Examen Final de GRAU-IA

(23 de junio de 2016)
Duración: 2 horas 30 minutos

1. (5 puntos) La compañía de mudanzas *Smart Move* quiere desarrollar una herramienta web capaz de asesorar a sus clientes respecto a cual es la forma más adecuada de trasladar sus pertenencias de su lugar actual al destino de la mudanza. Esta compañía es capaz de realizar diferentes tipos de mudanzas desde diferentes tipos de ubicaciones, ya sean viviendas (tanto pisos como casas unifamiliares), u oficinas (tanto un planta de oficinas como un edificio completo de oficinas).

   Una ubicación se compone de habitáculos, que se pueden clasificar, si son de viviendas, en dormitorio, salón, cocina y baño, o si son de oficinas en oficina, sala de reuniones, habitación de material y habitación multiusos. Respecto a las ubicaciones origen y destino de la mudanza, aparte de la dirección completa, es importante saber si tiene ascensor (incluidas sus dimensiones interiores) o solo hay un acceso de escaleras (de las que queremos saber el ancho y alto del lugar más estrecho). También son importantes los accesos al lugar, por ejemplo saber si hay una area de estacionamiento dentro del edificio, o si podemos aparcar cerca al estar junto a una calle con poco trafico, o si no se puede cargar delante del edificio sin cortar el tráfico.

   Respecto a las cosas a trasladar, se puede estimar el volumen (en litros) a partir del número de habitáculos que tiene el lugar y las características y número de los objetos que contienen (muebles desmontables, muebles no desmontables, objetos de pequeño tamaño, objetos voluminosos, objetos frágiles). Los muebles no desmontables y los objetos voluminosos no se pueden empaquetar y se debe indicar su peso y su dimensión más larga (en cm). Los muebles desmontables y los objetos pequeños se pueden empaquetar y se conoce el volumen aproximado que ocupan empaquetados. Para los objetos frágiles se indica si se pueden empaquetar o no (indicando el volumen en el caso de que sean empaquetables) y si hace falta personal especializado para trasladarlos.

   La empresa dispone de diferentes medios de mudanza (furgonetas, camiones grandes, contenedores de tren y grúas) y de cada uno sabemos el peso máximo de la carga (en kg). De las furgonetas, camiones grandes y trenes sabemos además el precio por kilómetro y el volumen de carga (en litros), mientras que de las gruas sabemos el precio de montaje de la grua y el precio por hora. La empresa también dispone de personal para realizar la mudanza (empaquetadores, cargadores, desmontadores, personal especializado, conductor y operador de grua), y de cada uno se tiene el NIF, el nombre y apellidos, un teléfono de contacto y el precio por hora. Una mudanza se compone de un conjunto de medios de mudanza y del personal necesario.

   Para poder obtener la recomendación de la mudanza el usuario se ha de registrar en la web aportando su NIF, nombre y apellidos, teléfono de contacto, email y escogiendo un nombre de usuario (por razones de seguridad el password no lo guardamos en la ontología). A la hora de solicitar una mudanza el usuario introduce los datos de la ubicación de origen y la de destino (y, apartir de ellos, se calcula la distancia al lugar de la mudanza), cual es el tiempo máximo de carga y descarga de los objetos a trasladar (en horas) y el precio máximo que quiere pagar por la mudanza.

   A partir de esta información y de las características de los objetos que hay que trasladar, el tipo de ubicación y las características de los accesos se quiere obtener una solución que indique si la mudanza debe hacerse mediante camiones grandes y/o furgonetas y cuántas hacen falta de cada tipo, si se ha de utilizar también el tren, si se ha de montar alguna grua, el tipo de personal que es necesario y el número aproximado de personas de cada tipo. También dará una estimación del coste total de la mudanza.

   El experto en logística de la empresa nos ha contado que ellos usan una serie de caracteristicas durante el proceso de diseño de una mudanza. Hay características que son sencillas de obtener, como por ejemplo:

   - el **tipo de mudanza**: *local* (menos de 10 Km), *regional* (menos de 100 Km) o *larga distancia*;
   - el **presupuesto**: *bajo* (menos de 1000 euros), *medio* (menos de 3000 euros) o *alto*;
   - la **duración de la mudanza**: *corta* (menos de 5 horas), *media* (menos de 10 horas), *larga*.

