3a práctica de IA
Planificación

Planificación de las reservas de habitaciones de un hotel

Joan Allés, Pau Bosch, Jiabo Wang
Cuatrimestre primavera 2021

Índice

1. El problema

Una central de reservas de un hotel nos pide un sistema capaz de asignar las peticiones de reserva que se reciben a habitaciones, siguiendo diferentes criterios y restricciones.

La central de reservas nos indica que las habitaciones están descritas por su identificador y el número de personas que puede alojar. Y que las reservas vienen descritas por su identificador, el número de personas en la reserva y los días de inicio y final de la reserva.

Los criterios y restricciones que se tienen en cuenta para asignar las reservas son:
En la habitación tiene que tener una capacidad igual o superior al número de personas por las que se hace la reserva, no puede haber dos reservas que se solapan en una misma habitación. Además se intentan optimizar las asignaciones intentando que queden el mínimo de reservas sin asignar, que de las asignadas se logre tener el máximo número de reservas que su orientación preferente y la orientación de la habitación coincidan. Que se minimicen el número de plazas desperdiciadas. Y que se optimice el número de habitaciones ocupadas durante el mes, asignando el menor número de habitaciones diferentes.
Dadas las características del problema, se puede deducir que es claramente un problema resoluble mediante planificación y, por tanto, hemos utilizado PDDL para la resolución.

2. Dominio y problema

2.1 Nivel Básico

En el nivel básico se obtiene una asignación de reservas a habitaciones en las que caben las personas de la reserva (la habitación puede ser más grande) y no hay solapamiento en las ocupaciones. Si no se pueden asignar todas las reservas no se asigna ninguna.

El dominio del nivel básico está formado por:

En este primer nivel utilizamos los **requerimientos** adl, typing y fluents.

Para la resolución del problema declaramos dos tipos de **variables** reserva y habitacion, ambos heredan de objeto.

Las **funciones** que usamos son capacidad, personas, dia_inicio, dia_final:

- (capacidad ?h - habitacion) -> Permite definir la capacidad máxima de la habitación h
- (personas ?r - reserva) -> Permite definir el número de personas para las que se hace la reserva r
- (dia_inicio ?r - reserva) -> Permite definir el día de inicio de la reserva r
- (dia_final ?r - reserva) -> Permite definir el día de fin de la reserva r

También fueron necesarios los **predicados** servido y asignado:

- (servido ?r - reserva) -> El predicado es cierto si a la reserva ya se le ha asignado una habitación.
- (asignado ?r - reserva ?h - habitacion) -> Si la reserva r está servida y ha sido asignada a la habitación h entonces el predicado será cierto.

En esta primera versión solo ha sido necesaria una **acción** que es asignar_habitacion, que recibe una reserva x y una habitación y como parámetros y si x aún no ha sido servida, la habitación y tiene una capacidad suficiente y no se solapa con ninguna otra reserva que también esté adjudicada a la habitación y. Entonces marca la reserva x como servida y crea el predicado (asignado ?x ?y), dejando así constancia de que la reserva x ha sido asignada a la habitación y.

Con esta definición del dominio es suficiente para resolver el nivel básico ya que si se pueden satisfacer todas las reservas la ejecución nos retornara donde tiene que ir cada reserva, y si no se pueden satisfacer todas nos saldrá que no se puede resolver el problema.

El **problema** del nivel básico está compuesto por:

- Un conjunto de **objetos** de reserva y habitacion.
- Se parte de un **estado inicial** con la definición de la capacidad de la habitación y de las personas y días iniciales y finales de las reservas.
- El **estado final** o objetivo es servir todas las reservas, y si no se puede, no servir ninguna.

2.2 Extensión 1

Esta es la primera extensión del nivel básico, y básicamente se quiere obtener una asignación de reservas a habitaciones en las que caben las personas de la reserva (la habitación puede ser más grande) y no hay solapamiento en las ocupaciones, como en el nivel básico, pero aquí a diferencia del nivel básico se intenta optimizar el número de reservas asignadas, pudiendo dejar reservas sin asignar (cosa que no se podía hacer en el nivel básico).

El **dominio** de esta extensión está formado por:

Al igual que el primer nivel, se utilizan adl, typing y fluents como **requerimientos**. Y se utilizan las mismas **variables**: reserva y habitacion.

Las **funciones** que se usan son capacidad, personas, dia_inicio, dia_final (al igual que en el nivel básico), pero se añade una de nueva, que es comptador, que sirve para indicar el número de reservas que no quedan servidas.

En el apartado de los **predicados**, a parte de servido y asignado, se añadió uno de nuevo, que es imposible:

- (imposible ?r - reserva) -> El predicado es cierto si la reserva es imposible de asignar, ya sea porque no queda ninguna habitación que tenga la capacidad para alojar el número de personas de la reserva, o porque no queda ninguna habitación libre en el período de la reserva.

Para esta primera extensión, ha sido necesario añadir una nueva **acción** a parte de asignar_habitacion del nivel básico, que es asignar_imposibilidad_reserva. Esta acción recibe una reserva x, comprueba como precondición que no esté servida y para todas las habitaciones su capacidad es mayor o igual que el número de personas de la reserva o está reservada dentro de una franja de días que colapsa con la reserva x. Entonces como efecto de esta acción, se marca la reserva x como imposible de asignar y se incrementa el comptador en una unidad.

El **problema** de esta primera extensión está compuesto por:

- Un conjunto de **objetos** de reserva y habitacion igual que el nivel básico.
- Se parte de un **estado inicial** con la definición de la capacidad de la habitación y de las personas y días iniciales y finales de las reservas y de un comptador que se inicia a 0.
- El **estado final** o objetivo es servir todas las reservas posibles o por lo contrario marcarlas como imposible, y el objetivo es minimizar el número de reservas imposibles, maximizando el número de reservas asignadas.

2.3 Extensión 2

En la extensión 2 se ha modificado la primera extensión para que ahora también se pueda indicar la orientación preferente de la habitación y se optimice asignando el máximo de reservas posibles y que dentro de las asignadas también haya el máximo número de reservas donde la orientación preferente y la de la habitación coincidan.

El **dominio** de esta extensión está formado por:

En este caso los **requerimientos** siguen siendo los mismos que los de la extensión 1, y a las **variables** se ha añadido uno de nuevo llamado orientacion. Así como también se ha creado una nueva **función** comptadorO, que sirve para contar el número de habitaciones que han sido asignadas y no se satisface su preferencia de orientación.

También hay dos nuevos **predicados** que son:

- (orientacionH ?h - habitacion ?o - orientacion) -> Permite definir la orientación de la habitación.
- (orientacionR ?r - reserva ?o - orientacion) -> Permite definir la orientación preferente de la reserva.

Por lo que hace a las **acciones** se conservan las mismas que en la anterior extensión y se ha modificado levemente la acción asignar_habitacion modificando el nombre donde ahora se llama asignar_habitecion_orientacion y donde se ha añadido a su efecto que si se asigna una habitación a una reserva y la orientación de la habitación no coincide con la preferente de la reserva se aumenta en uno el contadorO.

