IA-Practica 3

Planificación

Grupo: Sergi Avila Sanguesa, Francisco Sánchez Costas, Raul Montoya Pérez

ÍNDICE

1. El problema

Una central de reservas de hotel nos pide que implementemos un sistema que sea capaz de asignar las peticiones de reserva que se reciben a las habitaciones del hotel.

Cada habitación se describe por el número que la identifica y por el número de personas que puede alojar. Una habitación puede alojar desde una persona hasta un máximo de cuatro.

Cada reserva se describe también por el número que la identifica, por el número de personas que hacen la reserva, desde una persona hasta un máximo de cuatro, por el dia de inicio de la reserva y por el dia de finalización de ésta.

Como dato importante, solo se considera que las reservas son para un mes específico, el cual, tiene 30 días.

2. Dominio

Variables

- habitacion: tipo que indica que el objeto es una habitación.
- reserva: tipo que indica que el objeto es una reserva.

Funciones

- (idhab ?x - habitacion): contiene el número identificador de una habitación.
- (capacidad ?x - habitacion): contiene la capacidad de una habitación.
- (orienthab ?x - habitacion): contiene la orientación de una habitación.
- (idres ?x - reserva): contiene el número identificador de una reserva.
- (npers ?x - reserva): contiene el número de personas de una reserva.
- (dini ?x - reserva): contiene el día de inicio de una reserva.
- (dfi ?x - reserva): contiene el día de finalización de una reserva.
- (orientres ?x - reserva): contiene el número identificador de una reserva.
- (coste): contiene un valor para controlar el coste de las reservas asignadas.
- (nplazas): contiene un valor para controlar el número de plazas sin usar de las habitaciones.
- (nhab): contiene un valor para controlar el número de habitaciones que se ocupan.

Predicados

- (assignacion ?x - reserva ?y - habitacion): indica que la reserva x se ha asignado a la habitación y.
- (pendiente ?x - reserva): indica si una reserva está pendiente para asignar.
- (pendientehab ?x - habitacion): indica si una habitación está pendiente para asignar.

Acciones

- descarta(x): si una reserva x está pendiente de asignar, se marca como no pendiente y se actualizan los valores de las funciones coste.
- asigna(x, y): si una reserva x está pendiente de asignar, el número de personas de la reserva es menor o igual que la capacidad de la habitación y, y la reserva no se solapa con ninguna otra reserva que ya esté asignada a la habitación, se asigna la reserva x a la habitación y, se marca la reserva x como no pendiente y se actualizan los valores de coste, nplazas y nhab.

3. Modelización

En este apartado se describen los elementos que han aparecido en las distintas extensiones del problema. En cada una de las extensiones se han ido añadiendo nuevas funcionalidades a partir de otra extensión, por lo que solo se describirán aquellos nuevos cambios que han surgido respecto la versión anterior.

Nivel básico

En este escenario tenemos como tipos las reservas y las habitaciones.

Se utilizan las siguientes funciones:

- idhab, para saber el identificador de una habitación.
- capacidad, para saber la capacidad de una habitación.
- idres, para saber el identificador de una reserva.
- npers, para saber el número de personas de una reserva.
- dini, para saber el dia inicial de una reserva.
- dfi, para saber el dia de finalización de una reserva.

En cuanto a predicados se usan asignacion(x,y) para indicar que la reserva x se ha asignado a la habitación y, y pendiente(x) que indica si una reserva está pendiente para asignar.

Finalmente, tenemos una acción asigna (x, y) que asigna una reserva a una habitación si la reserva estaba pendiente por asignar y el número de personas de la reserva es menor que la capacidad de la habitación. El efecto será la creación de la asignación y la reserva x pasará a estar no pendiente.

Extensión 1

Esta extensión parte del escenario anterior. Se añade una función coste que se utilizará para controlar el número de reservas asignadas. También se añade una nueva acción descarta (x) que dada una reserva x que esté pendiente por asignar, la marcará como no pendiente y incrementará el valor de la función coste según el número de días de la reserva elevado al cubo. También se añadirá un efecto a la acción asigna del escenario básico el cual será incrementar el valor de la función coste según el número de días de la reserva al cuadrado, de ésta forma siempre que pueda asignar una reserva a una habitación lo hará antes de descartarla debido a que el coste siempre será menor al asignar.