   Otras requieren cierto razonamiento, como

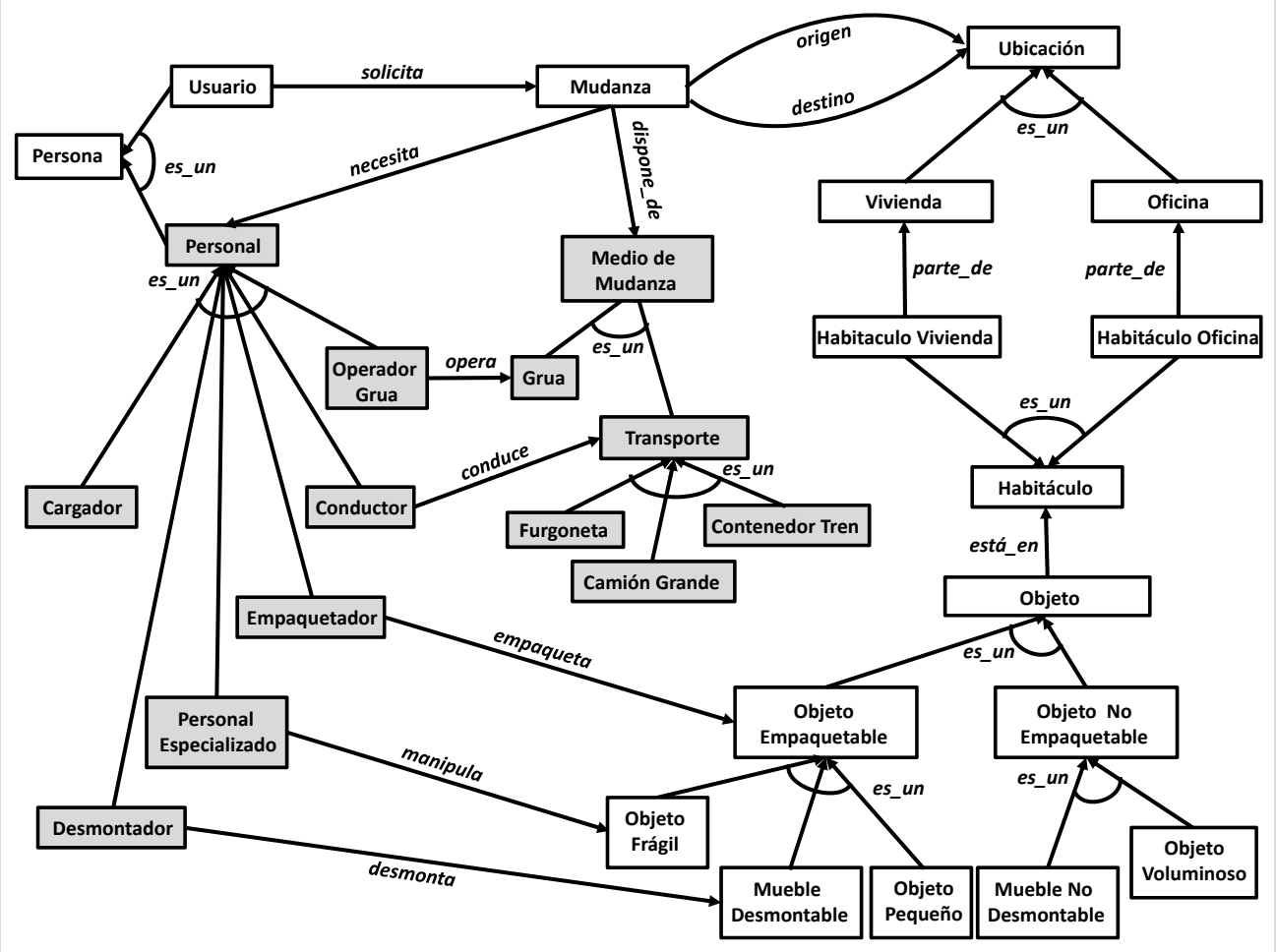
   - la **complejidad de la mudanza**: *sencilla* en la que la mayor parte de los objetos se pueden empaquetar, no hay que desmontar muchos muebles y no hay objetos voluminosos y frágiles, *normal* en la que hay que hacer bastante desmontaje, hay algunos objetos voluminosos pero no se necesitan grúas y hay algunos objetos frágiles y *difícil* en la que se necesitan grúas y hay bastantes objetos frágiles;
   - el **volumen de la mudanza**: *pequeño* si el número de habitáculos es inferior a 5 y no hay objetos voluminosos o muebles no desmontables, *medio* si es una casa unifamiliar o una planta de oficinas y hay pocos objetos no desmontables, *grande* si es una planta de oficinas y hay bastantes objetos no desmontables, *extremo* si es un edificio de oficinas o hay una gran cantidad de objetos no desmontables;
   - la **accesibilidad de la mudanza**: *accesible* si se pueden ubicar los medios de mudanza cerca de la ubicación y hay ascensor, *medianamente accesible* si hay ascensor, pero no caben todos los objetos voluminosos, *poco accesible* si no se pueden ubicar los medios de mudanza cerca de la ubicación y el ascensor no se puede utilizar para la mayor parte de los objetos voluminosos.

   A partir de estas características deciden qué tipo de medios de mudanza necesitamos y los tipos de personal, por ejemplo para una mudanza de volumen grande harán falta camiones, si la mudanza es de larga distancia y de volumen grande hará falta usar el tren, si la ubicación es poco accesible es mejor usar furgonetas, ... Si la complejidad de la mudanza es difícil y es de volumen grande harán falta empaquetadores, cargadores y desmontadores, si el volumen es pequeño con cargadores podría haber suficiente, ...

