Documentación de la práctica de Planificación

Laboratorio de Inteligencia Artificial
Curso 2017/2018

Guillem Rodríguez Corominas
David Moreno Borràs
Marco Soldan

Índice general

1. Introducción

Para esta tercera práctica, hemos resuelto un problema mediante un planificador implementado mediante el lenguaje PDDL.

Este sistema permitirá, a partir de un estado inicial en recomendar una serie de pasos que nos llevaran a la solución deseada (si existe el camino)

2. Identificación

La base del problema es asignar habitaciones a reservas hechas por personas teniendo en cuenta dos factores principales: los dias de la reserva y la capacidad de la habitación

2.1 Dominio

Hemos planteado el dominio con dos predicados principales:

(asignada ?res - reserva ?hab - habitacion)
(atendida ?res - reserva)

El predicado (atendida ?res - reserva) puede parecer redundante porque si tenemos (asignada ?res - reserva ?hab - habitacion) entonces la reserva ya está atendida, pero es útil para cuando en el goal queramos especificar (forall (?res - reserva) (atendida ?res)).

También tenemos las siguientes funciones:

```
(:functions
        (pers_hab ?hab - habitacion)
        (pers_res ?res - reserva)
        (dia_ini ?res - reserva)
        (dia_fin ?res - reserva)
    )
```

Que se usaran para los valores métricos, es decir (en el caso base): el número de personas de una reserva/habitación y el día de inicio y fin de una reserva.

Para acabar tenemos la acción **asignar_habitacion**, la idea es asignar una habitación a una reserva:

- Si la habitación no la usa nadie
- Si la habitación la usa alguien pero cuando la usa no se solapa con la reserva que consideramos actualmente. Para esto hay que hacer un forall con todos los predicados (asignada ?res - reserva ?hab - habitacion) que tenga esa habitación.

Adiciones extensión 1

Para la primera extensión añadimos un valor **(puntuacion)** que se incrementa cada vez que una reserva es atendida. Esto lo hacemos porque en esta versión queremos optimizar las reservas y usando fluents podemos decir que a parte del goal lo maximice: **(:metric maximize (puntuacion))**

También añadimos una nueva acción:

```
(:action no_atender_reserva
        :parameters (?res - reserva)
        :precondition (not (atendida ?res))
        :effect (and (atendida ?res) (increase (puntuacion) 1) )
 )
```

Esto se debe a que el goal del problema ha de ser:

```
(:goal (forall (?res - reserva) (atendida ?res)))
```

Para que el planificador busque la mejor solución así que con no_atender_reserva le damos la opción de no atender una, pero teniendo menos puntuación, así queremos que como atenderla de verdad da más puntuación atienda el máximo posible.

Adiciones extensión 2

En este caso hacemos algo parecido con dos valores: (no_atendidas) y **(bien_orientadas)**. Querremos minimizar no_atendida que se incrementa cada vez que no atendemos una reserva en no_atender_reserva. Esta vez, como teniamos problemas usando “when” tuvimos que hacer 2 funciones: asignar_habitacion y asignar_habitacion_buena, una para cuando la orientación es adecuada y otra para cuando no lo es. Ambas asignan la habitación, pero una da más valor a bien_orientadas, valor que queremos optimizar.

Adiciones extensión 3

En esta extensión hacemos algo parecido a la primera pero esta vez hacemos **(increase (punt_asignadas) (- (pers_hab ?hab) (pers_res ?res)))**, es decir, sumamos la diferencia de personas: el espacio desaprovechado. Lo que haremos con esto luego es decirle al metric que minimize este parámetro.

Adiciones extensión 4

Para este último caso tenemos un nuevo predicado **(utilizada ?hab - habitacion)** que nos indica si una habitación ha sido usada ya o no. Con esto lo que hacemos es tener dos acciones: asignar_habitacion y asignar_habitacion_nueva de forma que ambas asignan una habitación pero la primera lo hace con una habitación que ya esté utilizada y la segunda con una que no. Así hacemos que en asignar_habitacion_nueva aumente un valor (hab_asignadas) (pero no en asignar_habitacion normal) y luego le pedimos al metric que minimice hab_asignadas.

