuestra respuesta (no hace falta que cambiéis vuestro modelo, sólo que indiquéis que cambios habría que hacer.).

> En nuestro caso tenemos un solo operador con tres parámetros, y dos tipos de objetos (usaremos $NV = 1200$ para el número de vasos y $NC = 24 \times 24$ para el número de casillas). Si no tenemos en cuenta los predicados de la precondición, en teoría el primer parámetro se comparará con todos los vasos (NV), el segundo con todos los objetos ($NC + NV$) y el tercero también con todos los objetos ($NC + NV$), dando una complejidad de $NV * (NC + NV) * (NC + NV) = NV * (NC + NV)^2$. Pero si intentamos acotar mejor la complejidad y analizamos realmente cuantos objetos se van a explorar para los operadores, es mejor fijarse en los hechos que tendremos en el caso medio y en los predicados a comprobar en la precondición de la acción. De forma consciente se pusieron primero los predicados que filtran más rápido los objetos con los que instanciar el operador. Por eso se usa `libre` en los primeros dos predicados de la precondición, lo cual 1) reduce el número de comparaciones para `?vasoA` al mínimo entre el número de casillas (habrá como máximo NC hechos `libre` en todo momento, uno por cada casilla del tablero) y el número de vasos (en el primer parámetro solo miramos vasos); y 2) reduce el número de comparaciones para `?destinoB`) al número de hechos `libre` (que como ya hemos dicho, hay tantos como casillas, NC). El segundo parámetro `?bajoA` solo se comparará con el segundo parámetro de cada hecho `encima`, y de estos hay tantos como vasos (NV). La última condición (que mira que origen y destino sean de diferente color) se hace ya con variables instanciadas, por lo que tiene un coste constante. Poniéndolo todo junto la complejidad es $min(NC, NV) * NV * NC = min(NV * NC^2, NC * NV^2)$.
>
> Hay pocas mejoras que hacer ya que todas se aplicaron durante el modelado. La única alternativa que tiene potencial de ser mejor sería modelar el problema directamente con casillas, el color de las casillas, y cuantos vasos hay encima de cada casilla, pero sin enumerarlos. Pero para tener una solución eficiente habría que usar adecuadamente fluentes y funciones con un planificador como `Metric FF`.

6.1.2 Problema 6

Hace años la burbuja inmobiliaria y de infraestructuras junto a los delirios de grandeza de algunos políticos provocó la creación de múltiples aeropuertos distribuidos por la geografia española. Sin embargo la crisis económica ha obligado ahora al Gobierno a tomar medidas y exigir a estos aeropuertos un plan de viabilidad para evitar su cierre. Por eso los gestores de un pequeño aeropuerto de la costa Mediterránea nos piden participar en un proyecto piloto de automatización de la torre de control del aeropuerto usando técnicas de inteligencia artificial. En concreto nos piden un sistema automático que, cada media hora, recibe la lista de vuelos que están pendientes de aterrizar o despegar del aeropuerto (y en el caso de los pendientes de despegue, la puerta de embarque que ocupan) y ha de generar un plan que diga en que orden han de despegar o aterrizar los aviones (y en el caso de los que aterrizan, en que puerta de embarque han de desembarcar).

El aeropuerto dispone de dos pequeñas terminales (A y B) con cinco puertas de embarque ($A1$, $A2$, $B1$, $B2$ y $B3$). Dispone también de dos pistas para el despegue y aterrizaje de aviones ($ps1$ y $ps2$), pero los fuertes vientos de la zona hacen que a veces no se pueda usar una de ellas. Nunca puede haber dos aviones a la vez en la misma pista (por ejemplo un avión no puede aterrizar si hay otro avión que acaba de aterrizar o un avión a punto de despegar; un avión no puede entrar en la pista para despegar si hay otro a punto de despegar o un avión aterrizando que no ha salido de la pista). Como las pistas están muy cerca de las puertas de embarque y hay poco espacio para maniobrar, por motivos de seguridad no se permite a un avión en tierra desconectarse de una puerta de embarque e ir a una pista si la pista no está libre de aviones, ni a un avión que acaba de aterrizar dirigirse a una puerta de embarque si esta tiene un avión).

Para simplificar el problema y no tener que modelar tiempos de despegue, aterrizaje o desplazamiento entre pistas y puertas de embarque, aprovechando que el aeropuerto es tan pequeño (y que se han de observar unas distancias de seguridad entre aviones) consideraremos que un avión que ha de despegar solo puede estar o bien en la puerta de embarque, o bien en la pista a punto de despegar o bien ya ha despegado. De igual forma un avión que ha de aterrizar en el aeropuerto sólo puede estar o bien pendiente de aterrizar, aterrizado ya en la pista o en la puerta de embarque para desembarcar. No hace falta modelar los estados intermedios y solo se puede pasar de un estado al siguiente si la pista o la puerta de embarque a la que se ha de mover el avión no tiene otro avión ocupándola.