El número específico de medios de mudanza y personal de cada tipo se puede obtener también razonando a partir de las características definidas, por ejemplo a mayor volumen más personal será necesario, a más objetos frágiles o voluminosos más cargadores y personal especializado hará falta, si la duración de la mudanza ha de ser corta se habrá de incrementar el personal

(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 y los de la solución (los conceptos marcados en color gris en el diagrama).



De la ontología resultante 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. Es así como se ha decidido crear un concepto `Persona` con los atributos comunes de los conceptos `Usuario` y `Personal`, o el concepto `Transporte` con los atributos y relaciones comunes de los conceptos `Furgoneta`, `Camión Grande` y `Contenedor Tren`. Esto ha tenido un impacto importante en la taxonomía de objetos, ya que en vez de clasificarlos entre muebles y objetos, se han clasificado primero según si son empaquetables o no, ya que comparten atributos y, además, en el caso de los empaquetables tienen una relación con el concepto `Empaquetador`. La otra cosa que cabe remarcar es el criterio 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 y/o relaciones diferentes. Ese es el caso tanto de las subclases de `Personal` (varias de ellas tienen relaciones diferentes a sus hermanas) como las de `Objeto Empaquetable` (donde hay algun atributo diferente y también relaciones diferenciadas). Siguendo ese criterio se decidió no crear una taxonomía de subclases para los conceptos `Vivienda`, `Oficina`, `Habitáculo Vivienda` y `Habitáculo Oficina`, que tienen un atributo enumerado para modelar los subtipos (o el número de plantas en el caso de `Oficina`). No se siguió del todo ese criterio en el caso de las subclases de transporte, que a pesar de no tener ni atributos ni relaciones diferenciadas se han decidido poner como subclases por la relevancia que tienen estos tres conceptos en el apartado b y c del problema. Tampoco se siguió del todo ese criterio en las subclases de `Objeto No Empaquetable`, que se podrían substituir por un atributo en la superclase, pero se han añadido siguiendo el heurístico del diseño de ontologías que recomienda que ramas hermanas de la taxonomía tengan un nivel de granularidad similar. Se podría haber añadido algo de generalidad a la ontología añadiendo el concepto `Paquete` (el resultado del proceso realizado

por el `Empaquetador`) y haber definido que el `Cargador` mueve paquetes u objetos no empaquetables, pero como estos detalles no se mencionan en el enunciado y no son realmente relevantes para los siguientes apartados, no se han modelado para no añadir mayor complejidad al modelo.

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.

- `Persona`: NIF (string), nombre (string), apellidos (string), teléfono (string);
- `Usuario`: e-mail (string), username (string);
- `Mudanza`: max_precio (euros), maxt_carga (horas), maxt_descarga (horas)
  distancia (km), *volumen_total* (litros)
  Tipo_mudanza (enumeración: {local, regional, larga_distancia}), *Presupuesto* (enumeración: {bajo, medio, alto}), *Duración* (enumeración: {corta, media, larga}), *Complejidad* (enumeración: {sencilla, normal, difícil}), *Volumen* (enumeración: {pequeño, mediano, grande, extremo}), *Accesibilidad* (enumeración: {accesible, medianamente_accesible, poco_accesible});
- `Personal`: precio_hora (euros), *REC* (enumeración: {ninguno, hasta_5, hasta_10, más_de_10});
- `Medio_de_Mudanza`: peso_max (kg), *REC* (enumeración: {ninguna_o_una, dos_o_tres, más_de_tres});
- `Grua`: precio_hora (euros), precio_montaje (euros);
- `Transporte`: precio_km (euros), volumen_carga (litros);
- `Ubicación`: dirección (string), ascensor_largo (cm), ascensor_ancho (cm), ascensor_alto (cm), escalera_min_ancho (cm), escalera_min_alto (cm), estacionamiento (enumeración: {en_edificio, delante, lejos, ninguno});
- `Oficina`: num_plantas (entero)
- `Vivienda`: tipo (enumeración: {piso, casa})
- `Habitáculo_Oficina`: tipo (enumeración: {oficina, sala_reuniones, hab_material, hab_multiusos})
- `Habitáculo_Vivienda`: tipo (enumeración: {dormitorio, salón, cocina, baño})
- `Objeto_Empaquetable`: volumen (litros);
- `Objeto_No_Empaquetable`: peso (kg), dimensión_max (cm);
- `Objeto_Frágil`: empaquetable? (booleano), necesita_especialista? (booleano);

Como se puede ver, hemos decidido también representar la información derivada/calculada de los datos y las características abstractas como atributos (son los que tienen el nombre en *cursiva*, *Cursiva* o *CURSIVA*). De esta manera todos los conceptos que aparecerán en las reglas están soportados por la ontología. Las características en *cursiva* y minúsculas corresponden a los problemas concretos, las características en *Cursiva* y primera mayuscula a los problemas abstractos, y las características en *CURSIVA* todo en mayúsculas corresponden a las soluciones abstractas.

