PRÁCTICA DE PLANIFICACIÓN

Inteligencia Artificial

Fèlix Arribas Pardo, **David Williams Corral** y **Raúl García Fuentes**

El problema nos situa en un hotel con varias habitaciones, cada habitacion tiene una capacidad de 1 a 4, y varias reservas, también con capacidades diferentes. Nos piden que mediante `PDDL` encontremos la mejor solución, la mejor asignación de las reservas de habitaciones en el hotel en 30 dias. El problema es cada vez más complejo a medida que implementamos las 4 extensiones.

Nivel Básico

Dominio

Para representar nuestro sistema de reservas del hotel hemos decidido representar tres tipos necesarios, estos son **habitación** (*room*), **reserva** (*booking*) y **día** (*day*). Todas las soluciones trabajan con estos objetos. Las relaciones entre estos tipos son bastante sencillas y triviales:

- Una habitación esta libre (o no) un día en concreto.
- Una reserva és para unos dias.

A partir de estas dos sentencias sacamos dos predicados básicos, pero necesitamos uno mas para saber si una reserva se ha realizado, es decir, ya está procesada:

```
; true iff the room is empty that day
(free ?room - room ?day - day)

; true iff booking is for that day
(booked ?booking - booking ?day - day)

; true iff booking is satisfied
(scheduled ?booking - booking)
```

El predicado ***free*** es cierto si la habitación esta libre ese dia. El predicado ***booked*** es cierto si la reserva se ha cumplido para ese dia. El predicado ***scheduled*** es cierto si la reserva se ha satisfecho.

Para poder asignar una reserva a una habitación y que no haya problemas con la capacidad y el número de huéspedes, usamos dos funciones para obtener el *tamaño* de la habitación y el número de huéspede de la reserva:

```
; size of the room
(room_size ?room - room)
```

```
; amount of people of the booking
(book_size ?booking - booking)
```

Con todo esto, sólo necesitamos una acción: reservar (o *book*). Usa `(>= (room_size ?room) (book_size ?booking)` y comprueba como precondición que, para toda la duración de la reserva la habitación esté libre. También tine en cuenta que la reserva no esté satisfecha ya. Esta acción solo necesita dos parámetros: `(?room - room ?booking - booking)`. Finalmente el efecto que tiene sobre el hotel es:

- Reserva satisfecha: `(scheduled ?booking)`.
- Para todos los dias de la reserva, la habitación no esta libre: `(forall (?day - day) (when (booked ?booking ?day) (not (free ?room ?day))))`.

Problema

Para modelar el problema definimos una serie de objetos, definiendo su nombre, un guión y su tipo, y luego los inicializamos. En nuestro problema definimos la cantidad de habitaciones de la manera *roomX - room* siendo X el número de habitación. Lo mismo con las reservas, las declaramos como *bookXY - booking* de manera que X número de reserva y Y es el número de personas en la reserva. También declaramos los dias que durará nuestro problema de la manera *dayX - day* siendo X el número del dia (del 1 al 30).

La inicialización de estos objetos consiste básicamente en colocar las habitaciones en un estado inicial de libre, inicializar los predicados booked con el dia que la reserva X quiere reservar. Declarar el tamaño de cada habitación de la siguiente manera `(= (room_size roomX) Y)` y asi mismo declarar el tamaño de la reserva `(= (book_size bookXY) Y)`.

En el problema nos queremos asegurar que se reserva almenos una habitación. Por eso miramos que para todas las reservas, haya alguna (`or`) satisfecha:

```
(:goal (or (forall (?book - booking) (scheduled ?book))))
```

Extensión 1

En esta extensión se pide una optimización que, o bien ya hemos hecho con el nivel bàsico, o no sabemos encontrar.

Extensión 2

Dominio

En esta extensión se pide una optimización que, o bien ya hemos hecho con el nivel bàsico, o no sabemos encontrar.

Como requerimos añadir nueva información, añadimos funciones. Las funciones nuevas son, la orientación de la habitación, la orientación preferente de la reserva y por último el número total de reservas que se han quedado con un habitación sin su orientación preferente. Se usa un número para representar la orientación (podria ser: 1 - Norte, 2 - Este, 3 - Sur, 4 - Oeste):

```
; orientation of the room
(room_orientation ?room - room)

; prefered orientation of the booking
(book_orientation ?booking - booking)

; total of non oriented bookings
(non_oriented_bookings)
```