Extensión 2

Esta extensión parte del escenario anterior. Se añaden dos funciones nuevas orientahab (x) y orientres (x) cuyo valor contiene la orientación de una habitación y una reserva respectivamente. Se modifica la acción asigna (x, y) para incorporar en el efecto de ésta que si la orientación de la reserva x no se corresponde con la orientación de la habitación y, se añada a la función coste el número de días de la reserva. La penalización por orientación incorrecta es lineal respecto al número de días, y por lo tanto, es mejor realizar una reserva, aunque no coincida la orientación, antes que no asignarla a ninguna habitación.

Extensión 3

Partiendo del escenario de la extensión 1, se añade una nueva función nplazas que se utilizará para controlar el número de plazas que se desperdician al hacer una asignación de una reserva a una habitación del hotel según el número de personas de la reserva y la capacidad de la habitación. Se crea una nueva acción descarta(x, y) que eliminará la asignación de la reserva x a la habitación y si existe ésa asignación y la reserva x no está pendiente para asignar. Se modificará también la acción asigna (x, y) para incorporar en el efecto el aumento de nplazas según las plazas desperdiciadas en esa asignación.

Extensión 4

Saliendo de la extensión 3, añadimos una función nhab, que controlará el número de habitaciones diferentes que se asignan para todas las reservas. Creamos una nueva acción valorhab(x), que revisará todas las habitaciones que hayan quedado pendientes de reserva. El efecto que produce es que incrementa por cada habitación pendiente el valor de la función numhab en 1, para llevar la cuenta de cada una de las habitaciones que quedaban por asignar, además de dejar de poner pendiente cada habitación que contemos por motivos de control. Esto se hace como medida para prevenir modificar otras acciones.

4. Instancias del problema

Los diferentes objetos que hay en el problema son las reservas y las habitaciones del hotel. La entrada consiste en una cantidad determinada de habitaciones, donde se introduce el identificador y la capacidad de cada una, y de reservas, donde se introduce el identificador, número de personas, dia de inicio y dia de finalización de ésta. Además para cada habitación se marca como pendiente para asignar, así como para cada reserva se marca también como pendiente para asignar. Finalmente se inicializa a 0 los valores de las funciones coste, nplazas y nhab. Para la extensión 2 también se indicara la orientación de cada reserva y de cada habitación.

Nivel básico

Tenemos los objetos de habitaciones, con el identificador y la capacidad de cada una, y de reservas, donde se introduce el identificador, número de personas, dia de inicio y dia de finalización de ésta. Además tenemos una función pendiente para cada reserva que indica que cada una de las reservas está pendiente para asignar. Finalmente se alcanzará el goal cuando todas las las reservas estén asignadas, es decir, que todas las reservas no estén pendientes para asignar (goal (forall (?x - reserva)(not (pendiente ?x)))), de lo contrario no se realizará ninguna asignación.

Extensión 1

En esta extensión se añade una función coste que se inicializará a 0 y se utilizará para controlar el número de reservas asignadas. El valor de la función incrementará cuando se asigne una reserva a una habitación en función de los días que dure de la reserva. En este escenario el goal será el mismo que el anterior (goal (forall (?x - reserva)(not (pendiente ?x)))) pero además se minimizará el valor de la función coste (:metric minimize (coste)).

Extensión 2

La unica diferencia con la extension 1 es que ahora las habitaciones y reservas también se identifican por la orientación. El goal y metric es el mismo, pues la penalizacion por orientación incorrecta se hace directamente en el coste.

Extensión 3

Esta extensión parte del escenario de la 1a, la que utiliza la función coste para controlar el número de reservas asignadas. Además de lo que se había hecho para la 1a extensión, se añade otra función nplazas inicializada a 0, que se utilizará para controlar las plazas que se desperdician en las habitaciones del hotel al asignar una reserva a ésta. Se aumentará el valor de ésta función cuando se asigne una reserva a una habitación o se disminuirá

cuando se desasigne en función del número de personas de la reserva y la capacidad de la habitación. En cuanto al goal será el mismo que en la extensión 1 (goal (forall (?x - reserva)(not (pendiente ?x)))) pero se minimizará la combinación de las funciones coste y nplazas (:metric minimize (+(* 10 (nplazas))(coste))).