(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 3 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. Para empezar la solución que pide el enunciado solo puede ser una solución concreta, ya que es imposible poder calcular el precio final de la mudanza en el nivel abstracto sin acceder a los datos concretos de los costes de los medios de mudanza y personal. El segundo hecho importante es que tanto las características sencillas (tipo de mudanza, presupuesto, duración de la mudanza), como las que requieren razonamiento (complejidad, volumen y accesibilidad de la mudanza), pertenecen al problema abstracto. El propio enunciado nos da una pista de ello al darnos uno de los ejemplos de como se usan las características: "*si la mudanza es de larga distancia y de volumen grande hará falta usar el tren*". Este es un ejemplo de regla de asociación heurística que permite decidir sobre la necesidad de añadir contenedores de tren a la solución. Si colocamos las características sencillas (como el tipo de mudanza) como parte de la descripción del problema concreto, no podemos expresar entonces esta regla que aparece en el enunciado.

Una vez sabemos donde colocar algunos elementos del problema 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 derivada de la solicitud de mudanza que realiza el usuario (todos los datos asociados a los conceptos que aprecen sin sombrear en la imagen de la ontología del apartado anterior).

Existen unos pocos atributos concretos que el cliente no introduce y que requieren una fase previa de preprocesado de los datos de entrada (como la distancia entre la ubicacion origen y destino, o el cálculo del volumen total de la mudanza). En este caso estos datos derivados se obtendrían de forma más natural con prodecimientos que realizan cálculos directamente en la ontología o en un mapa on-line, y por ello no presentaremos ejemplos de reglas de preprocesado.

El segundo elemento son los problemas abstractos, estos estarán definidos a partir de las seis características que menciona el enunciado (tipo de mudanza, presupuesto, duración, complejidad, volumen y accesibilidad), y los valores que se asignen a cada característica. Para conectar los problemas concretos con los abstractos necesitamos definir las reglas de abstracción de datos, por ejemplo:

- si Mudanza.max_precio>3000 entonces Presupuesto=alto;
- si Mudanza.distancia<10km entonces Tipo_mudanza=local;
- si Mudanza.maxt_carga<5h y Mudanza.maxt_descarga<5h entonces Mudanza.Duración=corta;
- si cardinalidad(Habitáculo.parte_de(Mudanza.origen))<5 y cardinalidad(Habitáculo.parte_de(Mudanza.destino))<5 y cardinalidad(Objeto_Voluminoso)==0 y cardinalidad(Mueble_No_Desmontable)==0 entonces Mudanza.Volumen=pequeño;
- si Mudanza.origen.estacionamiento==en_edificio y Mudanza.destino.estacionamiento==en_edificio y Mudanza.origen.ascensor_largo>1.5m y Mudanza.origen.ascensor_ancho>1.3m y Mudanza.origen.ascensor_alto>2m y Mudanza.destino.ascensor_largo>1.5m y Mudanza.destino.ascensor_ancho>1.3m y Mudanza.destino.ascensor_alto>2m entonces Mudanza.Accesibilidad=accesible;

El tercer elemento son las soluciones abstractas. En este caso se indica la necesidad de incluir diferentes tipos de medios de mudanza y de personal en la solución, pero no la cantidad exacta de cada uno. Para discretizar los rangos de cantidades de los elementos de la solución abstracta hemos usado los mismo rangos que el enunciado sugiere para el apartado c).

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

- si Mudanza.Accesibilidad==poco_accesible y Mudanza.Volumen==pequeño entonces Furgoneta.REC=dos_o_tres y Camión_Grande.REC=ninguno_o_uno;
- si Mudanza.Volumen==grande y Mudanza.Tipo_mudanza≠larga_distancia entonces Camión_Grande.REC=más_de_tres y Furgoneta.REC=ninguno_o_uno y Cargador.REC=hasta_10 y Empacador.REC=hasta_10 y Desmontador.REC=hasta_5 y Conductor.REC=hasta_5 y Operador_grua.REC=ninguno;
- si Mudanza.Complejidad==dificil y Mudanza.Presupuesto==alto entonces Grua.REC=dos_o_tres y Camión_Grande.REC=dos_o_tres y Cargador.REC=más_de_10 y Empacador.REC=más_de_10 y Desmontador.REC=hasta_10 y Conductor.REC=hasta_5 y Operador_grua.REC=hasta_5;

El cuarto y último elemento son las soluciones concretas. En este caso corresponde al cálculo del número exacto de medios y personal que hace falta, echando mano de la solución abstracta y de los datos concretos del problema, y del precio total de la mudanza. Lo que sigue son algunos ejemplos de cálculos (suponemos la existencia de algunas acciones para las reglas de producción: una que nos permite convocar a Personal diciendo el tipo de personal y el número de individuos, otra para solicitar medios de mudanza, diciendo el medio y cuantos necesitamos, otra para calcular el coste de los medios solicitados y otra para calcular el coste del personal convocado).