Con este par de modificaciones ya se consigue el nuevo **objetivo o estado final** ya que al intentar minimizar la suma del contador * 10 y el contadorO se asigna el máximo número de reservas posibles y dentro de estas que coincida el número más grande posible de orientaciones de habitación y reserva. Además con la ponderación nos aseguramos que nunca se dejará sin asignar una reserva porque no haya ninguna habitación disponible que cumpla con su preferencia de orientación si esta se puede asignar a una habitación que no cumpla con su preferencia de orientación ya que si se asigna se penaliza sumando 1 mientras que si se deja sin satisfacer se penaliza sumando 10. Tanto **objetos** como el **estado inicial** del problema de esta extensión es la misma que la extensión 1, pero añadiendo orientación a las habitaciones y reservas.

2.4 Extensión 3

En la tercera extensión se ha modificado la primera extensión para que se optimice el número de plazas desperdiciadas. Se considera desperdicio de plazas asignar habitaciones con capacidad superior a las solicitadas en la reserva.

El **dominio** de esta extensión está formado por:

Al igual que el primer nivel, se utilizan adl, typing y fluents como **requerimientos**. Y se utilizan las mismas **variables**: reserva y habitacion.

Las **funciones** que se usan son capacidad, personas, dia_inicio, dia_final, comptador (al igual que en el nivel 1), pero se añade una de nueva, que es comptadorPlaces, que sirve para indicar el número de plazas desperdiciadas con las reservas asignadas hasta el momento.

En el apartado de los **predicados**, se han utilizado los mismos que en el nivel 1 que són servido, asignado, imposible.

Las **acciones** también se conservan las mismos que la extensión 1, pero la acción asginar_habitación ha sido modificada para incrementar el valor del comptadorPlaces con la diferencia entre la capacidad de la habitación asignada y el número de personas solicitadas en la reserva.

El **problema** de esta primera extensión está compuesto por:

- Un conjunto de **objetos** de reserva y habitacion igual que el nivel básico.
- Se parte de un **estado inicial** con la definición de la capacidad de las habitaciones y de las personas y días iniciales y finales de las reservas y de los contadores (comptador, comptadorPlaces) que se inician a 0.
- El **estado final** es servir todas las reservas posibles o por lo contrario marcarlas como imposible
- El **objetivo** es minimizar el número de reservas imposibles y el número de plazas desperdiciadas. Para dar prioridad a las reservas imposibles se han ponderado en 8 las reservas imposibles y en 1 el número de plazas desperdiciadas.

2.5 Extensión 4

En la cuarta extensión se ha modificado la tercera extensión para que además se optimice el número de habitaciones utilizadas. Aunque se tiene que tener en cuenta su optimización esta ha de ser menos significativa que la de asignar todas las reservas y superior a la optimización de plazas desperdiciadas.

El **dominio** de esta extensión está formado por:

Al igual que el primer nivel, se utilizan adl, typing y fluents como **requerimientos**. Y se utilizan las mismas **variables**: reserva y habitacion.

Las **funciones** que se usan son capacidad, personas, dia_inicio, dia_final, comptador, comptadorPlaces (al igual que en el nivel 3), pero se añade una de nueva, que es comptadorHabitacions, que sirve para indicar el número de habitaciones utilizadas en la asignación realizada hasta el momento.

En el apartado de los **predicados**, se han utilizado los mismos que en el nivel 3 que són servido, asignado, imposible. Además se ha añadido un nuevo predicado ocupada:

- (ocupada ?habitacion - habitacion) -> El predicado es cierto si la habitación tiene alguna reserva asignada. Necesitamos tenerla para saber el número de habitaciones utilizadas.

Las **acciones** también se conservan las mismos que la extensión 3, pero la acción asginar_habitación ha sido modificada para contar el número de habitaciones utilizadas y mantener el predicado ocupada coherentemente. En caso de asignar una reserva a una habitación que no esté ocupada el predicado ocupada pasará a ser cierto, y además, el comptadorHabitacions se incrementará en 1.

El **problema** de esta primera extensión está compuesto por:

- Un conjunto de **objetos** de reserva y habitacion igual que el nivel básico.
- Se parte de un **estado inicial** con la definición de la capacidad de las habitaciones y de las personas y días iniciales y finales de las reservas y de los contadores (comptador, comptadorPlaces, comptadorHabitacions) que se inician a 0.
- El **estado final** es servir todas las reservas posibles o por lo contrario marcarlas como imposible

- El **objetivo** es minimizar el número de reservas imposibles, el número de habitaciones utilizadas y el número de plazas desperdiciadas. Para dar prioridad a las reservas imposibles seguido del número de habitaciones se han ponderado en 8 las reservas imposibles, en 4 el número de habitaciones utilizadas y en 1 el número de plazas desperdiciadas.

3. Desarrollo de los modelos

Para poder solucionar el problema y sus optimizaciones optamos por un diseño incremental donde empezamos con un modelo lo más básico posible (el nivel básico) y poco a poco le hemos ido añadiendo funcionalidades para completar las diversas extensiones, siguiendo el guión del enunciado de la práctica.
Para cada extensión hemos desarrollado un modelo PDDL y un generador de entradas, que posteriormente lo hemos utilizado para el desarrollo de los juegos de prueba.

4. Generadores de problemas

Hemos hecho un conjunto de programas en lenguaje C++ que genera aleatoriamente ficheros con juegos de prueba para cada extensión:

- generador0.cc -> para el nivel básico
- generador1.cc -> para la extensión 1
- generador2.cc -> para la extensión 2
- generador3.cc -> para la extensión 3
- generador4.cc -> para la extensión 4

Genera un número de reservas aleatorias (entre 1 - 15) y un número de habitaciones aleatorias (entre 1 - 10), hemos decidido que tenga más reservas que habitaciones, ya que es lo más lógico.

Tanto el número de personas que puede alojar una habitación como el número de personas en la reserva es un número aleatorio entre 1 y 4, tal y como está descrito en el enunciado de la práctica.

Finalmente, los días de inicio y final de una reserva están comprendidos entre 1 y 30, como nos indica el enunciado.

El objetivo de cada extensión puede variar, por lo que hemos diferenciado 5 generadores diferentes. Por ejemplo, el goal del nivel básico es que todas las reservas sean servidas, mientras que en la extensión 1 puede quedar reservas sin ser servidas. Y algunos generadores de problema necesitan metric (el 2, 3 y 4) mientras que el 0 y el 1 no lo necesitan.

Por todo esto, hemos decidido hacer 5 versiones diferentes de generadores de problema, aunque tengan la mayor parte del código similar, hay algunos detalles que les diferencian, por lo que es necesario distinguirlos.

Para compilar y ejecutar cada uno de los generadores hay que utilizar las siguientes comandas:
g++ -o ej generador1.cc (para compilar el generador 1)
./ej (para ejecutarlo)
Al ejecutarlo, se crea un fichero de problema

5. Experimentación

Mediante la experimentación queremos ver como el tiempo de resolución evoluciona con el tamaño del problema. Para llevar a cabo esta experimentación, usamos como referencia la extensión 4 del problema porque consideramos que es la más compleja y es la que engloba casi todas las extensiones anteriores. Hemos usado el generador de problemas generador4.cc para generar los problemas de distintas medidas y lo hemos comparado.