Extensión 4

Partimos de la 3era extensión, usando esta vez la función numhab inicializada a 0 para controlar, como ya se ha indicado anteriormente, el número de habitaciones. El goal es el mismo que el anterior, pero con un pequeño cambio a la métrica para poder ahora minimizar el número de habitaciones diferentes:
(:metric minimize (+(* 10 (nplazas)) (* 5 (nhab)) (coste)))

5. Desarrollo de los problemas

Para desarrollar los diferentes problemas o modelos hemos seguido el guión de la práctica y nos hemos basado en un diseño incremental. Primero empezamos por el problema básico y luego fuimos ampliando hasta lo que se nos pedía en la extensión 1, hasta la extensión 4, que incluímos también todo aquello que hicimos en la extensión 3 (que a su vez incluye la 1). En cada dominio se han ido añadiendo nuevos elementos, hasta en el dominio de la extensión 4. Las diferentes extensiones son parecidas entre sí, sin embargo, pasar del dominio básico a la primera extensión fue un proceso más largo que desarrollar el básico, debido a tener que pensar en la mejor forma de incorporar la métrica en el planificador. Al final, vimos que el mejor equilibrio entre eficiencia y calidad de la solución era añadiendo la acción descartar y confiar en que el planificador haría un buen trabajo minimizando el coste.

6. Juegos de prueba

Se ha hecho un juego de prueba para cada extensión menos para la extensión 1, ya que en las extensiones 3 y 4 ya se prueban esta extensión también.

Nivel básico

Dos juegos de pruebas, uno con un solo solapamiento y dos habitaciones para que se puedan asignar las reservas que se solapan a habitaciones distintas y otro con dos habitaciones también pero 3 reservas solapadas entre sí para que no se asigne ninguna.

Input:

```
(define (problem basic)
  (:domain hotel)
  (:objects r1 r2 r3 r4 r5 - reserva
        h1 h2 - habitacion
  )
  (:init
        ;habitaciones
        (= (idhab h1) 1)
        (= (idhab h2) 2)
        (= (capacidad h1) 2)
        (= (capacidad h2) 4)
        ;reservas
        (= (idres r1) 1)
        (= (idres r2) 2)
        (= (idres r3) 3)
        (= (idres r4) 4)
        (= (idres r5) 5)

        (= (npers r1) 2)
        (= (npers r2) 1)
        (= (npers r3) 3)
        (= (npers r4) 4)
        (= (npers r5) 1)

        (= (dini r1) 1)
        (= (dfi r1) 20)

        (= (dini r2) 12)
        (= (dfi r2) 13)
```

```
        (= (dini r3) 14)
        (= (dfi r3) 15)

        (= (dini r4) 20)
        (= (dfi r4) 25)

        (= (dini r5) 26)
        (= (dfi r5) 30)
        ;;
        (pendiente r1)
        (pendiente r2)
        (pendiente r3)
        (pendiente r4)
        (pendiente r5)

  )
  (:goal
        (forall (?x - reserva )
        (not (pendiente ?x))
        )
  )
)
```

Output:

```
step    0: ASIGNA R5 H2
        1: ASIGNA R4 H2
        2: ASIGNA R3 H2
        3: ASIGNA R2 H2
        4: ASIGNA R1 H1
```

Justificación: Con un solo solapamiento de las reservas. Se espera que se asignen todas y las que se solapan las asigne en habitaciones distintas.

Input:

```
(define (problem basic)
  (:domain hotel)
  (:objects r1 r2 r3 r4 r5 - reserva
        h1 h2 - habitacion
  )
  (:init
        ;habitaciones
        (= (idhab h1) 1)
        (= (idhab h2) 2)
        (= (capacidad h1) 2)
```

```
        (= (capacidad h2) 4)
        ;reservas
        (= (idres r1) 1)
        (= (idres r2) 2)
        (= (idres r3) 3)
        (= (idres r4) 4)
        (= (idres r5) 5)

        (= (npers r1) 2)
        (= (npers r2) 1)
        (= (npers r3) 3)
        (= (npers r4) 4)
        (= (npers r5) 1)

        (= (dini r1) 1)
        (= (dfi r1) 10)

        (= (dini r2) 12)
        (= (dfi r2) 15)

        (= (dini r3) 14)
        (= (dfi r3) 15)

        (= (dini r4) 20)
        (= (dfi r4) 25)

        (= (dini r5) 26)
        (= (dfi r5) 30)
        ;;
        (pendiente r1)
        (pendiente r2)
        (pendiente r3)
        (pendiente r4)
        (pendiente r5)

  )
  (:goal
        (forall (?x - reserva )
        (not (pendiente ?x))
        )
  )
)
```

Output:
best first search space empty! problem proven unsolvable.

Justificación: Con tres reservas solapadas entre sí. Se espera que no se asigne ninguna ya que al haber tres reservas solapadas y solo dos habitaciones disponibles no las pueda asignar.

