Examen Final de GRAU-IA

(10 de enero de 2017)

Duración: 2 horas 30 minutos

1. (5 puntos) Dado el preocupante aumento de los niveles de contaminación en las grandes ciudades europeas se quiere incentivar la compra de vehículos híbridos y eléctricos. Para ello nos piden diseñar un programa que, a partir del tipo de conductor, le recomiende uno o varios vehículos de emisiones reducidas del mercado. En esta primera fase el programa solo recomendará vehículos para particulares y autónomos, dejando los coches de empresa y las flotas para futuras versiones de la herramienta.

   Al usuario (el conductor habitual del vehículo) se le pregunta por el nombre, la edad, el código postal en el que reside (para saber, por ejemplo, si vive en una área urbana especialmente contaminada) y el presupuesto total (cantidad máxima en euros que se quiere gastar en el vehículo). Y se le pide rellenar toda una serie de datos para conocer sus hábitos de desplazamiento en un año. De cada tipo de desplazamiento que suele realizar se le pregunta si es por ocio o por trabajo, los otros ocupantes del vehículo (de cada uno pregunta nombre y edad), la cantidad de equipaje que suele llevar (en litros), cuantos kilómetros hace al día en ese tipo de desplazamiento, el número de días al mes que se realiza, y el lugar de origen y destino de ese desplazamiento. De cada lugar de desplazamiento se quiere saber su tipología (urbano, interurbano asfaltado, tierra, abrupto) y si dispone de una estación de carga para vehículos eléctricos.

   Sobre los vehículos que queremos recomendar (coches y motos) tenemos información almacenada sobre la marca, el modelo, el número máximo de plazas (entre 2 y 7), el tipo de motorización (eléctrico, híbrido o híbrido enchufable), la potencia fiscal (en caballos), el tiempo de carga total de las baterías (en minutos, 0 para los híbridos), el consumo homologado (en litros/100 km), la autonomía homologada (en km), las emisiones homologadas de $C0_2$ (en gr/km) y el precio base (en euros). En el caso de las motos tenemos también el tipo de moto (scooter, triciclo, motocicleta) y su velocidad máxima (en km/h). En el caso de los coches solo realizará recomendaciones en 6 categorías: micro-coche, turismo, monovolumen, todocamino-compacto, todocamino o todoterreno. En el caso de los micro-coches, turismos, monovolumenes y todocaminos-compactos tendremos también la longitud del coche (en metros, con decimales), al ser un dato relevante en la compra. En los micro-coches tendremos también el ancho del vehículo (en metros, con decimales), y en turismos y todocaminos-compactos sabemos si se abaten los asientos traseros o no. Otra información relevante disponible para todocaminos-compactos, todocaminos y todoterrenos es el tipo de tracción (4x2, 4x4 o 4x4 desconectable), la altura libre de la carrocería al suelo (en cm) y si dispone de asistente para pendientes. Los compradores de todoterrenos son un público muy exigente, y por ello en este caso tenemos además información sobre si la caja de cambios dispone de reductora, la profundidad de vadeo (en cm), el ángulo de ataque, ángulo ventral y ángulo de salida (los tres en grados).

   Los expertos en movilidad nos han indicado una serie de criterios importantes a la hora de escoger un vehículo:

   - El **tipo de presupuesto** disponible, que se clasifica en bajo (por debajo de los 10000 euros), medio (entre 10000 y 25000 euros) y elevado (por encima de 25000 euros);
   - El **número de pasajeros** medio de los desplazamientos, que se clasifica en tres grupos: de 1 a 2 pasajeros, de 3 a 5 pasajeros y de 6 a 7 pasajeros;
   - La **necesidad de equipaje** media, que se clasifica en cuatro grupos: muy baja (por debajo de los 10 litros), baja (entre 10 y 100 litros en la mayoria de desplazamientos), media (entre 100 y 250 litros de equipaje medio, o si se lleva un menor de 10 años abordo), y elevada (más de 250 litros de equipaje, o si se lleva dos o más menores de 10 años abordo).
   - El **trayecto medio**, que se clasifica en corto (una mayoría de desplazamientos por debajo de 50 km), medio (desplazamientos con una media entre 50 y 200 km) y largo (con una media de más de 200 km por desplazamiento)
   - Los **kilómetros al año**, en que se distinguen tres grupos: por debajo de 2000 km/año, entre 2000 y 10000 km/año, y por encima de los 10000 km/año.
   - El tipo de **terreno** en el que se suele conducir principalmente, distinguiendo entre cuatro grupos: conductores mayormente urbanos, o mayormente interurbanos, o mixtos (conductores que ocasionalmente salen de vias asfaltadas) o bien off-road (conductores que conducen más del 25% de sus kilómetros en tierra o que conducen en algun momento por terreno abrupto);
   - La **disponibilidad de cargadores**, que estima la facilidad para encontrar una estación de carga en los lugares de origen y destino de los desplazamientos. Se considera una disponibilidad alta si al menos el 90% de los lugares de origen y destino de los desplazamientos de trabajo y el 75% de los lugares en desplazamientos de ocio disponen de estaciones de carga, disponibilidad media si los porcentajes son del 75% y 50% respectivamente, y disponibilidad baja en caso contrario.