Observación	El tiempo de resolución evoluciona con el tamaño del problema
Planteamiento	Hacer experimentos con problemas de diferentes tamaños y comparar sus tiempos de resolución
Hipótesis	Los problemas con tamaños más grandes van a tener un tiempo de resolución mayor que los problemas con tamaños más pequeños
Método	- Generar problemas de: (10 reservas y 2 habitaciones), (10 reservas y 4 habitaciones), (10 reservas y 5 habitaciones), (10 reservas y 6 habitaciones) y (10 reservas y 7 habitaciones) para la extensión 4 del problema. - Anotar y comparar sus tiempos de ejecución.

10 reservas y 2 habitaciones -> 0.02 seconds
10 reservas y 4 habitaciones -> 0.04 seconds
10 reservas y 5 habitaciones -> 1.68 seconds
10 reservas y 6 habitaciones -> 49.33 seconds
10 reservas y 7 habitaciones -> 188.05 seconds

Podemos apreciar que cuanto más grande es el tamaño del problema, más tarda el programa en ejecutarse.

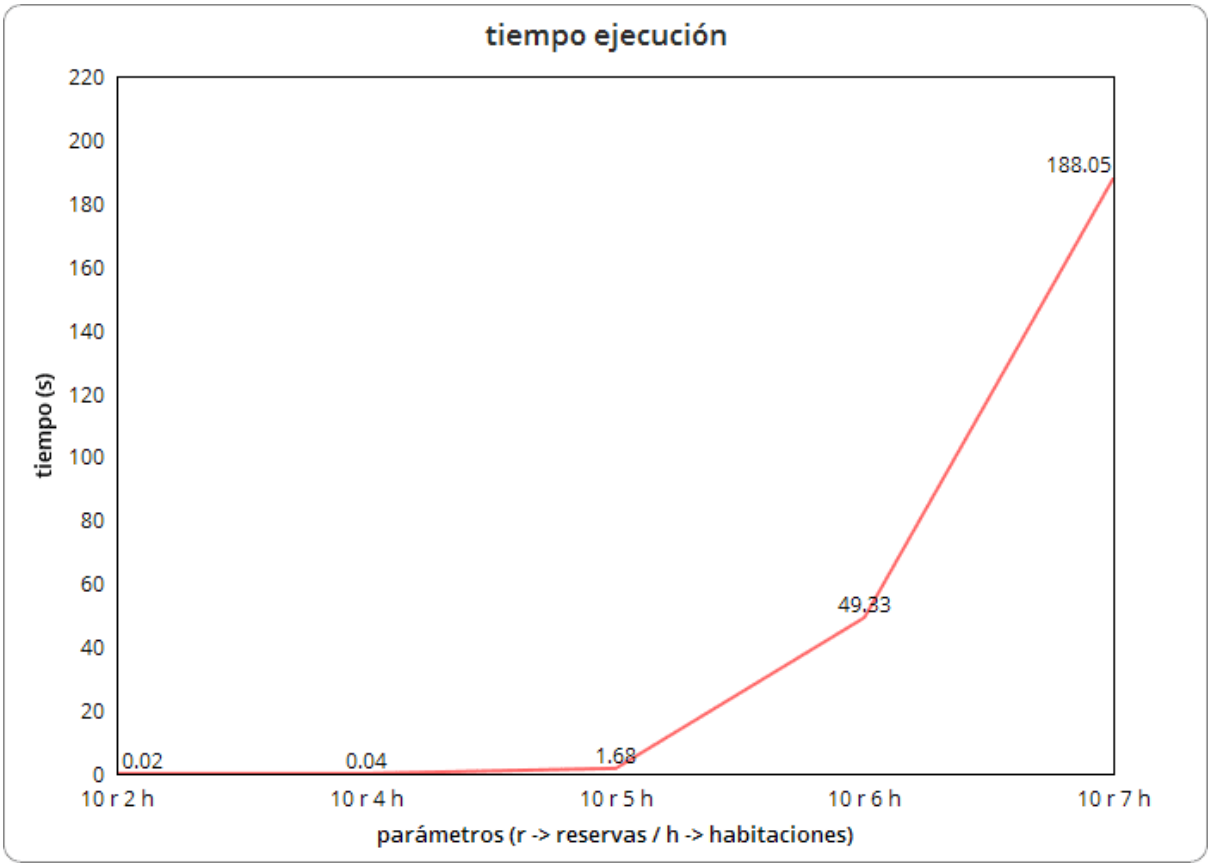

Gráfico 5.1: evolución del tiempo de ejecución según el tamaño del problema

Se puede ver que el tiempo de ejecución aumenta bastante rápido cuando se aumenta el tamaño del problema.

6. Juegos de prueba

6.1 Juego de prueba 1 (nivel básico)

6.1.1 Problema

```
(define (problem extensio0) (:domain planificador)
	(:objects c0 c1 c2 c3 c4 c5 c6 - reserva
		h0 - habitacion
	)
	(:init
		(= (personas c0) 1)
		(= (personas c1) 4)
		(= (personas c2) 1)
		(= (personas c3) 2)
		(= (personas c4) 3)
		(= (personas c5) 2)
		(= (personas c6) 3)
		(= (capacidad h0) 4)
		(= (dia_inicio c0) 9)
		(= (dia_final c0) 30)
		(= (dia_inicio c1) 8)
		(= (dia_final c1) 15)
		(= (dia_inicio c2) 19)
		(= (dia_final c2) 30)
		(= (dia_inicio c3) 27)
		(= (dia_final c3) 30)
		(= (dia_inicio c4) 6)
		(= (dia_final c4) 20)
		(= (dia_inicio c5) 2)
		(= (dia_final c5) 26)
		(= (dia_inicio c6) 22)
		(= (dia_final c6) 22)
	)
	(:goal (and (forall (?x - reserva) (servido ?x))))
)
```

6.1.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO0' defined
 ... done.

no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1g(s) + 5h(s) where
    metric is  plan length

advancing to distance:    7

best first search space empty! problem proven unsolvable.

time spent:    0.00 seconds instantiating 0 easy, 7 hard action templates
               0.00 seconds reachability analysis, yielding 28 facts and 7 actions
               0.00 seconds creating final representation with 28 relevant facts, 0 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 8 states, to a max depth of 0
               0.00 seconds total time
```

6.1.3 Explicación

Este juego de prueba ha sido creado con el generador de juegos de prueba, y como podemos observar tenemos 6 reservas y una única es imposible de satisfacer todas las reservas, por culpa de que solo tenemos una habitación y hay diversos solapamientos, como por ejemplo la reserva c0 y c1 entre otras, por lo que no obtenemos ninguna asignación así como se nos pedía en esta apartado.

6.2 Juego de prueba 2 (nivel básico)

6.2.1 Problema