1. Describe el dominio (incluyendo predicados, acciones, etc...) usando PDDL. Da una explicación razonada de los elementos que has escogido. Ten en cuenta que el modelo del dominio ha de poderse extender no sólo a más o menos aviones sinó también a más o menos pistas y a más o menos puertas de embarque.

Existen muchas posibles formas de modelar este dominio en PDDL (con más o menos predicados, con más o menos tipos, con más o menos operadores...) y por ello tomaremos decisiones que vayan encaminadas a rear un modelo que no contenga ineficiencias innecesarias.

Algunas cosas que tendremos en cuenta en la solución propuesta:

- intentaremos minimizar el número de operadores y el factor de ramificación de los mismos, de forma que se reduzca la exploración de que operadores son aplicables en cada momento;
- evitaremos operadores con parámetros similares, de forma que la existencia de unos objetos u otros dirija la instanciación de operadores;
- usaremos tipos en las variables para reducir en lo posible la cantidad de objetos que el planificador comprobará para cada parámetro del operador;
- intentaremos evitar el uso de `exists` y `forall` en las precondiciones y efectos de los operadores, ya que tienen un impacto negativo en el tiempo de cómputo y, en la práctica, aumentan el factor de ramificación por la existencia de variables ocultas (variables a instanciar que no son parámetros) dentro del operador;
- será más importante que la ejecución sea eficiente, aunque la representación sea más compleja (es decir, añadiremos predicados, operadores y tipos si eso puede facilitar la labor del planificador).

Una solución muy equilibrada sería la siguiente:

```
(define (domain aeropuerto1)
  (:requirements :adl :typing)
  (:types avion lugar - object
          pista puerta aire - lugar)
  (:predicates
   (en ?avion - avion ?lugar - lugar)
   (salida ?avion - avion)
   (libre ?l - lugar)
   (servido ?avion - avion)
  )

  (:action ir_a_pista
    :parameters (?av - avion ?pemb - puerta ?pst - pista )
    :precondition (and (en ?av ?pemb) (salida ?av)
                       (libre ?pst)
                  )
    :effect (and (en ?av ?pst) (not (en ?av ?pemb))
```

```
                    (libre ?pemb) (not (libre ?pst))
               )
     )

     (:action despegar
      :parameters (?av - avion ?pst - pista ?ai - aire)
      :precondition (and (en ?av ?pst) (salida ?av) )
      :effect (and (en ?av ?ai) (not (en ?av ?pst))
                    (libre ?pst) (servido ?av)
               )
    )

    (:action aterrizar
      :parameters (?av - avion ?ai - aire ?pst - pista)
      :precondition (and (en ?av ?ai) (not (salida ?av))
                          (libre ?pst)
                     )
      :effect (and (en ?av ?pst) (not(en ?av ?ai))
                    (not (libre ?pst))
               )
    )

    (:action ir_a_puerta_embarque
      :parameters (?av - avion ?pst - pista ?pemb - puerta )
      :precondition (and (en ?av ?pst) (not (salida ?av))
                          (libre ?pemb)
                     )
      :effect (and (en ?av ?pemb) (not (en ?av ?pst))
                    (libre ?pst) (not (libre ?pemb)) (servido ?av)
                     )
     )

)
```

En esta solución se distingue entre aviones, pistas, puertas de embarque y el aire. No se han modelado ni las terminales ni las ciudades de origen o destino de los aviones porque no hacen falta para solucionar el problema que se plantea. Se ha creado un tipo más (lugar) para poder decir que un avión esta o bien en una puerta de embarque o en una pista o en el aire. Las pistas que no estén disponibles por motivos del viento simplemente se consideraran que no estan libres. Se han creado tan solo cuatro predicados:

- `en`, que nos dice en que lugar (puerta de embarque, pista o aire) está el avión.
- `salida`, que si es cierto nos dice si el avión pertenece a la lista de salidas, y si está negado que pertenece a la lista de llegadas. Nos permite saber en que dirección se ha de mover cada avión y en combinación con el predicado anterior basta para saber el siguiente paso que se ha de hacer con ese avión. Desde el punto de vista del planificador se usa para restringir en todos los operadores los aviones que la precondición intenta explorar (por ejemplo, el operador `aterrizar` solo se intentará aplicar a aviones que no son de salida y que estan en el aire.
- `libre`, que nos dice que la puerta de embarque o la pista no esta ocupada por ningun avión. Aunque podríamos usar el predicado `en` para saber si una pista o una puerta de embarque tienen algún avión ocupándolas, es más efectivo añadir el predicado `libre`: si tuvieramos solo el predicado `en`, para saber si una puerta/pista esta libre tendriamos que

comprobar que para todos los aviones no hay ninguno que esté `en` esa puerta/plaza (y esto implica el uso de `forall` o de `(not exists)`).

