Práctica 3 de Inteligencia Artificial

[Grau - IA]

Planificación de películas y series para

Redflix.

1r cuatrimestre - Curso 2016-2017

Ismael Julià - Ana Mestre - Gorka Piñol

Índice

1. El problema.

La empresa *Redflix,* un distribuidor de contenidos audiovisuales, nos ha solicitado que le creemos un programa para gestionar el visionado de películas y series de televisión para los usuarios.

El factor más importante a tener en cuenta es la relación que hay entre los diferentes contenidos, es decir hay capítulos de series que son predecesores a otros capítulos o a alguna película, igual que hay películas predecesoras a otras películas o a alguna serie, por lo que hay que considerar el orden lógico de visionado. De igual forma, hay también contenidos paralelos, es decir, contenidos que deben ser visualizados previamente o el mismo día que otro en concreto.

Por último hay que tener en cuenta no sobrepasar el máximo tiempo que se puede visualizar una serie y/o película al día, que es de 200 minutos.

2. Dominio

- Variables

- contenido: tipo que indica que el objeto es un contenido (película, serie, etc).
- dia: tipo que indica que el objeto es un día.

- Funciones

- (ordenDia ?d - dia) : número que tiene un día en concreto, para poder establecer un orden.
- (ultimoDia ?c - contenido) : el último día que se le ha asignado a un predecesor de este contenido, para poder cumplir el orden entre contenidos con predecesores.
- (predecesores ?c - contenido) : la cantidad de predecesores que tiene cada contenido, para poder cumplir el orden entre contenidos con predecesores.
- (predecesoresAsignados ?c - contenido)) : la cantidad de predecesores de cada contenido que ya se ha asignado a un día.

- Predicados

- (predecesor ?x - contenido ?y - contenido): indica si el contenido x es anterior al contenido y.
- (ver ?x - contenido): indica que se debe ver el contenido x.
- (contenidoAsignado ?c - contenido) : indica si un contenido ya tiene asignado un día para su visualización.

- Acciones

- ver_predecesor(x, y): si *x* es predecesor de *y* y *x* no se ha visto pero se desea ver *y*, marca que se debe ver x.
- asignar_contenidos(d, x): para cada combinación de día *d* y contenido *x* busca cómo asignar los días para que se cumplan las restricciones de predecesores de cada contenido.

3. Modelización

En esta apartado se describen los diferentes elementos que aparecen en los distintos modelos desarrollados para cada una de las versiones del problema que han sido consideradas. En cada extensión se han añadido nuevas funcionalidades al modelo, así que en cada descripción sólo se añadirán los nuevos cambios.

3.1 Nivel básico

Todos los contenidos tienen 0 o 1 predecesores y ningún contenido es paralelo. El planificador deberá ser capaz de encontrar los contenidos deseados, donde estos tengan uno o ningún predecesor.

En este modelo se define únicamente un tipo (type): contenido, siendo este subtipo de object, restringiendo así las posibilidades de unificación. Sólo nos hace falta un type, ya que sólo trataremos de encontrar un contenido adecuado y nada más.

Por tanto, tenemos los predicados y las funciones siguientes:

(predecesor ?x - contenido ?y - contenido): contenido que debe ser visto antes que otro
(ver ?x - contenido): indica que un contenido ya puede ser visto. Cuando todos estén listos para ser vistos se habrá llegado a la solución que buscamos.
(predecesores ?c - contenido): lo usamos para saber el número de predecesores que tiene cada contenido

Los contenidos tienen tres estados: se quieren ver en un futuro, ya han sido vistos o en ninguno de los dos (no consideramos este último estado).

En esta versión, la asignación de orden de contenidos se realiza en dos acciones, la primera *ver_predecesor* se encarga de que hayamos visto los predecesores de un contenido objetivo antes de llegar a este, en el caso que no tenga predecesores, el objetivo es la solución. Hay que tener en cuenta que en el nivel básico ya se mira para más de un predecesor, así que la primera parte de la extensión 1 también se tiene cubierta.