- si Cargador.REC==más_de_10 y Mudanza.distancia>100km entonces llamar_a(Cargador, Mudanza.volumen/20);
- si Camión_Grande.REC==dos_o_tres y Furgoneta.REC==ninguno_o_uno y Mudanza.volumen_total≤2*Camión_Grande.volumen_carga entonces solicitar(Camión_Grande, 2) y llamar_a(Conductor,2);
- PrecioTotal = CalcularPrecioGrua() + CalcularPrecioTransporte(Mudanza.distancia) + CalcularPrecioPersonal();

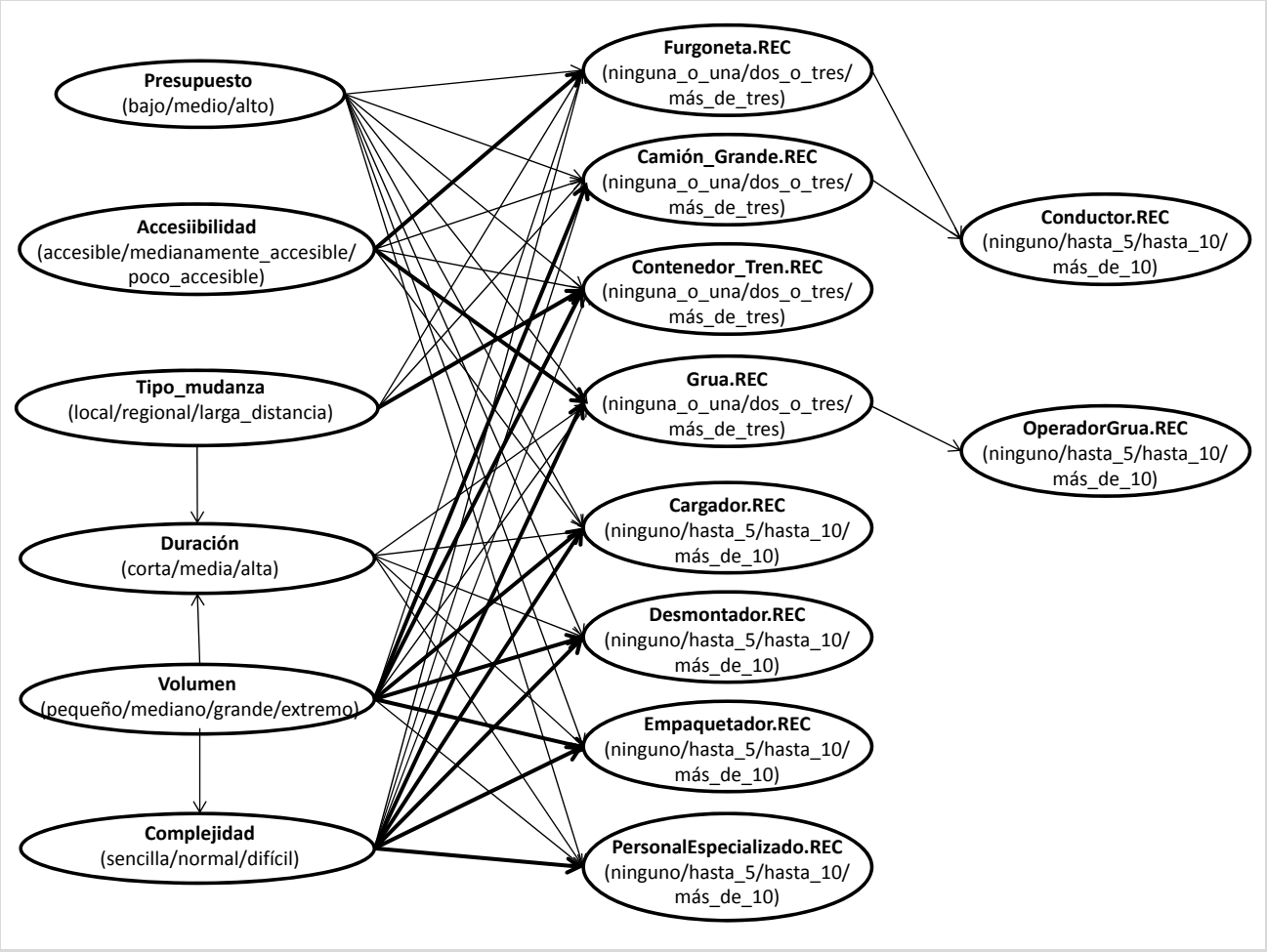
(c) La parte de asociación heurística se podría resolver mediante el formalismo de redes bayesianas. Las características de una solución se podrían definir de manera que se obtuvieran una serie de valores a partir de los cuales se pudiera hacer mejor la especialización. Por ejemplo se podría definir la necesidad de camiones grandes o furgonetas en tres valores *ninguno o uno*, *dos o tres* y *más de tres*, y las necesidades de los diferentes tipos de personal (empaquetadores, cargadores, desmontadores, personal especializado...) en *ninguno*, *hasta 5 personas*, *hasta 10 personas* y *más de 10 personas*, y así con el resto de variables de la solución abstracta. Define el problema de asociación heurística como una red bayesiana expresando en ella al menos las relaciones indicadas en el 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 de la mudanza (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. Dependiendo de si se ha optado por una solución en el apartado b) de tres fases o de cuatro fases, los nodos serán unos u otros.