```
(define (problem extensio0) (:domain planificador)
		(:objects c0 c1 - reserva
			h0 h1 h2 h3 h4 h5 - habitacion
		)
		(:init
			(= (personas c0) 3)
			(= (personas c1) 4)
			(= (capacidad h0) 4)
			(= (capacidad h1) 2)
			(= (capacidad h2) 4)
			(= (capacidad h3) 2)
			(= (capacidad h4) 1)
			(= (capacidad h5) 1)
			(= (dia_inicio c0) 26)
			(= (dia_final c0) 30)
			(= (dia_inicio c1) 10)
			(= (dia_final c1) 18)
		)
		(:goal (and (forall (?x - reserva) (servido ?x))))
)
```

6.2.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO0' defined
 ... done.

no metric specified. plan length assumed.

checking for cyclic := effects --- OK.
```

```
ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

advancing to distance:    2
                          1
                          0

ff: found legal plan as follows

step 0: ASIGNAR_HABITACION C1 H2
        1: ASIGNAR_HABITACION C0 H2

time spent:    0.00 seconds instantiating 0 easy, 4 hard action templates
               0.00 seconds reachability analysis, yielding 20 facts and 4 actions
               0.00 seconds creating final representation with 12 relevant facts, 0 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 7 states, to a max depth of 0
               0.00 seconds total time
```

6.2.3 Explicación

Este juego de prueba ha sido creado con el generador de juegos de prueba, y como podemos observar hay 2 reservas y 6 habitaciones y podemos encontrar diversas soluciones y el planificador nos de una de ellas, en este caso asignar la habitación h2 a la reserva c1 y la habitación h2 a la reserva c0.

Como podemos comprobar la asignación es correcta ya que no se solapan las dos reservas y la reserva c0 es para 3 personas y la c1 es para 4, y la habitación tiene capacidad para hasta 4 personas.

6.3 Juego de prueba 3 (extensión 1)

6.3.1 Problema

```
(define (problem extensio1) (:domain planificador)
		(:objects c0 c1 c2 c3 c4 - reserva
			h0 h1 h2 h3 - habitacion)
		(:init
			(= (personas c0) 2)
			(= (personas c1) 4)
			(= (personas c2) 3)
			(= (personas c3) 3)
			(= (personas c4) 4)
			(= (capacidad h0) 2)
			(= (capacidad h1) 1)
			(= (capacidad h2) 1)
			(= (capacidad h3) 3)
			(= (dia_inicio c0) 7)
			(= (dia_final c0) 30)
			(= (dia_inicio c1) 9)
			(= (dia_final c1) 30)
			(= (dia_inicio c2) 3)
			(= (dia_final c2) 21)
			(= (dia_inicio c3) 23)
			(= (dia_final c3) 30)
			(= (dia_inicio c4) 9)
			(= (dia_final c4) 10)
			(= (comptador) 0)
		)
		(:goal
			(and (forall (?x - reserva) (or (servido ?x) (imposible ?x)) ))
		)
		(:metric minimize (comptador))
)
```

6.3.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO1' defined
 ... done.

metric established (normalized to minimize): ((1.00*[RF0](COMPTADOR)) - () + 0.00)

checking for cyclic := effects --- OK.
```

```
ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
   metric is ((1.00*[RF0](COMPTADOR)) - () + 0.00)

advancing to distance:    6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step 0: ASIGNAR_IMPOSIBILIDAD_RESERVA C1
      1: ASIGNAR_IMPOSIBILIDAD_RESERVA C4
      2: ASIGNAR_HABITACION C3 H3
      3: ASIGNAR_HABITACION C2 H3
      4: ASIGNAR_HABITACION C0 H0
      5: REACH-GOAL

time spent:    0.00 seconds instantiating 0 easy, 13 hard action templates
               0.00 seconds reachability analysis, yielding 37 facts and 19 actions
               0.00 seconds creating final representation with 20 relevant facts, 1 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 34 states, to a max depth of 0
               0.00 seconds total time
```

6.3.3 Explicación

Este juego de pruebas ha sido generador por el generador automático y tenemos 5 reservas (c0, c1, c2, c3, c4) y 4 habitaciones (h0, h1, h2, h3) y el objetivo que queremos ver es que se obtenga una asignación de reserves a habitaciones en las que caben las personas de la reserva (la habitación puede ser más grande) y no hay solapamiento en las ocupaciones y que pueden quedar reservas sin asignar.

Se puede apreciar que ha encontrado un plan legal que resuelve el problema. A diferencia del nivel básico, en esta extensión puede quedar reservas sin asignar, por ejemplo c1 y c4. A c1 no se le puede asignar ninguna habitación porque tiene 4 personas y todas las habitaciones tienen capacidad para alojar 1, 2 o 3 personas. Lo mismo ocurre con c4, que también es de 4 personas. También se puede apreciar que no hay solapamiento de ocupaciones de las habitaciones. Ya que c0 se le asigna la habitación h0, mientras que c2 y c3 se le asignan la habitación h3, pero c2 va del 3 al 21 y c3 va del 23 al 30, por lo que no solapan y el resultado es totalmente correcto y esperado.

6.4 Juego de prueba 4 (extensión 1)

6.4.1 Problema

```
(define (problem test-01) (:domain planificador)
   (:objects
      c1 c2 c3 c4 c5 - reserva
      h1 - habitacion
   )
   (:init
      (= (capacidad h1) 3)
      (= (personas c1) 2)
      (= (personas c2) 2)
      (= (personas c3) 2)
      (= (personas c4) 2)
      (= (personas c5) 2)
      (= (dia_inicio c1) 1)
      (= (dia_final c1) 30)
      (= (dia_inicio c2) 2)
      (= (dia_final c2) 4)
      (= (dia_inicio c3) 5)
      (= (dia_final c3) 6)
      (= (dia_inicio c4) 6)
      (= (dia_final c4) 9)
      (= (dia_inicio c5) 9)
      (= (dia_final c5) 13)
      (= (comptador) 0)
   )
   (:goal
      (and (forall (?x - reserva) (or (servido ?x) (imposible ?x)) ))
   )
   (:metric minimize (comptador))
)
```

6.4.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'TEST-01' defined
 ... done.

metric established (normalized to minimize): ((1.00*[RF0](COMPTADOR)) - () + 0.00)

checking for cyclic := effects --- OK.
```

```
ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
   metric is ((1.00*[RF0](COMPTADOR)) - () + 0.00)

advancing to distance:    6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step 0: ASIGNAR_HABITACION C5 H1
      1: ASIGNAR_IMPOSIBILIDAD_RESERVA C1
      2: ASIGNAR_HABITACION C4 H1
      3: ASIGNAR_HABITACION C3 H1
      4: ASIGNAR_HABITACION C2 H1
      5: REACH-GOAL

time spent:    0.00 seconds instantiating 0 easy, 18 hard action templates
               0.00 seconds reachability analysis, yielding 25 facts and 50 actions
               0.00 seconds creating final representation with 26 relevant facts, 1 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 21 states, to a max depth of 0
               0.00 seconds total time
```

6.4.3 Explicación

Este juego de pruebas ha sido creado manualmente por nosotros y tenemos 5 reservas (c1, c2, c3, c4, c5) y 1 habitación (h1) y el objetivo que queremos ver es que se obtenga una asignación de reserves a habitaciones en las que caben las personas de la reserva (la habitación puede ser más grande) y no hay solapamiento en las ocupaciones y que pueden quedar reservas sin asignar y que se optimiza el número de reservas asignadas.