Los expertos nos han dado también ejemplos de otros criterios para seleccionar los vehículos a recomendar:

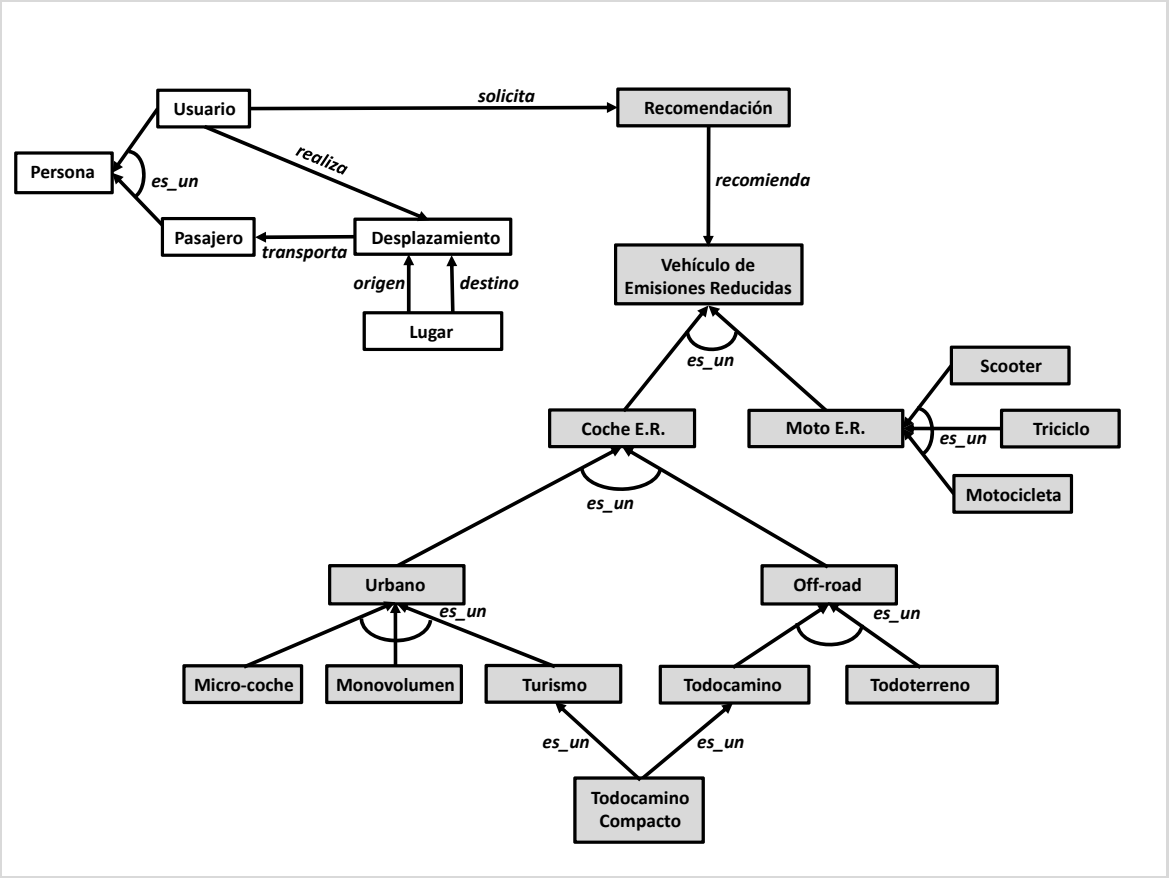
- Podemos determinar que la prioridad principal del usuario es un consumo reducido si sus trayectos suelen ser cortos o medios y si suele hacer más de 10000 kilómetros al año. Pero su prioridad será la autonomía si sus trayectos son medios o largos y si suele hacer menos de 10000 km/año.
- Para escoger el tipo de motorización más adecuada para el usuario se combinan varios factores. Los vehículos eléctricos suelen ser más adecuados cuando la principal prioridad es el consumo, si el trayecto medio es corto y si hay una disponibilidad de cargadores media o elevada. Los vehículos híbridos suelen ser más adecuados cuando la principal prioridad es la autonomía, si el trayecto medio es largo y si la disponibilidad de cargadores es baja. Los vehículos hibridos enchufables combinan lo mejor de los vehículos eléctricos y los híbridos, son aptos para usuarios cuya prioridad sea la autonomía o el consumo y si la disponibilidad de cargadores es media, pero no son aptos para presupuestos bajos.
- Para escoger el tipo de vehículo adecuado para el usuario se tienen en cuenta otros factores. Las motos son recomendables para tipos de presupuesto bajos, cuando el terreno no es off-road, se llevan de 1 a 2 pasajeros y la necesidad de equipaje es muy baja. Los micro-coches son más adecuados para presupuestos bajos o medios, cuando el terreno es mayormente urbano, se llevan 5 o menos pasajeros y la necesidad de equipaje es baja. Los turismos son más adecuados para presupuestos medios, cuando el terreno no es mixto ni off-road, se llevan 5 o menos pasajeros y la necesidad de equipaje es media. Los monovolúmenes son más adecuados para presupuestos altos, cuando el terreno no es mixto ni off-road, se llevan entre 3 y 7 pasajeros y la necesidad de equipaje es elevada. Los todocaminos compactos son más adecuados para presupuestos medios, uso en terrenos mixtos, cuando se llevan 5 o menos pasajeros y la necesidad de equipaje es media. Los todocaminos son más adecuados para presupuestos altos, uso en terrenos mixtos, cuando se llevan entre 3 y 7 pasajeros y la necesidad de equipaje es elevada. Finalmente los todoterrenos son aptos para terrenos off-road.