La figura a continuación corresponde a la solución en tres fases propuesta. Los nodos de la solución abstracta

son las dos columnas de la derecha y corresponden a las variables aleatorias discretas que se sugieren en el enunciado de este apartado. Los nodos a la izquierda corresponden a las 6 características abstractas que modelan la mudanza.

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. Las flechas marcadas en negrita corresponden a las dependencias mencionadas en el enunciado, mientras que el resto de flechas se han añadido usando el conocimiento del *sentido común*: `Tipo_mudanza` influye solo en las variables solución sobre medios de mundanza (cuanto más local, mayor uso de furgonetas y menor de trenes); `Accesibilidad` influye en las variables solución sobre medios de mundanza (a menor accesibilidad, transportes más pequeños y mayor probabilidad de uso de la grua) y en el uso de cargadores; el resto de variables del problema abstracto influyen directamente sobre todas las variables de la columna central. La cantidad de conductores y operadores de grua depende únicamente de los medios de mudanza mobilizados.



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 `Conductor.REC` (que depende de `Furgoneta.REC` y `CamiónGrande.REC`, la tabla de probabilidad podría ser algo como lo siguiente:

		Conductor.REC			
Furgoneta.REC	Camión_Grande.REC	ninguno	hasta_5	hasta_10	más_de_10
ninguna_o_una	ninguna_o_una	0,9	0,1	0,0	0,0
dos_o_tres	ninguna_o_una	0,2	0,8	0,0	0,0
⋮	⋮	⋮	⋮	⋮	⋮
dos_o_tres	dos_o_tres	0,0	0,6	0,4	0,0
⋮	⋮	⋮	⋮	⋮	⋮
más_de_tres	más_de_tres	0,0	0,0	0,3	0,7

La tabla intenta reflejar que la cantidad de conductores es directamente proporcional a la cantidad de vehículos mobilizados. Es importante asegurarse que la suma de los valores de cada fila de la tabla sumen 1.

2. (5 puntos) Una de las cosas que crea más quebraderos de cabeza en los preparativos de una boda es el colocar correctamente a los invitados en las diferentes mesas del banquete de forma que todo el mundo esté contento. Por ello un grupo de emprendedores quiere crear una app que ayude a las parejas a planificar la colocación de sus invitados en el banquete.

   Para cada una de las mesas que hay en la sala del banquete, la app recibe el número máximo de sillas que se pueden colocar en la mesa. También recibe un listado de los invitados al banquete. Para evitar problemas, la app permite introducir dos tipos de restricciones:

   - **La persona A necesita estar al lado de la persona B**: esta restricción indica que la persona B debe estar sentada o bien en la silla inmediatamente a la izquierda de A o en la silla inmediatamente a la derecha de A, y está pensada para que se pueda indicar que, por ejemplo, un niño muy pequeño tiene que estar en la silla al lado su madre, que una persona mayor tiene que estar al lado de alguien que se hará cargo de ella, o que una pareja de hermanos/as, amigos/as o casados/as no quieren estar separados durante el banquete;
   - **La persona A no soporta a la persona B**: a la app no le importa saber la naturaleza o los motivos de esa enemistad, pero lo que hará es evitar que A y B estén en la misma mesa.

   En ambos casos, si la relación de necesidad o de rechazo es en ambos sentidos, la app solo guarda uno de los sentidos para ahorrar memoria. Por ahora otros criterios de colocación de invitados en el banquete, como intentar intercalar a las personas por género dentro de una mesa, o la cercania de las mesas de invitados entre sí o a la mesa de los novios NO serán soportados por la app en su primera versión.

   Este grupo de emprendedores nos pide que desarrollemos un planificador simple que genere una asignación de invitados a mesas de forma que se cumplan las dos restricciones descritas, o que nos diga que no existe asignación posible.

   (a) 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 a más o menos mesas, a mesas con diferente número máximo de sillas y a más o menos invitados.