Se puede apreciar que ha encontrado un plan legal que resuelve el problema. Todas las reservas son de 2 personas y la habitación tiene capacidad para 3 personas, por lo que todas son candidatas a ser asignadas. Viendo los días de inicio y final de las reservas, se puede ver que c1 va del 1 al 30, y los otros 4 no se colapsan, por lo que el resultado tiene que asignar la habitación a c2, c3, c4 y c5, dejando sin habitación a c1, en este caso optimizaremos el número de reservas.

Efectivamente, el programa asigna c2, c3, c4 y c5, por lo que podemos concluir que funciona correctamente y cumple con lo esperado.

6.5 Juego de prueba 5 (extensión 2)

6.5.1 Problema

```
(define (problem extensio2) (:domain planificador)
        (:objects c0 c1 c2 c3 c4 - reserva
                h0 h1 h2 h3 - habitacion
                N S E O - orientacion)
        (:init
                (= (personas c0) 4)
                (= (personas c1) 1)
                (= (personas c2) 1)
                (= (personas c3) 2)
                (= (personas c4) 2)
                (= (capacidad h0) 1)
                (= (capacidad h1) 4)
                (= (capacidad h2) 2)
                (= (capacidad h3) 4)
                (= (dia_inicio c0) 10)
                (= (dia_final c0) 26)
                (= (dia_inicio c1) 2)
                (= (dia_final c1) 29)
                (= (dia_inicio c2) 20)
                (= (dia_final c2) 30)
                (= (dia_inicio c3) 17)
                (= (dia_final c3) 17)
                (= (dia_inicio c4) 22)
                (= (dia_final c4) 30)
                (orientacionH h0 N)
                (orientacionH h1 E)
                (orientacionH h2 O)
                (orientacionH h3 E)
                (orientacionR c0 O)
                (orientacionR c1 E)
                (orientacionR c2 N)
                (orientacionR c3 O)
                (orientacionR c4 E)
                (= (comptador) 0)
                (= (comptadorO) 0)
        )
        (:goal
                (and (forall (?x - reserva) (or (servido ?x) (imposible ?x)) ))
        )
        (:metric minimize (+ (* (comptador) 10) (comptadorO)))
)
```

6.5.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO2' defined
 ... done.

metric established (normalized to minimize):
((10.00*[RF1](COMPTADOR)1.00*[RF0](COMPTADORO)) - () + 0.00)

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((10.00*[RF1](COMPTADOR)1.00*[RF0](COMPTADORO)) - () + 0.00)

advancing to distance:    6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step 0: ASIGNAR_HABITACION_ORIENTACION C3 H2
        1: ASIGNAR_HABITACION_ORIENTACION C2 H0
        2: ASIGNAR_HABITACION_ORIENTACION C0 H1
        3: ASIGNAR_HABITACION_ORIENTACION C4 H2
        4: ASIGNAR_HABITACION_ORIENTACION C1 H3
        5: REACH-GOAL

time spent:    0.00 seconds instantiating 0 easy, 231 hard action templates
               0.00 seconds reachability analysis, yielding 51 facts and 222 actions
               0.00 seconds creating final representation with 48 relevant facts, 2 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.01 seconds searching, evaluating 44 states, to a max depth of 0
               0.01 seconds total time
```

6.5.3 Explicación

Este juego de pruebas ha sido creado automáticamente con el generador de juegos de prueba y tenemos 5 reservas, 4 habitaciones y las 4 orientaciones posibles. Y el objetivo es asignar el máximo de reservas posibles y que el número de reservas satisfechas con su

orientación sea máxima, priorizando asignar todas las reservas posibles por sobre de asignar la orientación preferente a una reserva si esto supone dejar a otra sin asignar. La solución que obtenemos por parte del planificador es asignar a la reserva c3 la habitación h2, a la reserva c2 la habitación h0, a la reserva c0 la habitación h1, a la reserva c4 la habitación h2 y a la reserva c1 la habitación h3.

La asignación de c3 a h2 es correcta ya que cumple la orientación preferente por parte del cliente y tiene la capacidad suficiente además es la única reserva que tiene como orientación preferente la misma que h2 y quepa en esta.

La asignación de c2 a h0 es correcta ya que cumple la orientación preferente por parte del cliente y tiene la capacidad suficiente además es la única reserva que tiene como orientación preferente la misma que h0.
La asignación de c0 a h1 es correcta ya que a pesar de no cumplir con su preferencia para la orientación no hay ninguna habitación con capacidad suficiente que la cumpla.

La asignación de c1 a h3 es correcta ya que cumple la orientación preferente por parte del cliente y tiene la capacidad suficiente además solo existe otra habitación con la misma orientación que es h1, pero esta ya está asignada a c0 y las reservas c0 y c1 se solapan.

La asignación de c4 a h2 es correcta ya que a pesar de no cumplir con su preferencia para la orientación no hay ninguna habitación libre que la cumpla. Ya que las habitaciones que tienen la misma orientación que la preferente por la reserva són h3 y h1 y a pesar de que ambas tienen capacidad suficiente es imposible asignar la reserva a ninguna de estas habitaciones. En el caso de h3 la habitación ya está ocupada durante su estancia y habría solapamiento, y la reserva que se encuentra en h3 tiene la misma orientación preferente. Por otro lado en la habitación h1 se encuentra la reserva c0 que tiene una orientación preferente diferente a la que tiene la habitación, pero vemos que no podemos asignarla a ninguna otra habitación ya que solo las habitaciones h1 y h3 tienen capacidad suficiente para alojar la reserva c0, y en la habitación ya hay una reserva , con la que se solapa, por lo que si se asignase c4 a la habitación h1 esto supondría que la reserva c0 quedaría sin asignar.

Así pues podemos afirmar que el planificador se ha comportado de manera correcta ya que se nos pedía asignar el máximo de reservas posibles y que el número de reservas satisfechas con su orientación sea máxima, priorizando asignar todas las reservas posibles

por sobre de asignar la orientación preferente a una reserva si esto supone dejar a otra sin asignar, y es lo que el planificador ha hecho.

6.6 Juego de prueba 6 (extensión 2)

6.6.1 Problema

```
(define (problem extensio2) (:domain planificador)

        (:objects c0 c1 c2 c3 - reserva
                h0 h1 h2 h3 h4 - habitacion
                N S E O - orientacion)
        (:init
                (= (personas c0) 3)
                (= (personas c1) 2)
                (= (personas c2) 4)
                (= (personas c3) 2)
                (= (capacidad h0) 2)
                (= (capacidad h1) 3)
                (= (capacidad h2) 1)
                (= (capacidad h3) 1)
                (= (capacidad h4) 3)
                (= (dia_inicio c0) 24)
                (= (dia_final c0) 30)
                (= (dia_inicio c1) 22)
                (= (dia_final c1) 30)
                (= (dia_inicio c2) 11)
                (= (dia_final c2) 11)
                (= (dia_inicio c3) 8)
                (= (dia_final c3) 18)
                (orientacionH h0 S)
                (orientacionH h1 O)
                (orientacionH h2 E)
                (orientacionH h3 O)
                (orientacionH h4 N)
                (orientacionR c0 E)
                (orientacionR c1 N)
                (orientacionR c2 S)
                (orientacionR c3 S)
                (= (comptador) 0)
                (= (comptadorO) 0)
        )
        (:goal
                (and (forall (?x - reserva) (or (servido ?x) (imposible ?x)) ))
        )
        (:metric minimize (+ (* (comptador) 10) (comptadorO)))
)
```

6.6.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO2' defined
 ... done.

metric established (normalized to minimize):
((10.00*[RF1](COMPTADOR)1.00*[RF0](COMPTADORO)) - () + 0.00)

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((10.00*[RF1](COMPTADOR)1.00*[RF0](COMPTADORO)) - () + 0.00)

advancing to distance:    5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step 0: ASIGNAR_IMPOSIBILIDAD_RESERVA C2
        1: ASIGNAR_HABITACION_ORIENTACION C3 H0
        2: ASIGNAR_HABITACION_ORIENTACION C0 H1
        3: ASIGNAR_HABITACION_ORIENTACION C1 H4
        4: REACH-GOAL

time spent:    0.00 seconds instantiating 0 easy, 18 hard action templates
               0.00 seconds reachability analysis, yielding 39 facts and 23 actions
               0.00 seconds creating final representation with 27 relevant facts, 2 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 33 states, to a max depth of 0
               0.00 seconds total time
```