3.2 Primera extensión

Esta extensión es construida a partir del problema básico, añadiendo de 0 a N predecesores de los contenidos fijados y los días, estando todos los predecesores en días anteriores. Debemos incluir que ya no es únicamente un predecesor, sino que debemos llegar hasta la raíz (el

primer predecesor), además de que debemos indicar todos los días disponibles para ver los diferentes contenidos.

Por tal de hacer eso debemos incluir dos predicados nuevos:

(contenidoAsignado ?c - contenido): para saber si un contenido ya ha estado asignado a un día concreto.
(predecesoresAsignados ?c - contenido): indica el número de predecesores de cada contenido que ya está asignado en un dia.

Empezamos comprobando primero de todo si el contenido es el objetivo a ver y si aún no ha sido asignado a ningún día, luego debemos mirar si todos los predecesores han sido asignados:

(= (predecesores ?x) (predecesoresAsignados ?x)).

Si ha sido asignado, el effect hará que para cada predecesor de un contenido objetivo se asigne a un día anterior al contenido objetivo:

(increase (predecesoresAsignados ?y) 1) .

3.3 Tercera extensión.

Para esta versión del prototipo hemos tenido que asegurarnos de no superar la máxima cantidad de contenidos que podemos asignarle a un día, que para nuestro caso serían tres. La manera en que la hemos planteado ha sido con la función *capacidadDia*.

Esta extensión está trabajada en la action *asignar_contenido*: cada vez que queremos asignarle un día a cada contenido, a parte de tener en cuenta que se cumplan las restricciones para los predecesores y que el contenido no haya sido asignado aún, miramos que la *capacidadDia* para ese día en concreto sea menor que 3, en caso de que lo cumpla, entonces podemos asignar un contenido a ese día y el siguiente paso es aumentar la *capacidadDia* en una unidad.

3.4 Cuarta extensión.

El planteamiento usado para esta extensión es bastante similar al necesario en la extensión anterior. Tenemos que añadir al dominio las funciones *minutosContenido* y *minutosDia*. En el problema habría que inicializar *minutosContenido* para cada contenido con los minutos de visualización que son necesarios. Para que sea lógica la resolución, este valor tendría que ser menor o igual a 200. También es necesario inicializar *minutosDia* a 0. Partiendo de estos valores, en la action *asignar_contenido* tenemos que añadir la siguiente precondición: la suma de los minutos que dura un contenido y de los minutos que tiene acumulados aquel día tiene que ser menor o igual a 200 minutos. Si cumple esta restricción, entonces podemos asignar ese día e incrementamos el valor *minutosDia* con el número que obtenemos de la función *minutosContenido.*

4. Instancias del problema

Los diferentes objetos que nos encontramos en el problema son los contenidos y días para ver los contenidos. La entrada consiste en una enumeración de los contenidos en la versión básica, y de los días disponibles para verlos en las siguientes extensiones. Se inicializan diferentes funciones, por una parte tenemos todos los contenidos que se quieren ver, que formarán el objetivo del problema, que obviamente son mutuos en todos los apartados (menos en el básico, en el cual sólo se usan los contenidos) y por otro lado tenemos los diferentes días también enumerados.

Nivel básico: utilizamos la función *predecesor*, por tal de decir qué contenidos son anteriores a otros para llegar hasta nuestro objetivo. Por otro lado tenemos los contenidos objetivo que queremos ver con la función *ver*. Por último, cuando los contenidos que tenemos para *ver* ya estén asignados juntos con sus predecesores habremos llegado al goal: *(:goal (and (forall (?x - contenido) (not (ver ?x))))* y por lo tanto ya habremos llegado al estado final de todos los contenidos objetivo que habíamos marcado al principio para ser vistos.

Extensión 1: añadimos los días como objetos y las funciones *predecesoresAsignados* por tal de saber los predecesores que ya han sido asignados en los diferentes días que proponemos, y *ordenDia*, por tal de saber qué día va antes. Debemos remarcar que si necesitamos que los predecesores estén en días anteriores a los contenidos objetivo necesitaremos N + 1 días, siendo N el número de contenidos que hay, esto es necesario porque empezamos con el día 0, que nunca asignaremos a un contenido, pero lo necesitamos para hacer las comparaciones entre días anteriores y posteriores. De esta manera tenemos realmente tantos días para asignar como contenidos para poder garantizar una solución.