Obviamente también ha hecho falta modificar la acción de book, pero no ha hecho falta crear una nueva. Como precondición mira si hay alguna **habitación disponible con la orientación preferente** libre durante todos los días de la reserva y del tamaño adecuado. También mira si hay alguna habitación disponible que cumpla todos los requisitos pese a que **no sea de la orientación deseada**. Si se realiza una reserva cuya habitación no tiene la orientación deseada se suma uno al número de reservas que no tienen la orientación deseada.

```
(when (not (= (book_orientation ?booking) (room_orientation ?room)))
    (increase (non_oriented_bookings) 1))
```

En el problema se usa `Metric-FF` para minimizar el numero de reservas con una orientación incorrecta (no preferida por el huésped):

```
(:goal (and (forall (?book - booking) (scheduled ?book))))
(:metric minimize (non_oriented_bookings))
```

Extensión 3

Dominio

Ahora nos deshacemos de la orientación de la habitación, pero intentamos reducir las plazas desperdiciadas. Una habitación para 4 personas con sólo una persona nos supone un *waste* de plazas. Y esa es la función que utilizaremos. Que almacena el número total de plazas desperdiciadas, número que queremos minimizar.

En el efecto de la acción de reservar añadimos un incremento al waste `(increase (waste) (- (room_size ?room) (book_size ?booking)))` para cada dia que se hace la reserva `(forall (?day - day) (when (booked ?booking ?day) ... ))`.

Problema

Vamos a querer minimizar *waste*, así que volvemos a usar `Metric-FF`:

```
(:goal (and (forall (?book - booking) (scheduled ?book))))
(:metric minimize (waste))
```

Extensión 4

Dominio

Finalmente, en esta extensión queremos abrir el mínimo posible de habitaciones. Por delante de el desperdicio de plazas. Para este problema añadimos una nueva función y un nuevo predicado:

```
; funcion: number of rooms booked at least once
(different_rooms_booked)

; predicate: true iff room is booked at least once
(used ?room - room)
```

Cada vez que se haga una reserva en una habitación `not used`, sin haberse utilizado antes, incrementaremos el valor de las habitaciones utilizadas, o `different_rooms_booked`.

Problema

El problema que encontramos es que se le esta dando el mismo valor a un plaza desperdiciada que a una habitación usada. Por eso en el problema tenemos en cuenta esto y se multiplica por una constante el número de habitaciones utilizadas.

```
(:goal (and (forall (?book - booking) (scheduled ?book))))
(:metric minimize (+ (waste) (* (different_rooms_booked) 90))
```

¿Porque 90? Es el maximo desperdicio que puede generar una reserva: Una reserva del 1 al 30 (30 días) de una sola persona en una habitación de 4 plazas. Esto es: `(4 plazas - 1 persona) * 30 días = 90 plazas despediciadas`.

Ejecución de la pràctica

Para ejecutar esta práctica se necesita **Metric-FF**. Nos ha costado mucho conseguir ejecutar código, ya que este compilador solo nos funcionaba en los ordenadores de la universidad.

```
# dentro de Metric-FF/
./ff -o <path>/basicDomain.pddl -f <path>/basicProblem.pddl -O
```

Problemas de prueba

Los problemas disponibles són muy sencillos.

Nivel básico

Para el nivel báscio tenemos 10 reservas, cada una un día diferente, para una persona, y 10 reservas más, cada una un día diferente, para dos personas. También disponemos de dos habitaciones, una de dos plazas y otra de sólo una. Únicamente hace falta comprobar que no hay ninguna reserva de dos personas en la habitación de una plaza.

Extensión 2

Aqui influye la orientación. Por eso hemos creado dos reservas, predefiniendo la orientación a 1. Estas son en dias diferentes, no se solapan. También tenemos dos habitaciones, totalmente libres, pero una con orientación 1 y otra con orientación 0. Al ejecutarlo vemos que la habitación con orientación 0 se queda sin reservas.

Extensión 3

En este caso, queremos observar como se reduce el desperdicio, así que creamos dos habitaciones, una con capacidad para 2 personas y otra para 3. También creamos dos reservas que no se solapan entre ellas, una con 2 personas y otra con 3. Hay que ver como encajan las reservas con las habitaciones. La reserva de 2 personas nunca irá a la habitación de 3.

Extensión 4

Este problema es igual que el anterior, pero queremos ver como da un resultado diferente. ¿Por qué tendría que dar un resultado diferente? En la extension 3 se abrían 2 habitaciones para tener desperdicio 0. Ahora preferimos no abrir una

habitación a tener desperdicio. Así pues, tras ejecutar vemos que la reserva de 2 personas acaba en la habitación de 3 plazas. La habitación de 2 plazas no se usa nunca.