Extensión 2

Input

```
(define (problem basic)
 (:domain hotel)  (:objects r1 r2 - reserva
        h1 - habitacion
 )
 (:init
        (= (idhab h1) 1)
        (= (capacidad h1) 1)
        (= (orienthab h1) 4)

        (pendiente r1)
        (= (idres r1) 1)
        (= (npers r1) 1)
        (= (dini r1) 2)
        (= (dfi r1) 12)
        (= (orientres r1) 3)

        (pendiente r2)
        (= (idres r2) 2)
        (= (npers r2) 1)
        (= (dini r2) 2)
        (= (dfi r2) 12)
        (= (orientres r2) 4)

        (= (coste) 0)
        )
 (:goal
        (forall (?x - reserva )
        (not(pendiente ?x))
        )
 )
 (:metric minimize (coste))
)
```

Output:

```
step    0: ASIGNA R2 H1
        1: DESCARTA R1
```

Justificación: En este juego de pruebas sencillo se comprueba el funcionamiento de la parte específica de la extensión. El planificador tiene que escoger entre 2 reservas iguales

pero una con una orientación que coincide con la habitación y otra que no. Efectivamente, escoge la que coincide

Extensión 3

Input:

```
(define (problem basic)
  (:domain hotel)
  (:objects r1 r2 r3 r4 r5 - reserva
        h1  - habitacion
  )
  (:init
        ;habitaciones
        (= (idhab h1) 1)

        (= (capacidad h1) 4)

        ;reservas
        (= (idres r1) 1)
        (= (idres r2) 2)
        (= (idres r3) 3)
        (= (idres r4) 4)
        (= (idres r5) 5)

        (= (npers r1) 1)
        (= (npers r2) 2)
        (= (npers r3) 2)
        (= (npers r4) 1)
        (= (npers r5) 3)

        (= (dini r1) 1)
        (= (dfi r1) 10)

        (= (dini r2) 11)
        (= (dfi r2) 21)

        (= (dini r3) 1)
        (= (dfi r3) 10)

        (= (dini r4) 26)
        (= (dfi r4) 28)

        (= (dini r5) 28)
        (= (dfi r5) 29)
        ;;
```

```
        (pendiente r1)
        (pendiente r2)
        (pendiente r3)
        (pendiente r4)
        (pendiente r5)
        ;
        (= (coste) 0)
        (= (nplazas) 0)
 )
 (:goal
        (forall (?x - reserva )
        (not (pendiente ?x))
        )
 )
 (:metric minimize (+(* 10 (nplazas))(coste)))
)
```

Output:

```
step    0: ASIGNA R5 H1
        1: DESCARTA R4
        2: ASIGNA R3 H1
        3: ASIGNA R2 H1
        4: DESCARTA R1
```

Justificación: En este juego de prueba se introducen cinco reservas y una sola habitación de capacidad mayor al número de personas de cualquier reserva. Hay dos pares de reservas que se solapan, R1-R3 y R4-R5. De estos pares se espera que se asignen R3 y R5 ya que el número de personas de éstas reservas se aproxima más a la capacidad de la habitación que no R1 y R4 respectivamente. De esta forma, se prueba que no se solapen las reservas y además que se aplique el criterio del desperdicio de plazas.

Extensión 4

Input:

```
(define (problem basic)
 (:domain hotel)  (:objects r1 r2 r3 r4 r5 - reserva
        h1 h2 h3 h4 h5 - habitacion
 )
 (:init
        (= (idhab h1) 1)
        (= (capacidad h1) 4)
        (= (idhab h2) 2)
        (= (capacidad h2) 4)
        (= (idhab h3) 3)
        (= (capacidad h3) 4)
        (= (idhab h4) 4)
```

```
        (= (capacidad h4) 4)
        (= (idhab h5) 5)
        (= (capacidad h5) 3)

        (pendiente r1)
        (= (idres r1) 1)
        (= (npers r1) 1)
        (= (dini r1) 19)
        (= (dfi r1) 23)

        (pendiente r2)
        (= (idres r2) 2)
        (= (npers r2) 1)
        (= (dini r2) 20)
        (= (dfi r2) 26)

        (pendiente r3)
        (= (idres r3) 3)
        (= (npers r3) 3)
        (= (dini r3) 2)
        (= (dfi r3) 8)

        (pendiente r4)
        (= (idres r4) 4)
        (= (npers r4) 3)
        (= (dini r4) 24)
        (= (dfi r4) 26)

        (pendiente r5)
        (= (idres r5) 5)
        (= (npers r5) 3)
        (= (dini r5) 24)
        (= (dfi r5) 26)

        (= (coste) 0)
        (= (nplazas) 0)
        (= (nhab) 0)
        )
  (:goal
        (forall (?x - reserva )
        (not(pendiente ?x))
        )
  )
  (:metric minimize (+ (+(* 10 (nplazas)) (* 100 (nhab)) )(coste)) )
)
```