6.6.3 Explicación

Este juego de pruebas ha sido creado automáticamente con el generador de juegos de prueba y tenemos 4 reservas, 5 habitaciones y las 4 orientaciones posibles. Y el objetivo es asignar el máximo de reservas posibles y que el número de reservas satisfechas con su orientación sea máxima, priorizando asignar todas las reservas posibles por sobre de asignar la orientación preferente a una reserva si esto supone dejar a otra sin asignar. La solución que obtenemos por parte del planificador es asignar a la reserva c3 y c1 a la habitación h0 la reserva c0 a la habitación h1 y la reserva c2 como imposible de asignar.

Que salga la reserva c2 como imposible de asignar es correcto ya que nos encontramos con que de las 5 habitaciones ninguna tiene capacidad suficiente para albergar a las 4 personas de la reserva c2.

Se asigna correctamente la habitación h0 a la reserva c3, y que coincide con la orientación y tiene la capacidad suficiente.

Asignar la habitación h1 a la reserva c0 es correcto ya que a pesar de no satisfacer su orientación preferente, no hay ninguna habitación que la cumpla y c0 pueda ser asignada.

Se asigna correctamente la habitación h4 a la reserva c1, y que coincide con la orientación y tiene la capacidad suficiente.

Así pues podemos afirmar que el planificador se ha comportado de manera correcta ya que se nos pedía asignar el máximo de reservas posibles y que el número de reservas satisfechas con su orientación sea máxima, priorizando asignar todas las reservas posibles por sobre de asignar la orientación preferente a una reserva si esto supone dejar a otra sin asignar, y es lo que el planificador ha hecho.

6.7 Juego de prueba 7 (extensión 3)

6.7.1 Problema

```
(define (problem extensio3) (:domain planificador)
        (:objects c0 c1 c2 c3 c4 c5 c6 - reserva
            h0 h1 h2 h3 h4 h5 - habitacion
        )
        (:init
            (= (personas c0) 1)
            (= (personas c1) 3)
            (= (personas c2) 1)
            (= (personas c3) 4)
            (= (personas c4) 3)
            (= (personas c5) 4)
            (= (personas c6) 4)
            (= (capacidad h0) 1)
            (= (capacidad h1) 2)
            (= (capacidad h2) 1)
            (= (capacidad h3) 3)
            (= (capacidad h4) 1)
            (= (capacidad h5) 4)
            (= (dia_inicio c0) 27)
            (= (dia_final c0) 30)
            (= (dia_inicio c1) 27)
            (= (dia_final c1) 30)
            (= (dia_inicio c2) 30)
            (= (dia_final c2) 30)
            (= (dia_inicio c3) 30)
            (= (dia_final c3) 30)
            (= (dia_inicio c4) 17)
            (= (dia_final c4) 30)
            (= (dia_inicio c5) 22)
            (= (dia_final c5) 30)
            (= (dia_inicio c6) 22)
            (= (dia_final c6) 30)
            (= (comptador) 0)
            (= (comptadorPlaces) 0)
        )

        (:goal
            (and (forall (?x - reserva) (or (servido ?x) (imposible ?x)) ))
        )

        (:metric minimize (+ (comptadorPlaces) (* (comptador) 8)))
)
```

6.7.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO3' defined
 ... done.


metric established (normalized to minimize):
((1.00*[RF0](COMPTADORPLACES)8.00*[RF1](COMPTADOR)) - () + 0.00)

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is ((1.00*[RF0](COMPTADORPLACES)8.00*[RF1](COMPTADOR)) - () + 0.00)

advancing to distance:    8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: ASIGNAR_HABITACION C2 H4
        1: ASIGNAR_HABITACION C0 H0
        2: ASIGNAR_IMPOSIBILIDAD_RESERVA C5
        3: ASIGNAR_IMPOSIBILIDAD_RESERVA C3
        4: ASIGNAR_HABITACION C6 H5
        5: ASIGNAR_IMPOSIBILIDAD_RESERVA C1
        6: ASIGNAR_HABITACION C4 H3
        7: REACH-GOAL


time spent:    0.00 seconds instantiating 0 easy, 722 hard action templates
               0.00 seconds reachability analysis, yielding 82 facts and 276 actions
               0.00 seconds creating final representation with 60 relevant facts, 2 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.02 seconds searching, evaluating 2123 states, to a max depth of 0
               0.02 seconds total time
```

6.7.3 Explicación

Este juego de prueba ha sido realizado por el generador implementado. El nivel 3 de la solución destaca respecto al nivel 1 por tener en cuenta el número de plazas e intenta minimizar el desperdicio de plazas. Por tanto, este juego de pruebas va destinado a comprobar que optimiza de forma correcta. El juego tiene 7 reservas y 6 habitaciones.

Este juego tiene 3 habitaciones individuales (H0, H2, H4) y tiene sólo 2 reservas (C0, C2) individuales, por tanto, no ha habido problemas para asignarlas. Las otras reservas son para 3 y 4 personas y la habitación para 2 personas ha quedado sin asignar. Hay una habitación triple (H3) y otra cuádruple (H5) para 5 reservas (2 triples C1, C4 y 3 cuádruples C3, C5, C6). Las reservas son todas incompatibles entre sí. Se asignan una reserva triple (C4) y una cuádruple (C6) para minimizar el número de plazas sobrantes.

6.8 Juego de prueba 8 (extensión 3)

6.8.1 Problema