El sistema dará dos tipos de salidas:

- Para aquellos usuarios con la mayoría de trayectos por debajo de los 200 km y que hacen menos de 2000 km al año el sistema intentará concienciarlos de que no necesitan tener un coche en propiedad, y que lo mejor es que combinen el uso del transporte público con el uso esporádico de un vehículo alquilado en una empresa de renting. Eso si, se le aconsejará el tipo de vehículo ecológico (moto, micro-coche, turismo, monovolumen, todocamino-compacto, todocamino o todoterreno) y el tipo de motorización (eléctrico, híbrido o híbrido enchufable) más adecuado a sus necesidades.
- Para el resto de usuarios recomendará la compra de un vehículo ecológico y mostrará la lista de modelos disponibles en el mercado, incluyendo el precio y los impuestos. La aplicación dispone de una función `mostrar_lista_vehículos(tipo_vehículo, tipo_motorización, presupuesto_total, código_postal)` que dado el tipo de vehículo escogido, el tipo de motorización, el presupuesto total (en euros) y el código postal del usuario muestra por pantalla la lista de vehículos disponibles en el mercado que corresponden al tipo de motorización y vehículo recomendados y cuyo precio base más el importe del impuesto de matriculación que ha de pagar en su zona de residencia están por debajo del presupuesto total.

(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 sobre el usuario, sus desplazamientos y las personas que le acompañan. También ha de incorporar los conceptos de la solución (en gris en la figura a continuación), que incluyen la recomendación a realizar y la taxonomía de vehículos de emisiones reducidas (E.R. en la figura).



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, 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 `Persona`, `Urbano` y `Off-Road`. No se siguió del todo ese criterio en el caso de las subclases de `Moto E.R.`, que a pesar de que las subclases no tienen ni atributos ni relaciones diferenciadas se han añadido siguiendo el heurístico del diseño de ontologías que recomienda que ramas hermanas de la taxonomía tengan un nivel de granularidad similar (en este caso la rama hermana del `Coche E.R.`). Una cosa a remarcar del modelo es que se ha decidido que `Todocamino Compacto` tenga herencia múltiple de `Turismo` y de `Todocamino` ya que de esta manera obtiene todos los atributos que necesita. En caso que necesitaramos usar la ontología con un razonador que no soporte herencia múltiple podríamos hacer que `Todocamino Compacto` herede únicamente de `Todocamino` y añadirle los atributos de `Turismo` que le faltan.

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. Hay otros atributos que se deberían añadir (como el tipo de cambio de marchas, la aceleración de 0 a 100 km/h o los extra que lleva el coche) si quisiéramos una ontología lo más general y reutilizable posible.

- `Persona`: nombre (string), edad (entero);
- `Usuario`: presupuesto_total (euros), código_postal (entero), media_pasajeros (real), media_niños_pasaje (real), media_equipaje (litros), media_trayecto (km), total_kilometros_año (km), media_cargadores_lugar_trabajo (real), media_cargadores_lugar_ocio (real), *Tipo_Presupuesto* (enumeración: {bajo, medio, elevado}), *Num_Pasajeros* (enumeración: {1a2, 3a5, 6a7}), *Necesidad_Equipaje* (enumeración: {muy_baja, baja, media, elevada}), *Terreno* (enumeración: {may_urbano, may_interurbano, mixto, off-road}), *Disponibilidad_Cargadores* (enumeración: {alta, media, baja}), *Trayecto_Medio* (enumeración: {corto, medio, largo}), *Km_Año* (enumeración: {<2000, 2000a10000, >10000}), *Prioridad* (enumeración: {consumo, autonomía, indistinta});
- `Desplazamiento`: motivo (enumeración: {ocio, trabajo }), litros_equipaje (entero), km_dia (entero), dias_mes (entero);