   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 crear 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 banquete)
  (:requirements :adl :typing)

  (:types comensal silla mesa - object)

  (:predicates
    (necesita ?x - comensal ?y - comensal)
    (nosoporta ?x - comensal ?y - comensal)
    (pertenece_a ?sl - silla ?ms - mesa)
    (izquierda_de ?sizq - silla ?sder -silla)
    (colocado_en ?c - comensal ?sl - silla)
    (colocado_mesa ?c - comensal ?m - mesa)
    (colocado ?c - comensal)
    (asignada ?sl - silla)
  )

  (:action colocar_comensal
    :parameters (?c - comensal ?sl - silla ?ms - mesa ?sizq - silla ?sder - silla)
    :precondition (and (not(colocado ?c)) (not(asignada ?sl)) (pertenece_a ?sl ?ms)
                       (izquierda_de ?sizq ?sl) (izquierda_de ?sl ?sder)
                       (forall (?c2 - comensal)
                           (and
                               (imply (necesita ?c ?c2)
```

```
                                        (or (colocado_en ?c2 ?sizq) (colocado_en ?c2 ?sder))

                                    )
                                    (imply (nosoporta ?c ?c2)
                                        (and (colocado ?c2) (not (colocado_mesa ?c2 ?ms)))
                                    )
                                )
                            )
                        )
    :effect (and (colocado ?c) (colocado_en ?c ?sl) (colocado_mesa ?c ?ms) (asignada ?sl))
   )
)
```

En esta solución se distingue entre tres tipos de objetos: los comensales, las mesas y las sillas. Modelamos la capacidad de cada mesa modelando las sillas y a que mesa pertenecen. Se han creado varios predicados:

- `necesita`, que nos sirve para modelar la restricción de que una persona necesita tener a otra a su lado.
- `nosoporta`, que nos sirve para modelar la restricción de que una persona no puede tener a otra en la misma mesa.
- `pertenece_a`, que nos dice que una silla pertenece a una cierta mesa.
- `izquierda_de`, que nos dice que una silla está a la izquierda de otra. Como toda silla tiene otra a su izquierda, con este predicado es suficiente para describir la posición de las sillas en la mesa. Tampoco nos hace falta añadir un predicado `(derecha_de ?sder - silla ?sizq - silla)`, ya que se puede expresar fàcilmente con `(izquierda_de ?sizq - silla ?sder - silla)`.
- `colocado_en`, que es el predicado que permite modelar la solución obtenida indicando que un comensal ya ha sido colocado en una cierta silla.
- `colocado`, que nos dice si un comensal ya ha sido colocado en alguna silla. Este predicado se añade como filtro en la acción `colocar_comensal`, haciendo que el planificador solo unifique la variable `?c` con comensales que aun no han sido colocados. También se usa en el fichero de problema (ver apartado siguiente) para expresar la condición objetivo: el plan acabará cuando todos los comensales estan sentados (que es una condición más fácil de comprobar que mirar, para todo comensal, si existe una silla tal que el comensal está colocado en la silla).
- `colocado_mesa`, es un predicado que en principio puede parecer redundante pero no lo es. De no tener este predicado en el caso de tener una restricción de `nosoporta`, necesitariamos un `exists` para añadir una variable `sl2` que seria la silla en la que está colocado el comensal non-grato, y mirariamos entonces que la silla no pertenezca a la mesa en la que queremos colocar al comensal.
- `asignada`, que nos dice que una silla ya tiene a un comensal colocado encima de ella. Podríamos usar el predicado `colocado_en` para saber si una silla tiene alguna persona sentada, pero de esa forma para saber si una silla esta libre o no tendriamos que comprobar que, para todos los comensales en el modelo del problema no hay ninguno que esté `colocado_en` esa silla (y esto implica el uso de `forall` o de `(not exists)`). Por ello es más efectivo añadir el predicado `asignada`, que sabiendo la silla nos dice si está asignada o no, y podemos usarlo como filtro en la precondición de la acción.

El modelo tiene un único operador: `colocar_comensal`. Se ha optado por un único operador que lo controle todo, en vez de diferentes operadores que coloquen personas según si tienen o no diferentes restricciones. Esta segunda opción no es menos compleja, al contrario, es más fácil que se nos escape la colocación de una persona que tiene restriccion de necesidad y de no soportar si entra en el operador inadecuado. Las dos primeras condiciones de la precondición filtran comensales ya colocados y sillas ya asignadas, la tercera sirve para saber la mesa a la que pertenece la silla `sl` (algo que necesitaremos para controlar las restricciones `nosoporta`) y las dos últimas condiciones nos permiten saber las sillas que están al lado (a izquierda y a derecha) de la silla `sl` (algo que necesitaremos para controlar las restricciones `necesita`). A continuación se coloca un `forall` para gestionar las posibles restricciones del comensal `?c` que queremos colocar con otro comensal `?c2` (como un comensal puede tener cero, una o más de una de cada una de las restricciones este cuantificador no lo podemos evitar añadiendo parámetros a la acción). En el caso de que `?c` no soporte a `?c2`, comprobamos que `?c2` no esté en la misma mesa que `?c` (gracias al predicado `colocado_mesa`, que como ya se ha explicado evita la necesidad de un `exists`). En el caso de que `?c` necesite a `?c2` también se ha podido evitar el uso de `exists` para saber donde se sienta `?c2`, ya que solo hay dos opciones posibles: la silla a la izquierda de `sl` y la silla a la derecha de `sl`, que obtenemos con la unificación de dos parámetros extra (`sizq` y `sder`) en la cuarta y quinta condición de la precondición. Podemos colocar estos dos parametros extra, ya que para toda silla siempre existe una silla a su izquierda y una silla a su derecha, independientemente de si el usuario que queremos sentar tiene restricciones o no. Si se cumplen todas las condiciones de la precondición, el efecto es que el comensal `?c` será colocado en la silla `?sl`, marcando también que `?c` ha sido colocado y que `?sl` ha sido asignada, para filtrar así futuras unificaciones de esos objetos.

Es importante remarcar que este modelo no solo permite modelar más o menos comensales y más o menos mesas, sino también más o menos sillas en cada mesa (podemos incluso modelar mesas en las que caben más sillas y otras en las que caben menos).

(b) Para probar nuestro planificador el jefe del proyecto nos ha proporcionado los datos de una boda reciente, en concreto dos mesas de amigos de los novios. Una mesa puede tener como máximo 7 sillas y la otra máximo 6

sillas. En ellas se ha de colocar a 12 amigos de los novios:

- *Antonio* y *Ainhoa* son recien casados y, por lo tanto, inseparables,
- *Lola* y *Lolo* son mellizos y van juntos a todas partes,
- *Esperanza* pidió a los novios que la colocaran al lado de *Enrique*,
- *Susana* y *Manu* son pareja y tienen dos hijos, *Manolito* y *Susanita*, que no comen bien, y por ello necesitan estar los dos padres en la misma mesa de los niños para tenerlos controlados,
- *Jesus* y *Quico* vienen solos al banquete y no necesitan de nadie.

Desafortunadamente hay desavinencias entre los amigos: *Ainhoa* no soporta a *Susana*, y *Quico* terminó de forma unilateral su relación con *Esperanza* hace unas semanas.

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

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