2.2 Problemas

Para hacer los problemas tenemos una serie de objetos como:

```
(:objects
        hab1 - habitacion
        res1 res2 res3 res4 - reserva
)
```

Y el estado inicial, en el que indicamos que las reservas no están atendidas, el número de personas que puede haber en cada habitación y los días y número de personas de cada reserva:

```
(:init
    (not (atendida res1))
    (not (atendida res2))
    (not (atendida res3))
    (not (atendida res4))


        (= (pers_hab hab1) 4)
```

```
        (= (pers_res res1) 3)
        (= (dia_ini res1) 1)
        (= (dia_fin res1) 5)
        (= (pers_res res2) 2)
        (= (dia_ini res2) 6)
        (= (dia_fin res2) 7)
        (= (pers_res res3) 1)
        (= (dia_ini res3) 8)
        (= (dia_fin res3) 10)
        (= (pers_res res4) 3)
        (= (dia_ini res4) 2)
        (= (dia_fin res4) 3)
    )
```

Y finalmente el objetivo: que todas las reservas estén atendidas:

```
    (:goal (forall (?res - reserva) (atendida ?res)))
```

3. Problemas de prueba

Ahora veremos si nuestro sistema funciona como esperábamos mediante una serie de pruebas diseñadas por nosotros.

Nivel Básico

Prueba 1

En esta prueba tendremos reservas y habitaciones que podrán ser asignadas correctamente todas pero una no así que veremos que como se pide para el nivel básico si no se pueden asignar todas las reservas no se asigna ninguna

Entrada

```
(define (problem problemabasico)

    (:domain nivelbasico)
    (:objects
                hab1 - habitacion
                res1 res2 res3 res4 - reserva
    )
    (:init
    (not (atendida res1))
    (not (atendida res2))
    (not (atendida res3))
    (not (atendida res4))


        (= (pers_hab hab1) 4)

        (= (pers_res res1) 3)
        (= (dia_ini res1) 1)
        (= (dia_fin res1) 5)
        (= (pers_res res2) 2)
        (= (dia_ini res2) 6)
        (= (dia_fin res2) 7)
        (= (pers_res res3) 1)
        (= (dia_ini res3) 8)
        (= (dia_fin res3) 10)
        (= (pers_res res4) 3)
        (= (dia_ini res4) 2)
```

```
        (= (dia_fin res4) 3)
    )

    (:goal
        (forall (?res - reserva) (atendida ?res))
    )
```

Como podemos ver tenemos una única habitación y muchas reservas, debería poder asignarlas pero al final la última reserva se solapa con otra.

Salida

best first search space empty! problem proven unsolvable.

Como podemos ver no ha encontrado ninguna solución

Prueba 2

Usaremos exactamente el mismo caso que la primera prueba pero añadiremos una segunda habitación, así la reserva que se solapaba se asignará a esta y todas las demás se podrán asignar a la otra

Entrada

```
(define (problem problemabasico)

    (:domain nivelbasico)
    (:objects
            hab1 - habitacion
            res1 res2 res3 res4 - reserva
    )
    (:init
    (not (atendida res1))
    (not (atendida res2))
    (not (atendida res3))
    (not (atendida res4))


        (= (pers_hab hab1) 4)

        (= (pers_res res1) 3)
        (= (dia_ini res1) 1)
        (= (dia_fin res1) 5)
        (= (pers_res res2) 2)
        (= (dia_ini res2) 6)
        (= (dia_fin res2) 7)
```

```
        (= (pers_res res3) 1)
        (= (dia_ini res3) 8)
        (= (dia_fin res3) 10)
        (= (pers_res res4) 3)
        (= (dia_ini res4) 2)
        (= (dia_fin res4) 3)
    )

    (:goal
        (forall (?res - reserva) (atendida ?res))
    )
```

Salida

```
ff: found legal plan as follows

step    0: ASIGNAR_HABITACION RES4 HAB2
    1: ASIGNAR_HABITACION RES3 HAB2
    2: ASIGNAR_HABITACION RES2 HAB2
    3: ASIGNAR_HABITACION RES1 HAB1
```

Como podemos ver ha podido asignar 4 reservas con solo 2 habitaciones evitando que se solapen.

Extensión 1

Prueba 1

En esta prueba tendremos el mismo caso que antes con una sola habitación y dos reservas que se solapan, pero veremos que, como nos indica el enunciado, pueden quedar reservas sin asignar así que esta vez sí nos dará una solución