- `Lugar`: tipología (enumeración: {urbano, interurbano asfaltado, tierra, abrupto}), estación_carga? (booleano);
- `Recomendación`: *TIPO_VEHICULO* (enumeración: {moto, micro-coche, turismo, monovolumen, todocamino-compacto, todocamino, todoterreno}), *TIPO_MOTORIZACION* (enumeración: {eléctrico, híbrido, híbrido_enchufable}), *TIPO_ADQUISICION* (enumeración: {compra, renting});
- `Vehículo de Emisiones Reducidas`: marca (string), modelo (string), max_plazas (entero), motorización (enumeración: {eléctrico, híbrido, híbrido_enchufable}), potencia_fiscal_cv (entero), tiempo_carga (minutos), consumo (l/100km), autonomia (km), emisiones_CO2 (gr/km), precio_base (euros);
- `Moto E.R.`: velocidad_máxima (entero);
- `Coche E.R.`: num_puertas (entero), volumen_maletero (litros);
- `Urbano`: longitud (metros);
- `Micro-coche`: ancho (metros);
- `Turismo`: asientos_traseros_abatibles? (booleano);
- `Off-road`: tracción (enumeración: {4x2, 4x4, 4x4_desconectable}), altura_al_suelo (cm), asistente_pendientes? (booleano);
- `Todoterreno`: reductora? (booleano), profundidad_vadeo (cm), ángulo_ataque (grados), ángulo_ventral (grados), ángulo_salida (grados);

Como se puede ver, 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.

(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 no es correcto acceder en el nivel de solución abstracta a valores concretos como el presupuesto total del cliente o el código postal para construir la lista de vehículos recomendados.

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 sobre el usuario, sus desplazamientos, los lugares por los que pasa y los pasajeros, tal y como los hemos identificado en la ontología. En este caso hace falta un preproceso de los datos, ya que en la mayoría de reglas de abstraccion se necesita saber los valores medios (kilómetros recorridos, pasajeros en trayecto, ...) o mayoritarios (tipo de terreno por el que se circula). Esto requiere el cálculo de una serie de valores (por ejemplo, multiplicar para cada tipo de desplazamiento los km_dia * dias_mes * 12 para saber los km_año recorridos en ese desplazamiento, y usar ese valor no solo para calcular los km_año totales sinó también la proporción de km por cada tipo de terreno por el que el usuario circula, basándose en la tipología de los lugares de origen y destino). Suponemos que tenemos métodos para realizar todos estos cálculos a partir de los datos que introduce el usuario, y estos quedan almacenados en la ontología mediante atributos extra que añadimos para tal efecto: `media_pasajeros`, `media_niños_pasaje`, `media_equipaje`, `media_trayecto`, `total_kilometros_año`, `media_cargadores_lugar_trabajo`, `media_cargadores_lugar_ocio`,etc.

El segundo elemento son los problemas abstractos, estos estarán definidos a partir de las siete características que menciona el enunciado (*Tipo_Presupuesto*, *Num_Pasajeros*, *Necesidad_Equipaje*, *Terreno*, *Disponibilidad_Cargadores*, *Trayecto_Medio*, *Km_Año*) al que añadimos un criterio más, la *Prioridad*, ya que esta variable está más relacionada con la descripción del problema abstracto (las necesidades del usuario) que con la descripción de la solución abstracta (el vehículo más adecuado para el usuario).

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

- si Usuario.presupuesto < 10000 euros entonces Usuario.Tipo_Presupuesto=bajo;
- si Usuario.presupuesto > 25000 euros entonces Usuario.Tipo_Presupuesto=elevado;
- si Usuario.media_pasajeros ≤ 2 entonces Usuario.Num_Pasajeros=1a2;
- si Usuario.media_pasajeros > 5 entonces Usuario.Num_Pasajeros=6a7;
- si Usuario.media_equipaje > 25 litros o Usuario.media_niños_pasaje > 1,9 entonces Usuario.Necesidad_Equipaje=elevada;
- si Usuario.media_trayecto < 50km entonces Usuario.Trayecto_Medio=corto;
- si Usuario.total_kilometros_año < 2000 entonces Usuario.Km_Año='<2000';

El tercer elemento son las soluciones abstractas. En este caso tenemos tres aspectos diferentes que analizar (el tipo de vehículo, el tipo de motorización y el tipo de adquisición) y hemos creado una variable cualitativa para cada aspecto, que corresponde con los atributos del concepto `Recomendación` del apartado anterior: *TIPO_VEHICULO* (enumeración: {moto, micro-coche, turismo, monovolumen, todocamino-compacto, todocamino, todoterreno}), *TIPO_MOTORIZACION* (enumeración: {eléctrico, híbrido, híbrido_enchufable}), *TIPO_ADQUISICION* (enumeración: {compra, renting}). Para calcular el valor de la propiedad abstracta `Prioridad` a partir de las otras variables abstractas del problema se requieren una serie de reglas de inferencia previas a la asociación heurística, como por ejemplo:

- si Usuario.Trayecto_Medio=(corto o medio) y Usuario.Km_Año='>10000' entonces Usuario.Prioridad=consumo;
- si Usuario.Trayecto_Medio=(medio o largo) y Usuario.Km_Año=('<2000' o '2000a10000')
- si Usuario.Trayecto_Medio= largo y Usuario.Km_Año='>10000' entonces Usuario.Prioridad=indiferente;
- si Usuario.Trayecto_Medio=corto y Usuario.Km_Año=('<2000' o '2000a10000') entonces Usuario.Prioridad=indiferente;

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

- si Usuario.Prioridad=consumo y Usuario.Trayecto_Medio=corto y Usuario.Disponibilidad_Cargadores=(media o elevada) entonces Recomendación.TIPO_MOTORIZACION=eléctrico;
- si Usuario.Prioridad=autonomia y Usuario.Trayecto_Medio=largo y Usuario.Disponibilidad_Cargadores=baja entonces Recomendación.TIPO_MOTORIZACION=híbrido;
- si Usuario.Tipo_Presupuesto=bajo y Usuario.Num_Pasajeros='1a2' y Usuario.Terreno≠off-road y Usuario.Necesidad_Equipaje=muy_baja entonces Recomendación.TIPO_VEHÍCULO=moto;
- si Usuario.Tipo_Presupuesto=(bajo o medio) y Usuario.Num_Pasajeros=('1a2' o '3a5') y Usuario.Terreno=urbano y Usuario.Necesidad_Equipaje=baja entonces Recomendación.TIPO_VEHÍCULO=micro-coche;
- si Usuario.Terreno=off-road entonces Recomendación.TIPO_VEHÍCULO=todoterreno;
- si Usuario.Trayecto_Medio=(corto o medio) y Usuario.Km_Año='<2000' entonces Recomendación.TIPO_ADQUISICIÓN=renting;

El cuarto y último elemento son las soluciones concretas. En este caso corresponde a la generación de las dos posibles salidas del sistema, aprovechando las soluciones abstractas y refinándolas con datos concretos del problema para poder generar las salidas descritas en el enunciado. En este caso necesitamos solo dos reglas de producción para el refinamiento. Suponemos la existencia de un método `recomendar_renting` para generar la pantalla en caso de que se le quiera recomendar al usuario la opción del reting de un vehículo.

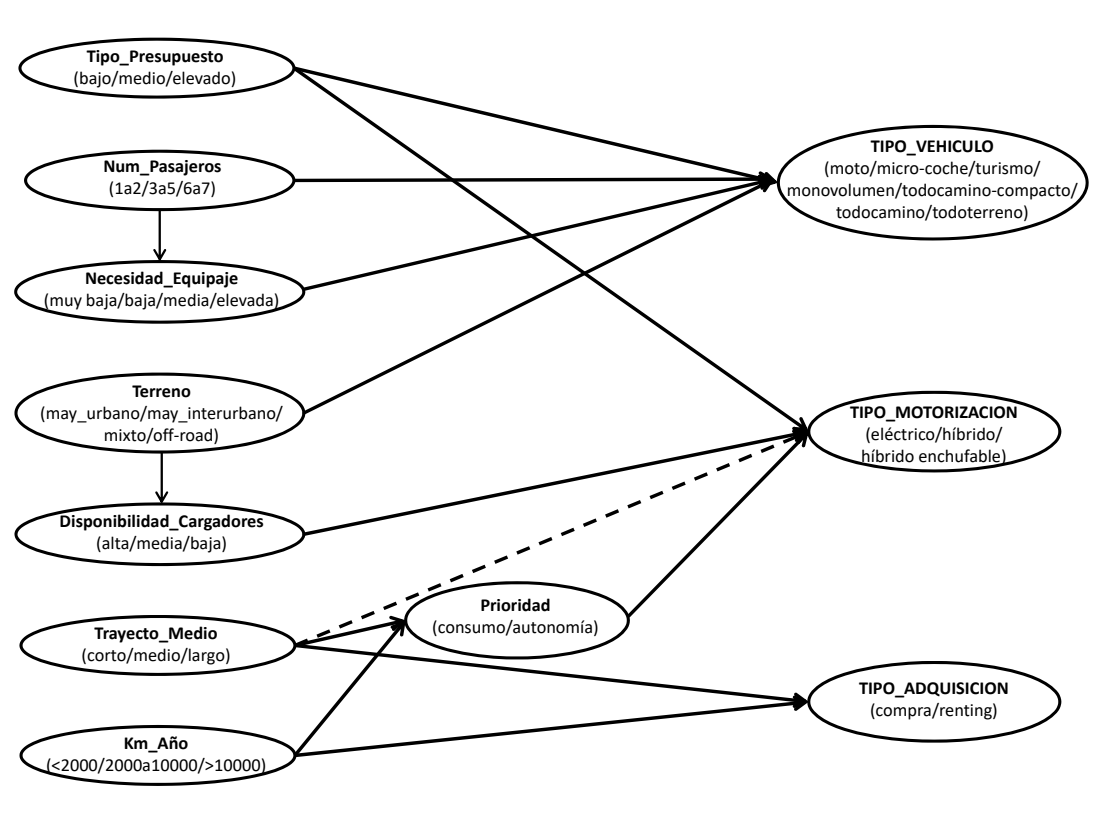
- si Recomendación.TIPO_ADQUISICIÓN=renting entonces recomendar_renting(Recomendación.TIPO_VEHÍCULO, Recomendación.TIPO_MOTORIZACION);
- si Recomendación.TIPO_ADQUISICIÓN=compra entonces mostrar_lista_vehículos(Recomendación.TIPO_VEHÍCULO, Recomendación.TIPO_MOTORIZACION, Usuario.presupuesto_total, Usuario.código_postal)