Extensión 3: utilizamos dos nuevas funciones, *ultimoDia* para saber el último día que se le ha asignado a un predecesor del contenido objetivo y *capacidadDia* para saber cuántos contenidos se han asignado en un día, ya que como mucho pueden haber tres y necesitamos llevar un control a medida que hacemos las diferentes asignaciones a los contenidos predecesores y objetivo. Llegaremos al estado final cuando todos los contenidos estén asignados para ser vistos con la restricción previamente marcada.

Extensión 4: por último debemos fijar un total de 200 minutos por día, así que debemos añadir *minutosDia* igual al número de días y inicializado a cero para controlar que con *minutosContenido* que irán sumando en cada día asignado no nos pasemos del tiempo por día. Nos encontraremos en el estado final cuando estén todos los contenidos predecesores y objetivo asignados a los diferentes días, cumpliendo para cada día la restricción de tiempo.

5. Desarrollo de los modelos

Para el desarrollo de los distintos problemas hemos optado por un diseño incremental basado en prototipos siguiendo el guión del enunciado de la práctica. Empezamos por el problema básico y se fue ampliando poco a poco, teniendo en cuenta cada una de las extensiones, menos la 2, que después de haberlo intentado no conseguimos integrarla en las siguientes extensiones. En cada prototipo hemos ido añadiendo nuevos elementos del problema hasta llegar a la última extensión. Cabe destacar que la implementación de cada prototipo se ha hecho sin desarrollar diversos prototipos entre extensiones, ya que la dificultad nos permitía saltar de extensión en extensión en pocos pasos. A parte de los modelos en PDDL hemos desarrollado dos generadores, centrados en dar entradas para la extensión 1 + 3 y extensión 1 + 4, y lo hemos utilizado para hacer los diversos experimentos de la siguiente sección.

6. Juegos de prueba

6.1 Primer juego de pruebas.

Usando como dominio el nivel básico.

Input:

```
(define (problem test-01) (:domain planificador)
        (:objects c1 c2 c3 c4 c5 - contenido)
        (:init
        (predecesor c1 c2)
        (predecesor c2 c3)
        (predecesor c4 c5)

        (ver c3)
        (ver c5)
        )

        (:goal (and (forall (?x - contenido) (ver ?x))))
```

Output:

```
step    0: VER_PREDECESOR C4 C5
        1: VER_PREDECESOR C2 C3
        2: VER_PREDECESOR C1 C2
```

Justificación:

La intención del juego de pruebas es la de ver el contenido 3 y el 5 pero inevitablemente tiene que ver también el 4, el 2 y el 1 porque son predecesores de los que queremos ver y tal como muestra el output, lo cumple de forma satisfactoria.

6.2 Segundo juego de pruebas.

Usando el dominio del planificador para la extensión 1 con la extensión 3.

Input:

```
(define (problem extensio3) (:domain planificador)
        (:objects c0 c1 c2 c3 c4 c5 - contenido
                d0 d1 d2 d3 d4 d5 d6 - dia)
(:init
(predecesor c1 c0)
(predecesor c0 c5)
(ver c0)
(ver c1)
(ver c2)
(ver c5)
(= (ordenDia d0) 0)
(= (ordenDia d1) 1)
(= (ordenDia d2) 2)
(= (ordenDia d3) 3)
(= (ordenDia d4) 4)
(= (ordenDia d5) 5)
(= (ordenDia d6) 6)
(= (ultimoDia c0) 0)
(= (ultimoDia c1) 0)
(= (ultimoDia c2) 0)
(= (ultimoDia c3) 0)
(= (ultimoDia c4) 0)
(= (ultimoDia c5) 0)
(= (capacidadDia d0) 0)
(= (capacidadDia d1) 0)
(= (capacidadDia d2) 0)
(= (capacidadDia d3) 0)
(= (capacidadDia d4) 0)
(= (capacidadDia d5) 0)
(= (capacidadDia d6) 0)
(= (predecesoresAsignados c0) 0)
(= (predecesoresAsignados c1) 0)
(= (predecesoresAsignados c2) 0)
(= (predecesoresAsignados c3) 0)
(= (predecesoresAsignados c4) 0)
(= (predecesoresAsignados c5) 0)
(= (predecesores c0) 1)
```

```
(= (predecesores c1) 0)
(= (predecesores c2) 0)
(= (predecesores c3) 0)
(= (predecesores c4) 0)
(= (predecesores c5) 1)
)

(:goal (forall (?x - contenido) (imply (ver ?x) (contenidoAsignado ?x)))))
```