Entrada

```
(define (problem jocprova)

    (:domain extension1)
    (:objects
            hab1 - habitacion
            res1 res2 res3 res4 - reserva
    )
    (:init
    (not (atendida res1))
    (not (atendida res2))
```

```
        (not (atendida res3))
        (not (atendida res4))


                (= (pers_hab hab1) 4)

                (= (pers_res res1) 3)
                (= (dia_ini res1) 1)
                (= (dia_fin res1) 5)
                (= (pers_res res2) 2)
                (= (dia_ini res2) 6)
                (= (dia_fin res2) 7)
                (= (pers_res res3) 1)
                (= (dia_ini res3) 8)
                (= (dia_fin res3) 10)
                (= (pers_res res4) 3)
                (= (dia_ini res4) 2)
                (= (dia_fin res4) 3)

                (= (puntuacion) 0)
        )
        (:goal
                (forall (?res - reserva) (atendida ?res))
        )

        (:metric maximize (puntuacion))
)
```

Salida

```
ff: found legal plan as follows

step    0: ASIGNAR_HABITACION RES4 HAB1
        1: ASIGNAR_HABITACION RES3 HAB1
        2: ASIGNAR_HABITACION RES2 HAB1
        3: NO_ATENDER_RESERVA RES1
```

Como podemos ver descarta una de las reservas para poder asignar las demás a pesar de tener solo una habitación

Prueba 2

En esta prueba vamos a ver que no asignamos una habitación si la reserva tiene más personas de la personas de la habitación .

Entrada

```
(define (problem problemaextension1)

	(:domain extension1)
	(:objects
			hab1 hab2 - habitacion
			res1 res2 res3 - reserva
	)
	(:init
	(not (atendida res1))
	(not (atendida res2))
	(not (atendida res3))

		(= (pers_hab hab1) 4)
		(= (pers_hab hab2) 3)

		(= (pers_res res1) 3)
		(= (dia_ini res1) 1)
		(= (dia_fin res1) 5)

		(= (pers_res res2) 3)
		(= (dia_ini res2) 1)
		(= (dia_fin res2) 5)

		(= (pers_res res3) 5)
		(= (dia_ini res3) 6)
		(= (dia_fin res3) 7)

		(= (puntuacion) 0)
	)

	(:goal
		(forall (?res - reserva) (atendida ?res))
	)
	(:metric maximize (puntuacion))
)
```

Salida

ff: found legal plan as follows

step 0: NO_ATENDER_RESERVA RES3
1: ASIGNAR_HABITACION RES2 HAB2
2: ASIGNAR_HABITACION RES1 HAB1

Como podemos ver la tercera reserva no se cumple (es por 5 personas).

Extensión 2

Prueba 1

En esta prueba queremos demostrar que el programa no quita una reserva solo porque no tenga la orientación adecuada. Vamos a poner 3 reservas en días diferentes por el lado norte y una en medio por el lado sur en un hotel que tiene solo habitaciones al lado norte.

Entrada

(define (problem problemaextension2)

(:domain extension2)
(:objects
hab1 hab2 - habitacion
res1 res2 res3 res4 - reserva
)
(:init
(not (atendida res1))
(not (atendida res2))
(not (atendida res3))
(not (atendida res4))

(= (pers_hab hab1) 4)
(= (orientacio hab1) 1)

(= (pers_hab hab2) 4)
(= (orientacio hab2) 1)

(= (pers_res res1) 3)
(= (dia_ini res1) 1)
(= (dia_fin res1) 5)

```
            (= (orientacio res1) 1)

            (= (pers_res res2) 2)
            (= (dia_ini res2) 6)
            (= (dia_fin res2) 7)
            (= (orientacio res2) 1)

            (= (pers_res res3) 1)
            (= (dia_ini res3) 8)
            (= (dia_fin res3) 10)
            (= (orientacio res3) 1)

            (= (pers_res res4) 3)
            (= (dia_ini res4) 4)
            (= (dia_fin res4) 7)
            (= (orientacio res4) 0)

            (= (bien_orientadas) 0)
            (= (no_atendidas) 0)
        )

        (:goal
            (forall (?res - reserva) (atendida ?res))
        )
        (:metric maximize (- (bien_orientadas) (no_atendidas)))
)
```

Salida

```
ff: found legal plan as follows

step    0: ASIGNAR_HABITACION RES4 HAB2
        1: ASIGNAR_HABITACION_BUENA RES3 HAB2
        2: ASIGNAR_HABITACION_BUENA RES2 HAB1
        3: ASIGNAR_HABITACION_BUENA RES1 HAB1
```

Como podemos ver el programa no quita la reserva 4 solo porque no puede tener la orientación corecta.

Prueba 2

Para la segunda prueba queremos demonstrar que a poder ser (si tenimos muchas habitaciones) nuestro código pone todas las reservas de forma que coincidan las orientaciones.