(c) Las características que se usan durante el proceso de recomendación de vehículos ecológicos no son independientes entre sí. El número de ocupantes en el vehículo suele influir en la necesidad de equipaje (a mayor número de pasajeros, más equipaje) y la disponibilidad de cargadores suele depender del tipo de terreno por el que circula normalmente (mayor disponibilidad en terrenos urbanos, disponibilidad mínima en terrenos off-road). 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 las necesidades del usuario 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 en el apartado anterior. Los nodos de la solución serán los nodos finales de la red bayesiana (a la derecha del gráfico, con el nombre de la variable en MAYÚSCULAS). Los nodos a su izquierda son las características abstractas que modelan las necesidades de movilidad del usuario. El rango de valores de las variables corresponde en casi todas las variables con las enumeraciones de las mismas definidas en el apartado a). La única variable a la que se le ha cambiado el dominio es `Prioridad`, que en los apartados a) y b) tenía un dominio formado por tres valores: `consumo`, `autonomía` e `indistinta`. Al convertir `Prioridad` en una variable aleatoria discreta, el valor `indistinta` corresponde a una distribución de probabilidades donde no destaque por encima ni la probabilidad de `consumo` ni la probabilidad de `autonomía`, y por ello se ha eliminado el valor `indistinta` del dominio.

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. En este caso estaban bastante claras las dependencias entre variables, y la única duda es si toda la influencia que pueda tener la variable `Trayecto_Medio` sobre `Tipo_Motorización` le puede llegar a través de la variable `Prioridad` o si `Trayecto_Medio` es también nodo padre directo de `Tipo_Motorización` (la flecha punteada en la figura).



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 `Prioridad` (que depende de `Trayecto_Medio` y `Km_Año`), la tabla de probabilidad podría ser algo como lo siguiente:

Trayecto_Medio	Km_Año	Prioridad consumo	autonomía
corto	>10000	0,99	0,01
corto	2000a10000	0,9	0,1
corto	<2000	0,7	0,3
medio	>10000	0,6	0,4
medio	2000a10000	0,5	0,5
medio	<2000	0,4	0,6
largo	>10000	0,3	0,7
largo	2000a10000	0,1	0,9
largo	<2000	0,01	0,99

La tabla intenta reflejar, por ejemplo, que la probabilidad de que la prioridad del cliente sea el consumo será mayor cuan mayor sean los kilómetros recorridos al año y cuan más cortos sean los trayectos en media. Es importante asegurarse que las probabilidades de los valores del nodo hijo en cada fila de la tabla sumen 1.

2. (5 puntos) Una empresa española ha decidido entrar en el sector de los coches autónomos poco a poco. Bajo el nombre de *GoAuto* ha desarrollado tres tipos de vehículo eléctrico totalmente autónomos que no tienen volante ni otro tipo de mecanismos de manejo manual: el *GoAuto*2, vehículo de 2 plazas similar al de Google, el *GoAuto*4 con similares características y 4 plazas, y el *GoAuto*7 con 7 plazas. Para ayudar al desarrollo de la tecnología la empresa ha creado un proyecto piloto de movilidad urbana autónoma personalizada en las calles de una pequeña ciudad para substituir a los taxis y a los micro-buses. Se ha creado una red de 15 *GoPuntos*, puntos de recogida que substituyen a las antiguas paradas de bus y taxi de la población, con la idea de crear más puntos en el futuro. Los ciudadanos pueden pedir a través de una app que uno de estos vehículos los lleve desde un punto de recogida a otro punto en la ciudad. Alguno de los vehículos autónomos que tenga plazas libres recogerá al ciudadano en el punto de recogida y lo llevará al punto de

destino, pudiendo parar por el camino en otros puntos para recoger o descargar pasajeros. La empresa nos pide que desarrollemos un planificador simple que genere un plan para servir las peticiones de movilidad de los ciudadanos con los vehículos autónomos disponibles en cada momento (al ser prototipos a veces falla alguno y se han de servir las peticiones con el resto).

(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 *GoPuntos* y más o menos *GoAutos*.