Output:

```
step    0: ASIGNA R5 H5
        1: ASIGNA R4 H4
        2: ASIGNA R3 H5
        3: ASIGNA R2 H3
        4: ASIGNA R1 H5
```

Justificación: En esta ejecución se minimiza en número de habitaciones utilizados. Hay bastantes solapamientos, pero aun asi solo se utilizan 3 de las 5 habitaciones.

Input:

```
(define (problem basic)
  (:domain hotel)  (:objects r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 - reserva
        h1 h2 h3 h4 h5 - habitacion
  )
  (:init
        (= (idhab h1) 1)
        (= (capacidad h1) 4)
        (= (idhab h2) 2)
        (= (capacidad h2) 4)
        (= (idhab h3) 3)
        (= (capacidad h3) 4)
        (= (idhab h4) 4)
        (= (capacidad h4) 4)
        (= (idhab h5) 5)
        (= (capacidad h5) 3)
        (pendiente r1)
        (= (idres r1) 1)
        (= (npers r1) 1)
        (= (dini r1) 1)
        (= (dfi r1) 10)
        (pendiente r2)
        (= (idres r2) 2)
        (= (npers r2) 1)
        (= (dini r2) 11)
        (= (dfi r2) 21)
        (pendiente r3)
        (= (idres r3) 3)
        (= (npers r3) 3)
        (= (dini r3) 1)
        (= (dfi r3) 10)
        (pendiente r4)
        (= (idres r4) 4)
        (= (npers r4) 3)
```

```
        (= (dini r4) 10)
        (= (dfi r4) 15)
        (pendiente r5)
        (= (idres r5) 5)
        (= (npers r5) 3)
        (= (dini r5) 11)
        (= (dfi r5) 17)
        (pendiente r6)
        (= (idres r6) 6)
        (= (npers r6) 3)
        (= (dini r6) 28)
        (= (dfi r6) 29)
        (pendiente r7)
        (= (idres r7) 7)
        (= (npers r7) 3)
        (= (dini r7) 1)
        (= (dfi r7) 20)
        (pendiente r8)
        (= (idres r8) 8)
        (= (npers r8) 3)
        (= (dini r8) 26)
        (= (dfi r8) 28)
        (pendiente r9)
        (= (idres r9) 9)
        (= (npers r9) 3)
        (= (dini r9) 28)
        (= (dfi r9) 29)
        (pendiente r10)
        (= (idres r10) 10)
        (= (npers r10) 3)
        (= (dini r10) 28)
        (= (dfi r10) 29)

        (= (coste) 0)
        (= (nplazas) 0)
        (= (nhab) 0)
        )
  (:goal
        (forall (?x - reserva )
        (not(pendiente ?x))
        )
  )
  (:metric minimize (+ (+(* 10 (nplazas)) (* 100 (nhab)) )(coste)) )
)
```

Output:

step 0: ASIGNA R10 H5
1: ASIGNA R9 H4
2: ASIGNA R8 H3
3: ASIGNA R7 H5
4: ASIGNA R6 H2
5: ASIGNA R5 H4
6: ASIGNA R4 H3
7: ASIGNA R3 H4
8: ASIGNA R2 H2
9: ASIGNA R1 H2

Justificación: En esta ejecución se minimiza en número de habitaciones utilizados. Hay bastantes solapamientos, pero con 4 habitaciones tendría que ser el mínimo, por lo que se espera que se coloquen en 4 habitaciones y no en más..

7. Conclusiones

Como conclusión, hemos resuelto un problema de planificación a partir de un modelo usando el mismo sistema de planificación para resolverlo de forma automática. Hemos analizado un problema de asignación de reservas a habitaciones de un hotel y hemos modelizado los elementos a partir de objetos, funciones, predicados y acciones propias de PDDL. Para completar la práctica se ha ido desarrollando a partir de las extensiones propuestas en el guión. Para solucionar las diferentes instancias se ha usado Fast Forward y se ha podido observar que este tipo de sistemas son capaces de ofrecer una buena planificación en un tiempo de ejecución razonable.