Entrada

```
(define (problem problemaextension2)

        (:domain extension2)
        (:objects
                        hab1 hab2 hab3 hab4 hab5 - habitacion
                        res1 res2 res3 res4 - reserva
        )
        (:init
        (not (atendida res1))
        (not (atendida res2))
        (not (atendida res3))
        (not (atendida res4))

                (= (pers_hab hab1) 4)
                (= (orientacio hab1) 1)

                (= (pers_hab hab2) 4)
                (= (orientacio hab2) 2)

                (= (pers_hab hab3) 4)
                (= (orientacio hab3) 3)

                (= (pers_hab hab4) 4)
                (= (orientacio hab4) 4)

                (= (pers_hab hab5) 4)
                (= (orientacio hab5) 4)

                (= (pers_res res1) 3)
                (= (dia_ini res1) 1)
                (= (dia_fin res1) 5)
                (= (orientacio res1) 1)

                (= (pers_res res2) 3)
                (= (dia_ini res2) 1)
                (= (dia_fin res2) 5)
                (= (orientacio res2) 2)

                (= (pers_res res3) 3)
                (= (dia_ini res3) 1)
                (= (dia_fin res3) 5)
                (= (orientacio res3) 3)
```

```
            (= (pers_res res4) 3)
            (= (dia_ini res4) 1)
            (= (dia_fin res4) 5)
            (= (orientacio res4) 4)

            (= (bien_orientadas) 0)
            (= (no_atendidas) 0)
        )

        (:goal
            (forall (?res - reserva) (atendida ?res))
        )
        (:metric maximize (- (bien_orientadas) (no_atendidas)))
)
```

Salida

```
ff: found legal plan as follows

step    0: ASIGNAR_HABITACION_BUENA RES4 HAB5
        1: ASIGNAR_HABITACION_BUENA RES3 HAB3
        2: ASIGNAR_HABITACION_BUENA RES2 HAB2
        3: ASIGNAR_HABITACION_BUENA RES1 HAB1
```

Como podemos ver el programa va a poner todas la reservas en buenas habitaciones por orientación.

Extensión 3

Prueba 1

En esta prueba tendremos un caso muy restrictivo en el que hay dos habitaciones solo y coge las reservas que desperdician menos espacio.

Entrada

```
(define (problem problemaextension3)

    (:domain extension3)
    (:objects
                hab1 hab2 - habitacion
```

```
                res1 res2 res3 - reserva
    )
    (:init
        (not (atendida res1))
        (not (atendida res2))


        (= (pers_hab hab1) 4)
        (= (pers_hab hab2) 4)

        (= (pers_res res1) 2)
        (= (dia_ini res1) 1)
        (= (dia_fin res1) 4)

        (= (pers_res res2) 4)
        (= (dia_ini res2) 1)
        (= (dia_fin res2) 4)

        (= (pers_res res3) 4)
        (= (dia_ini res3) 1)
        (= (dia_fin res3) 4)


        (= (punt_asignadas) 0)
        (= (no_asignadas) 0)
    )

    (:goal
        (forall (?res - reserva) (atendida ?res))
    )
    (:metric minimize (+ (punt_asignadas) (no_asignadas)))
)
```

Salida

```
step    0: NO_ATENDER_RESERVA RES1
        1: ASIGNAR_HABITACION RES3 HAB1
        2: ASIGNAR_HABITACION RES2 HAB2
```

Como podemos ver asigna la reserva 2 y la 3, que son las que tienen más personas

Prueba 2

Para comprobar que funciona y no ha sido casualidad cogemos el mismo caso que antes pero esta vez hacemos que la reserva 3 tenga menos personas (es decir desaproveche más el espacio)

Entrada

```
(define (problem problemaextension3)

   (:domain extension3)
   (:objects
                hab1 hab2 - habitacion
                res1 res2 res3 - reserva
   )
   (:init
        (not (atendida res1))
        (not (atendida res2))


        (= (pers_hab hab1) 4)
        (= (pers_hab hab2) 4)

        (= (pers_res res1) 4)
        (= (dia_ini res1) 1)
        (= (dia_fin res1) 4)

        (= (pers_res res2) 4)
        (= (dia_ini res2) 1)
        (= (dia_fin res2) 5)

        (= (pers_res res3) 2)
        (= (dia_ini res3) 1)
        (= (dia_fin res3) 4)


        (= (punt_asignadas) 0)
        (= (no_asignadas) 0)
   )

   (:goal
        (forall (?res - reserva) (atendida ?res))
   )
   (:metric minimize (+ (punt_asignadas) (no_asignadas)))
)
```

Salida

step 0: NO_ATENDER_RESERVA RES3
1: ASIGNAR_HABITACION RES2 HAB1
2: ASIGNAR_HABITACION RES1 HAB2

Extensión 4

Prueba 1

En esta prueba vamos a poner muchas habitaciones y reservas que no se solapen entre una y la otra, y esperamos que se vaya a usar una sola habitación.