```
(define (problem extensio3) (:domain planificador)
        (:objects c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 - reserva
            h0 h1 h2 h3 h4 h5 h6 - habitacion
        )
        (:init
            (= (personas c0) 1)
            (= (personas c1) 1)
            (= (personas c2) 3)
            (= (personas c3) 3)
            (= (personas c4) 3)
            (= (personas c5) 1)
            (= (personas c6) 1)
            (= (personas c7) 4)
            (= (personas c8) 4)
            (= (personas c9) 2)
            (= (capacidad h0) 3)
            (= (capacidad h1) 1)
            (= (capacidad h2) 3)
            (= (capacidad h3) 3)
            (= (capacidad h4) 4)
            (= (capacidad h5) 1)
            (= (capacidad h6) 2)
            (= (dia_inicio c0) 25)
            (= (dia_final c0) 30)
            (= (dia_inicio c1) 13)
            (= (dia_final c1) 30)
            (= (dia_inicio c2) 26)
            (= (dia_final c2) 30)
            (= (dia_inicio c3) 5)
            (= (dia_final c3) 20)
            (= (dia_inicio c4) 25)
            (= (dia_final c4) 30)
            (= (dia_inicio c5) 10)
            (= (dia_final c5) 15)
            (= (dia_inicio c6) 25)
            (= (dia_final c6) 30)
            (= (dia_inicio c7) 18)
            (= (dia_final c7) 30)
            (= (dia_inicio c8) 19)
            (= (dia_final c8) 30)
            (= (dia_inicio c9) 20)
            (= (dia_final c9) 30)
            (= (comptador) 0)
```

```
        (= (comptadorPlaces) 0)
    )
    (:goal
        (and (forall (?x - reserva) (or (servido ?x) (imposible ?x)) ))
    )
    (:metric minimize (+ (comptadorPlaces) (* (comptador) 8)))
)
```

6.8.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO3' defined
 ... done.


metric established (normalized to minimize):
((1.00*[RF0](COMPTADORPLACES)8.00*[RF1](COMPTADOR)) - () + 0.00)

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
   metric is ((1.00*[RF0](COMPTADORPLACES)8.00*[RF1](COMPTADOR)) - () + 0.00)

advancing to distance:   11
                         10
                          9
                          8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: ASIGNAR_HABITACION C9 H6
        1: ASIGNAR_HABITACION C5 H4
        2: ASIGNAR_HABITACION C4 H0
        3: ASIGNAR_HABITACION C2 H2
        4: ASIGNAR_IMPOSIBILIDAD_RESERVA C7
        5: ASIGNAR_HABITACION C8 H4
        6: ASIGNAR_HABITACION C6 H3
```

7: ASIGNAR_HABITACION C1 H5
8: ASIGNAR_HABITACION C3 H0
9: ASIGNAR_HABITACION C0 H1
10: REACH-GOAL

time spent: 315.74 seconds instantiating 0 easy, 169945 hard action templates
0.03 seconds reachability analysis, yielding 147 facts and 42408 actions
0.03 seconds creating final representation with 125 relevant facts, 2 relevant fluents
0.09 seconds computing LNF
0.46 seconds building connectivity graph
11.15 seconds searching, evaluating 15750 states, to a max depth of 0
327.50 seconds total time

6.8.3 Explicación

Este juego de prueba ha sido realizado por el generador implementado. El nivel 3 de la solución destaca respecto al nivel 1 por tener en cuenta el número de plazas e intenta minimizar el desperdicio de plazas. Por tanto, este juego de pruebas va destinado a comprobar que optimiza de forma correcta. El juego tiene 10 reservas y 7 habitaciones.

Como se puede comprobar, tenemos dos habitaciones con capacidad 1(H1, H5) y tenemos 4 reservas para 1 persona (C0, C1, C5, C6). Podemos observar que la C0 y la C1 han sido asignadas a las habitaciones H1 y H5 respectivamente, pero las 4 reservas se solapan entre sí y por este motivo no se han podido asignar las reservas C5 y C6 a habitaciones con tamaño 1.

Tenemos una habitación doble (H6) que se ha asignado a la única reserva doble (C9). El resto de reservas con un máximo de 3 personas han sido asignadas a habitaciones con capacidad 3 a excepción de la 5 que no ha podido ser asignada por motivos de solapamiento. Finalmente, tenemos una habitación cuádruple (H4) y dos reservas para 4 personas pero están solapadas (C7, C8) , y por tanto, solo se asigna una habitación y la otra queda como no posible de asignar. La reserva 5 que no se había podido asignar a una habitación con su capacidad ha sido asignada a la habitación cuádruple (H4). Cabe recordar que la reserva 5 era para una persona, por tanto, se han priorizado las demás reservas con capacidad mayor que eran posibles para las habitaciones triples.

La conclusión que podemos obtener es que siempre que sea posible asignar reservas minimizando el desperdicio de plazas el planificador lo ha hecho de forma correcta.

6.9 Juego de prueba 9 (extensión 4)

6.9.1 Problema

```
(define (problem extensio4) (:domain planificador)
	(:objects c0 c1 c2 c3 c4 c5 c6 - reserva
		h0 h1 h2 h3 h4 - habitacion
	)
	(:init
		(= (personas c0) 2)
		(= (personas c1) 3)
		(= (personas c2) 2)
		(= (personas c3) 1)
		(= (personas c4) 3)
		(= (personas c5) 4)
		(= (personas c6) 4)
		(= (capacidad h0) 3)
		(= (capacidad h1) 4)
		(= (capacidad h2) 4)
		(= (capacidad h3) 3)
		(= (capacidad h4) 2)
		(= (dia_inicio c0) 15)
		(= (dia_final c0) 16)
		(= (dia_inicio c1) 27)
		(= (dia_final c1) 29)
		(= (dia_inicio c2) 8)
		(= (dia_final c2) 30)
		(= (dia_inicio c3) 6)
		(= (dia_final c3) 30)
		(= (dia_inicio c4) 25)
		(= (dia_final c4) 30)
		(= (dia_inicio c5) 9)
		(= (dia_final c5) 30)
		(= (dia_inicio c6) 4)
		(= (dia_final c6) 8)
		(= (comptador) 0)
		(= (comptadorPlaces) 0)
		(= (comptadorHabitacions) 0)
	)
	(:goal
		(and (forall (?x - reserva) (or (servido ?x) (imposible ?x)) ))
	)
	(:metric minimize (+ (+(comptadorPlaces) (* (comptador) 8)) (*
(comptadorHabitacions) 4)))
)
```

6.9.2 Output

```
ff: parsing domain file
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO4' defined
 ... done.

metric established (normalized to minimize):
((1.00*[RF1](COMPTADORPLACES)8.00*[RF2](COMPTADOR)4.00*[RF0](COMPTADORH
ABITACIONS)) - () + 0.00)

task contains conditional effects. turning off state domination.

checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is
((1.00*[RF1](COMPTADORPLACES)8.00*[RF2](COMPTADOR)4.00*[RF0](COMPTADORH
ABITACIONS)) - () + 0.00)

advancing to distance:    8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step 0: ASIGNAR_HABITACION C2 H0
        1: ASIGNAR_HABITACION C3 H4
        2: ASIGNAR_HABITACION C0 H3
        3: ASIGNAR_HABITACION C4 H3
        4: ASIGNAR_HABITACION C1 H2
        5: ASIGNAR_HABITACION C6 H1
        6: ASIGNAR_HABITACION C5 H1
        7: REACH-GOAL

time spent:    2.86 seconds instantiating 0 easy, 18438 hard action templates
               0.00 seconds reachability analysis, yielding 93 facts and 4926 actions
               0.00 seconds creating final representation with 86 relevant facts, 3 relevant fluents
               0.01 seconds computing LNF
               0.01 seconds building connectivity graph
```

1.46 seconds searching, evaluating 15274 states, to a max depth of 0
4.34 seconds total time

6.9.3 Explicación

Este juego de prueba pretende comprobar el correcto funcionamiento del nivel 4 implementado. El nivel 4 tiene un cuenta un factor más al optimizar y consiste en intentar asignar el número mínimo de habitaciones siempre que sea posible asignar las mismas reservas que con un número mayor de habitaciones.