- `servido`, que nos dice si un avión ya ha alcanzado su objetivo (despegar o aterrizar y desembarcar en una puerta de embarque). El plan acabará cuando todos los aviones estan servidos (que es una condición más fácil de comprobar que mirar, para todo avión, que o bien despegó o bién aterrizó y está en una puerta de embarque. Además, como veremos en el apartado siguiente, facilita la escritura del objetivo del problema ya que para los aviones que aterrizan no hemos de crear una lista de predicados asignándoles de forma estática la puerta de embarque en la que han de desembarcar.

El modelo consta de cuatro operadores: `ir_a_pista`, `despegar`, `aterrizar` e `ir_a_puerta_embarque`, que son autoexplicativos. Las precondiciones se han colocado de forma que minimicen los objetos que el planificador intenta probar (por ejemplo, el operador `ir-a-pista` solo explora, de los pocos aviones que estan en una puerta de embarque, aquellos con el predicado `salida` cierto). Aunque se podría usar el predicado `servido` como filtro en todas las operaciones (no se deben de considerar aviones que ya han sido servidos) en este caso la combinación de los predicados `en` y `salida` nos permite saber en todo momento si ese avión está servido o no y limita de forma eficaz el factor de ramificación real de los operadores a los pocos aviones que cumplen ambas condiciones.

camiones y más o menos ferries, sino también más o menos camiones dentro del ferry (podemos incluso modelar ferries en los que caben más camiones y otros en los que caben menos).

2. El aeropuerto nos ha proporcionado el listado de salidas y llegadas programadas para la próxima media hora. También nos avisa que sólo tenemos en estos momentos la pista *ps*2 operativa (un fuerte viento lateral hace impracticable usar la otra pista en los próximos minutos). Nos pide que el planificador genere un plan de salidas y llegadas para la próxima media hora en la que se asigne un orden entre los despegues y aterrizajes (no ha de seguir el orden en el que aparecen en la tabla) y se asigne una puerta de embarque para los vuelos que aterrizan.

SALIDAS		
vuelo	*destino*	*puerta*
IB0051	Madrid	A1
BA6136	Londres	B1
AF0700	Lyon	B2
AL8860	Milán	B3

LLEGADAS		
vuelo	*procedente de*	*puerta*
IB0121	Málaga	—
VY0256	Barcelona	—
KL0333	Amsterdam	—
LH4044	Berlín	—

Describe este problema usando PDDL. Da una breve explicación de cómo modelas el problema.

En este caso el modelado del problema está totalmente marcado por el modelado del dominio del apartado anterior.

```
(define (problem aeropuertos-2pistas5puertas8aviones)
   (:domain aeropuerto1)
   (:objects IB0121 VY0256 KL0333 LH4044 IB0051 BA6136 AF0700 AL8860 - avion
             A1 A2 B1 B2 B3 - puerta
             ps1 ps2 - pista
             air - aire
    )
```

```
(:init
  (en IB0121 air)
  (en VY0256 air)
  (en KL0333 air)
  (en LH4044 air)
  (en IB0051 A1) (salida IB0051)
  (en BA6136 B1) (salida BA6136)
  (en AF0700 B2) (salida AF0700)
  (en AL8860 B3) (salida AL8860)
  (libre A2)
  (libre ps2)
)


(:goal (forall (?av - avion) (servido ?av)))
)
```

Se ha creado un objeto por cada vuelo, puerta y pista que menciona el enunciado. Los nombres escogidos intentan hacer más facil de leer el modelo, aunque al planificador le da lo mismo el nombre escogido para cada objeto.

El estado inicial esta compuesto por tres bloques de predicados:

- el primero, en el que modelamos los vuelos de llegada; la ausencia de un predicado `salida` para los primeros cuatro vuelos indica que estos vuelos no son salidas, sino llegadas.
- el segundo, en el que especificamos los vuelos de salida, diciendo en que puerta de embarque se encuentran
- el último, que describe la disponibilidad del resto de elementos del aeropuerto. Para modelar el hecho de que la pista `ps1` no esta disponible en este caso hemos optado por omitir el predicado `(libre ps1)` (y con ello no será considerada por ninguno de los operadores). Una alternativa igualmente válida seria directamente eliminar `ps1` de la lista de objetos de tipo `pista` disponibles para el planificador.

El estado objetivo usa el predicado `servido` en vez del predicado `en` para no tener que especificar para las llegadas en que puerta han de desembarcar (queremos que nos lo diga el planificador, no que se lo tengamos que decir).