Entrada

```
(define (problem problemaextension4)

    (:domain extension4)
    (:objects
            hab1 hab2 hab3 hab4 - habitacion
            res1 res2 res3 res4 - reserva
    )
    (:init
    (not (utilizada hab1))
    (not (utilizada hab2))
    (not (utilizada hab3))

    (not (atendida res1))
    (not (atendida res2))
    (not (atendida res3))
    (not (atendida res4))

        (= (pers_hab hab1) 4)
        (= (pers_hab hab2) 4)
        (= (pers_hab hab3) 3)
        (= (pers_hab hab4) 3)

        (= (pers_res res1) 3)
        (= (dia_ini res1) 1)
        (= (dia_fin res1) 7)

        (= (pers_res res2) 3)
        (= (dia_ini res2) 9)
```

```
            (= (dia_fin res2) 12)

            (= (pers_res res3) 2)
            (= (dia_ini res3) 14)
            (= (dia_fin res3) 17)

            (= (pers_res res4) 2)
            (= (dia_ini res4) 19)
            (= (dia_fin res4) 22)


            (= (punt_asignadas) 0)
            (= (no_asignadas) 0)
            (= (hab_asignadas) 0)
        )

        (:goal
            (forall (?res - reserva) (atendida ?res))
        )
        (:metric minimize (+ (punt_asignadas) (+ (* (no_asignadas) 60) (* (hab_asignadas)
30))))
)
```

Salida

```
ff: found legal plan as follows

step    0: ASIGNAR_HABITACION_NUEVA RES4 HAB3
        1: ASIGNAR_HABITACION RES3 HAB3
        2: ASIGNAR_HABITACION RES1 HAB3
        3: ASIGNAR_HABITACION RES2 HAB3
```

Prueba 2

Vamos a poner varios días que se solapen ahora y esperamos que tambien vaya a usar las mismas habitaciones.

Entrada

```
(define (problem problemaextension4)

        (:domain extension4)
```

```
(:objects
                hab1 hab2 hab3 hab4 - habitacion
                res1 res2 res3 res4 res5 res6 - reserva
)
(:init
(not (utilizada hab1))
(not (utilizada hab2))
(not (utilizada hab3))
(not (utilizada hab4))

(not (atendida res1))
(not (atendida res2))
(not (atendida res3))
(not (atendida res4))

        (= (pers_hab hab1) 4)
        (= (pers_hab hab2) 4)
        (= (pers_hab hab3) 3)
        (= (pers_hab hab4) 3)

        (= (pers_res res1) 3)
        (= (dia_ini res1) 1)
        (= (dia_fin res1) 7)

        (= (pers_res res2) 3)
        (= (dia_ini res2) 9)
        (= (dia_fin res2) 12)

        (= (pers_res res3) 2)
        (= (dia_ini res3) 14)
        (= (dia_fin res3) 17)

        (= (pers_res res4) 2)
        (= (dia_ini res4) 19)
        (= (dia_fin res4) 22)

        (= (pers_res res5) 4)
        (= (dia_ini res5) 9)
        (= (dia_fin res5) 12)

        (= (pers_res res6) 1)
        (= (dia_ini res6) 19)
        (= (dia_fin res6) 22)

        (= (punt_asignadas) 0)
        (= (no_asignadas) 0)
```

```
            (= (hab_asignadas) 0)
        )

        (:goal
            (forall (?res - reserva) (atendida ?res))
        )
        (:metric minimize (+ (punt_asignadas) (+ (* (no_asignadas) 60) (* (hab_asignadas) 30))))
)
```

Salida

```
ff: found legal plan as follows

step    0: ASIGNAR_HABITACION_NUEVA RES3 HAB2
        1: ASIGNAR_HABITACION_NUEVA RES4 HAB3
        2: ASIGNAR_HABITACION RES1 HAB3
        3: ASIGNAR_HABITACION RES2 HAB3
        4: ASIGNAR_HABITACION RES5 HAB2
        5: ASIGNAR_HABITACION RES6 HAB2
```

Como podemos ver de esta manera se van a usar 2 habitaciones en total, que es el mínimo posible. Las habitaciones no van a ser siempre utilizadas de manera óptima porque tenemos que tener en cuenta los 2 beneficios separados.