```
(define (problem banquete-2mesas-7sillas-6sillas-12comensales)
   (:domain banquete)
   (:objects Antonio Ainhoa Lola Lolo Esperanza Enrique Susana Manu
             Manolito Susanita Jesus Quico - comensal
             s1_1 s1_2 s1_3 s1_4 s1_5 s1_6 s1_7 s2_1 s2_2 s2_3 s2_4 s2_5 s2_6 - silla
             m1 m2 - mesa
    )

(:init
  (necesita Antonio Ainhoa) (necesita Lola Lolo) (necesita Esperanza Enrique)
  (necesita Susana Manu) (necesita Manolito Manu) (necesita Susanita Susana)
  (nosoporta Ainhoa Susana) (nosoporta Quico Esperanza)
  (pertenece_a s1_1 m1)(pertenece_a s1_2 m1)(pertenece_a s1_3 m1)(pertenece_a s1_4 m1)
  (pertenece_a s1_5 m1)(pertenece_a s1_6 m1)(pertenece_a s1_7 m1)
  (izquierda_de s1_1 s1_2)(izquierda_de s1_2 s1_3)(izquierda_de s1_3 s1_4)
  (izquierda_de s1_4 s1_5)(izquierda_de s1_5 s1_6)(izquierda_de s1_6 s1_7)
  (izquierda_de s1_7 s1_1)
  (pertenece_a s2_1 m2) (pertenece_a s2_2 m2) (pertenece_a s2_3 m2) (pertenece_a s2_4 m2)
  (pertenece_a s2_5 m2) (pertenece_a s2_6 m2)
  (izquierda_de s2_1 s2_2)(izquierda_de s2_2 s2_3)(izquierda_de s2_3 s2_4)
  (izquierda_de s2_4 s2_5)(izquierda_de s2_5 s2_6)(izquierda_de s2_6 s2_1)
)

(:goal (forall (?c2 - comensal) (colocado ?c2)))
)
```

Se ha creado un objeto por cada comensal y cada mesa. También hemos creado un objeto por cada una de las sillas (que luego asociamos a las mesas a las que pertenecen). Los nombres intentan hacerlo más facil de leer, pero al planificador le da lo mismo el nombre escogido para cada objeto.
El estado inicial indica las restricciones de necesidad/rechazo y luego describe la colocación de las sillas en mesas. En el caso de las relaciones `necesita`, como el enunciado nos dice que incluso en el caso de necesidad en los dos sentidos el planificador solo considera uno de los sentidos, para todas las parejas que se mencionan en el enunciado hemos usado un único predicado `necesita`. En el caso de la familia, para asegurar que se les pone a todos juntos en la mesa, se ha optado por hacer que cada hijo necesite a uno de los padres y que uno de los padres necesite al otro.
El estado objetivo usa el predicado `colocado` (el plan acaba cuando todos los comensales han sido colocados en mesas). Se podria haber descrito listando los doce predicados `colocado` (uno por comensal), pero en este caso se ha optado por la expresión `forall` en adl, ya que nos permite cambiar el número de comensales del problema sin tener que cambiar la descripción del estado objetivo, y no nos añade complejidad (en Fast Forward el `forall` en los objetivos se instancia una sola vez generando los predicados individuales, con un coste lineal respecto al número de objetos a explorar).

Las notas se publicaran el día **29 de junio**.