Output:

```
step    0: ASIGNAR_CONTENIDOS D1 C2
        1: ASIGNAR_CONTENIDOS D1 C1
        2: ASIGNAR_CONTENIDOS D2 C0
        3: ASIGNAR_CONTENIDOS D3 C5
```

Justificación:

Como se puede ver en el output, C1 se visiona antes que C0 y C0 antes que C5, tal y como se han indicado en el input las relaciones de predecesores. C2 se visiona al principio (aunque podría verse en cualquier posición, dado que no tiene ninguna relación con el resto de contenidos) y en ninguno de los días se ven más de tres contenidos.

6.3 Tercer juego de pruebas:

Usando el dominio del planificador para la extensión 1 + la extensión 4.

Input:

```
(define (problem extensio4) (:domain planificador)
        (:objects c0 c1 c2 c3 - contenido
                d0 d1 d2 d3 d4 - dia)
(:init
(predecesor c1 c2)
(predecesor c1 c3)
(ver c0)
(ver c1)
(ver c2)
(= (ordenDia d0) 0)
(= (ordenDia d1) 1)
```

```
(= (ordenDia d2) 2)
(= (ordenDia d3) 3)
(= (ordenDia d4) 4)
(= (ultimoDia c0) 0)
(= (ultimoDia c1) 0)
(= (ultimoDia c2) 0)
(= (ultimoDia c3) 0)
(= (minutosContenido c0) 120)
(= (minutosContenido c1) 81)
(= (minutosContenido c2) 100)
(= (minutosContenido c3) 30)
(= (minutosDia d0) 0)
(= (minutosDia d1) 0)
(= (minutosDia d2) 0)
(= (minutosDia d3) 0)
(= (minutosDia d4) 0)
(= (predecesores c0) 0)
(= (predecesores c1) 0)
(= (predecesores c2) 1)
(= (predecesores c3) 1)
(= (predecesoresAsignados c0) 0)
(= (predecesoresAsignados c1) 0)
(= (predecesoresAsignados c2) 0)
(= (predecesoresAsignados c3) 0)
)

(:goal (forall (?x - contenido) (imply (ver ?x) (contenidoAsignado ?x)))))
```

Ouput:

```
step    0: ASIGNAR_CONTENIDOS D3 C0
        1: ASIGNAR_CONTENIDOS D1 C1
        2: ASIGNAR_CONTENIDOS D2 C2
```

Justificación:

El planificador asigna correctamente a los días el contenido a visionar, teniendo en cuenta las restricciones de tiempo (200 minutos diarios) y el orden de predecesores (C1 antes que C2).

7. Conclusión

Hemos resuelto un problema de planificación a partir de un modelo y hemos usando el mismo sistema de planificación para su resolución automática. En esta práctica hemos analizado un problema de asignación de contenidos que desea ver un usuario y hemos modelizado los elementos a partir de objetos, predicados y acciones propias del PDDL. Por tal de lograrlo, hemos seguido un desarrollo en el cual íbamos incluyendo las nuevas extensiones propuestas en el guión de la práctica. Por tal de solucionar las diferentes instancias hemos usado Fast Forward y hemos comentado los resultados obtenidos, gracias a ello hemos podido comprobar que este tipo de sistema son capaces de ofrecer al usuario una buena planificación en un tiempo de ejecución razonable.