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 GoAuto)
  (:requirements :adl :typing)

  (:types persona lugar goauto - object
          plaza gopunto - lugar)

  (:predicates
   (estacionado ?coche - goauto ?l - lugar)
   (dentro ?pl - plaza ?coche - goauto)
   (libre ?pl - plaza)
   (destino ?p - persona ?g - gopunto)
   (en ?p - persona ?l - lugar)
   (pendiente ?p - persona)
   (servido ?p - persona)
  )

  (:action montar_en_goauto
    :parameters (?pe - persona ?go - goauto ?pl - plaza ?l - gopunto)
    :precondition (and (pendiente ?pe) (en ?pe ?l)
                       (estacionado ?go ?l)
                       (dentro ?pl ?go) (libre ?pl)
                  )
    :effect (and (en ?pe ?pl)
                 (not (en ?pe ?l)) (not (libre ?pl)) (not (pendiente ?pe))
            )
   )

  (:action bajar_de_goauto
    :parameters (?pe - persona ?go - goauto ?pl - plaza ?l - gopunto)
    :precondition (and (en ?pe ?pl) (dentro ?pl ?go)
                       (estacionado ?go ?l)
```

```
                        (destino ?pe ?l)
                    )
    :effect (and (en ?pe ?l) (libre ?pl) (servido ?pe)
                    (not (en ?pe ?pl))
                    )
   )

  (:action mover_go_auto
    :parameters (?go - goauto ?ori - gopunto ?des - gopunto)
    :precondition (estacionado ?go ?ori)
    :effect (and (estacionado ?go ?des) (not (estacionado ?go ?ori))
            )
  )
)
```

En esta solución se distingue entre personas (los usuarios del servicio), goAutos (los coches autónomos que estan disponibles), las plazas dentro de cada coche (cada plaza es un asiento en el que podemos poner una persona dentro del coche, modelamos la capacidad de cada coche asignando cada plaza disponible a un coche dado) y goPuntos (las paradas donde pueden subir y bajar usuarios). El tipo lugar se ha creado para poder usar polimorfismo en el predicado en, y usar un solo predicado para decir que una persona o bien está en un goPunto o bien en la plaza dentro de un coche.
La lista de predicados es la siguiente:

- `estacionado`, que nos dice en que goPunto está parado el coche en cada momento.
- `dentro`, que nos dice que una plaza esta dentro de un cierto coche.
- `en`, un predicado polimórfico que nos dice en que lugar (goPunto o plaza dentro de coche) está la persona. También lo usaremos para describir el punto de origen de cada usuario.
- `destino`, que nos dice cual es el goPunto al que quiere llegar el usuario.
- `libre`, que nos dice que la plaza no esta ocupada por ningun usuario. Podríamos usar el predicado `en` para saber si una plaza tiene alguna persona sentada, pero para saber si una plaza esta libre o no tendríamos que comprobar que para todas las personas en el modelo del problema no hay ninguna persona que esté `en` esa plaza (y esto implica el uso de `forall` o de `(not exists)`). Por ello es más efectivo añadir el predicado `libre`, que sabiendo la plaza nos dice si está libre o no.
- `pendiente`, que nos dice si una persona está pendiente de ser transportada. Se usa como filtro para restringir las personas que la precondición del operador `montar_en_goauto` intenta explorar (una vez el usuario sube al coche, ya no está pendiente y no se va a volver a considerar como candidato a subir a un coche).
- `servido`, que nos dice si una persona ya ha sido transportada a su destino. El plan acabará cuando todos los usuarios estan servidos (que es una condición más fácil de comprobar que mirar, para toda persona, que esta en un goPunto p y que ese goPunto es su destino). Es importante remarcar que en este modelo `servido` no se puede substituir por `(not pendiente)`, ya que de hecho con los dos predicados podemos modelar tres estadios de la persona: pendiente de ser recogida, no pendiente de ser recogida pero no ha llegado a destino, y persona que ha llegado a su destino.

En vez de añadir un predicado `averiado` para indicar los coches que no estan disponibles, se ha optado por declarar en el fichero de problema solo aquellos goAutos que funcionan.
Se ha simplificado el problema a un modelo con solo tres operadores: `montar_en_goauto`, `bajar_de_goauto` y `mover_go_auto`, que son autoexplicativos. El movimiento del coche por la ciudad se ha simplificado a lo mínimo necesario, de forma que `mover_go_auto` hace que el coche se mueva de un goPunto a otro, sin modelar el camino que sigue. Las precondiciones se han colocado de forma que minimicen los objetos que se prueban (por ejemplo, tal como hemos dicho antes el operador `montar_en_goauto` solo explora usuarios con el predicado `pendiente`). En el operador `mover_go_auto` no se han colocado restricciones (moverse a goPuntos con personas pendientes de ser recogidas o que son destino de alguien dentro del coche) e intentará moverse entre un punto y todos los demás. Con esto reducimos la longitud del plan, pero sólo podemos hacerlo porque el tamaño del problema es pequeño (15 goPuntos en un área urbana de tamaño reducido); si se quisiera generalizar el modelo para tamaños de problema más grandes entonces posiblemente se debería restringir el operador, poniendo algun tipo de restriccion en las rutas entre goPuntos para reducir el factor de ramificación.

Es importante remarcar que este modelo no solo permite modelar más o menos goPuntos, más o menos personas y más o menos coches, sino también más o menos asientos dentro de cada coche.

(b) El jefe del proyecto nos ha proporcionado una tabla como ejemplo de las peticiones de movilidad que el sistema recibe cada 15 minutos, con una serie de ciudadanos que se han de transportar entre los 15 puntos de la red (GoPunto1, GoPunto2 ... GoPunto15). También nos dice que al inicio de ese intervalo de 15 minutos tenemos un vehículo *GoAuto*2 de 2 pasajeros vacio estacionado en GoPunto3 y un vehículo *GoAuto*4 de 4 pasajeros vacio estacionado en GoPunto7, y que el vehículo *GoAuto*7 no está disponible por avería. Nos pide que el planificador genere un plan en el que los vehículos disponibles acaben llevando todos los ciudadanos a sus puntos de destino.

ciudadano	GoPunto origen	GoPunto destino
Julian	GoPunto1	GoPunto3
Paula	GoPunto7	GoPunto12
David	GoPunto14	GoPunto10
Eva	GoPunto3	GoPunto5
Laura	GoPunto3	GoPunto7
Felipe	GoPunto3	GoPunto8
Amelia	GoPunto1	GoPunto5
Ivan	GoPunto13	GoPunto8
Daniela	GoPunto6	GoPunto11
Hugo	GoPunto12	GoPunto10

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. El resultado es el siguiente:

```
(define (problem GoAuto-2coches-15puntos-10usuarios)
   (:domain GoAuto)

   (:objects Julian Paula David Eva Laura Felipe Amelia Ivan Daniela Hugo - persona
             GoAuto2 GoAuto4 - goauto
             p21 p22 p41 p42 p43 p44 - plaza
             GoPunto1 GoPunto2 GoPunto3 GoPunto4 GoPunto5 GoPunto6
             GoPunto7 GoPunto8 GoPunto9 GoPunto10 GoPunto11 GoPunto12
             GoPunto13 GoPunto14 GoPunto15 - gopunto
    )

(:init
  (en Julian GoPunto1) (destino Julian GoPunto3)
  (en Paula GoPunto7) (destino Paula GoPunto12)
  (en David GoPunto14) (destino David GoPunto10)
  (en Eva GoPunto3) (destino Eva GoPunto5)
  (en Laura GoPunto3) (destino Laura GoPunto7)
  (en Felipe GoPunto3) (destino Felipe GoPunto8)
  (en Amelia GoPunto1) (destino Amelia GoPunto5)
  (en Ivan GoPunto13) (destino Ivan GoPunto8)
  (en Daniela GoPunto6) (destino Daniela GoPunto11)
  (en Hugo GoPunto12) (destino Hugo GoPunto10)
  (estacionado GoAuto2 GoPunto3)
  (estacionado GoAuto4 GoPunto7)
  (dentro p21 GoAuto2) (libre p21)
  (dentro p22 GoAuto2) (libre p22)
  (dentro p41 GoAuto4) (libre p41)
  (dentro p42 GoAuto4) (libre p42)
  (dentro p43 GoAuto4) (libre p43)
  (dentro p44 GoAuto4) (libre p44)
  (pendiente Julian) (pendiente Paula)
  (pendiente David) (pendiente Eva)
```

```
  (pendiente Laura) (pendiente Felipe)
  (pendiente Amelia) (pendiente Ivan)
  (pendiente Daniela) (pendiente Hugo)
)

(:goal (forall (?p - persona) (servido ?p)))
)
```

Se ha creado un objeto por cada goPunto, cada coche disponible (`GoAuto7` no se incluye al estar averiado) y cada persona. También hemos creado seis plazas de coche, dos que están dentro del `GoAuto2` y cuatro más que están dentro del `GoAuto4`. Los nombres intentan hacerlo más facil de leer, pero al planificador le da lo mismo el nombre escogido para cada objeto.
El estado inicial es una fotografia de la configuración que aparece en el enunciado, indicando el goPunto origen y destino de cada usuario, la posición inicial de cada uno de los goAutos y la pertenencia de cada plaza a los coches. Se añade tambien el predicado `pendiente` a todos los usuarios.
El estado objetivo usa el predicado `servido` (el plan acaba cuando todos los usuarios han sido servidos por los coches disponibles). Se podría haber descrito listando los diez predicados `servido` (uno por persona), pero en este caso se ha optado por la expresión `forall` en adl, ya que nos permite cambiar el número de personas del problema sin tener que cambiar la descripción del estado objetivo, y no nos añade complejidad (en FastForward 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 publicarán el día **25 de enero**.