El juego de pruebas ha sido generado por el generador implementado. Se ha obtenido un juego de pruebas con 7 reservas y 5 habitaciones. Se han podido asignar todas las reservas utilizando las 5 habitaciones. Queremos comprobar que realmente nuestro sistema funciona y no era posible asignar todas las reservas utilizando un número menor de habitaciones.

Tenemos una habitación doble (H4) y tenemos tres reservas que podrían asignarse a H4 (C0, C2, C3), pero están solapadas entre sí, y por tanto, solo se puede asignar una de las tres, el sistema ha asignado la C3. Tenemos 2 habitaciones triples (H0, H3) y podrían asignarse 4 reservas (C0, C1, C2, C4). De las posibles reservas solamente son compatibles las reservas C0 con C1 y C0 con C4. El sistema ha asignado las reservas C0 juntamente con C4 para intentar minimizar el número de habitaciones correctamente y C2 de forma arbitraria. En caso de no haber podido asignar todas las reservas se habría preferido asignar las reservas con el menor número de plazas desperdiciadas, pero este número va a ser constante en caso de asignarse.

Finalmente, tenemos dos habitaciones cuádruples (H1, H2) para 3 reservas (C1, C5, C6) y solamente són compatibles C1 con C6 y C5 con C6. Se asigna C5 y C6 conjuntamente y C1.

Para comprobar que realmente no había una solución mejor necesitamos comprobar que C3 y C2 que han sido asignados de forma arbitraria no son compatibles con C1 y C5. Además también es necesario tener en cuenta que C0 y C4 conjuntamente no son compatibles con C1 y C5. Una vez comprobados estos solapamientos demostramos que la asignación realizada no podía realizarse con un número menor de habitaciones.

6.10 Juego de prueba 10 (extensión 4)

6.10.1 Problema

```
(define (problem extensio4) (:domain planificador)

        (:objects c0 c1 c2 c3 c4 - reserva
                h0 h1 h2 h3 h4 - habitacion
        )

        (:init
                (= (personas c0) 2)
                (= (personas c1) 1)
                (= (personas c2) 1)
                (= (personas c3) 1)
                (= (personas c4) 2)
                (= (capacidad h0) 3)
                (= (capacidad h1) 2)
                (= (capacidad h2) 3)
                (= (capacidad h3) 4)
                (= (capacidad h4) 2)
                (= (dia_inicio c0) 30)
                (= (dia_final c0) 30)
                (= (dia_inicio c1) 26)
                (= (dia_final c1) 30)
                (= (dia_inicio c2) 29)
                (= (dia_final c2) 30)
                (= (dia_inicio c3) 24)
                (= (dia_final c3) 26)
                (= (dia_inicio c4) 21)
                (= (dia_final c4) 24)
                (= (comptador) 0)
                (= (comptadorPlaces) 0)
                (= (comptadorHabitacions) 0)
        )

        (:goal
                (and (forall (?x - reserva) (or (servido ?x) (imposible ?x)) ))
        )

        (:metric minimize (+ (+(comptadorPlaces) (* (comptador) 8)) (*
(comptadorHabitacions) 4)))
)
```

6.10.2 Output

```
ff: parsing domain file
```

```
domain 'PLANIFICADOR' defined
 ... done.
ff: parsing problem file
problem 'EXTENSIO4' defined
 ... done.


metric established (normalized to minimize):
((1.00*[RF1](COMPTADORPLACES)8.00*[RF2](COMPTADOR)4.00*[RF0](COMPTADORH
ABITACIONS)) - () + 0.00)

task contains conditional effects. turning off state domination.



checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
   metric is
((1.00*[RF1](COMPTADORPLACES)8.00*[RF2](COMPTADOR)4.00*[RF0](COMPTADORH
ABITACIONS)) - () + 0.00)

advancing to distance:    6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: ASIGNAR_HABITACION C3 H4
        1: ASIGNAR_HABITACION C1 H1
        2: ASIGNAR_HABITACION C4 H1
        3: ASIGNAR_HABITACION C2 H0
        4: ASIGNAR_HABITACION C0 H4
        5: REACH-GOAL


time spent:    0.00 seconds instantiating 0 easy, 540 hard action templates
               0.00 seconds reachability analysis, yielding 75 facts and 572 actions
               0.00 seconds creating final representation with 76 relevant facts, 3 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.01 seconds searching, evaluating 538 states, to a max depth of 0
               0.01 seconds total time
```

6.10.3 Explicación

Este juego de prueba pretende comprobar el correcto funcionamiento del nivel 4 implementado. El nivel 4 tiene un cuenta un factor más al optimizar y consiste en intentar asignar el número mínimo de habitaciones siempre que sea posible asignar las mismas reservas que con un número mayor de habitaciones.

El juego de pruebas ha sido generado por el generador implementado. Se ha obtenido un juego de pruebas con 5 reservas y 5 habitaciones. Se han podido asignar todas las reservas utilizando 3 habitaciones. Queremos comprobar que realmente nuestro sistema funciona y no era posible asignar todas las reservas utilizando un número menor de habitaciones, además siempre que sea posible se han dejado sin utilizar las habitaciones mayores para minimizar las plazas desperdiciadas.

Todas nuestras reservas pueden asignarse a las habitaciones que disponemos porque su capacidad es superior a la solicitada en la reserva. Al no tener incompatibilidades entre reservas y plazas basta con no asignar las habitaciones mayores para comprobar que se utilizan las habitaciones menores posibles. El sistema lo ha hecho de forma correcta (H3 con capacidad 4 y H2 con capacidad 3 no han sido asignadas).

Para comprobar que no era posible asignar todas las reservas utilizando un número menor a tres habitaciones basta con ver que las reservas C0, C1, C2 son incompatibles entre sí, y por tanto, necesitamos las tres habitaciones para poder asignarlas todas.

7. Planificación de la práctica

En cuanto a la planificación de la práctica seguimos la guía que nos fue dada en el laboratorio de IA, por lo que lo primero que hicimos fue entender el enunciado y comprender el funcionamiento de PDDL y Fast Forward. A continuación, fue pensar y diseñar el dominio para el nivel básico.

Una vez contamos con un primer diseño y sabíamos por dónde empezar, nos dividimos los distintos niveles de manera que pudiésemos trabajar por separado y compartiendo los avances que íbamos haciendo. Paralelamente, se hicieron los distintos generadores de problema.

Finalmente, hicimos los distintos juegos de prueba y terminamos de documentar toda la práctica.

Fecha	Contenido
Semana 1	Lectura del enunciado, Introducción sobre PDDL y Fast Forward y una primera discusión sobre el dominio del problema
Semana 2	Diseño de las diferentes extensiones del problema, creación de los juegos de prueba y documentación final

Reparto del trabajo:

- Nivel básico → Conjuntamente
- Nivel 1 → Joan
- Nivel 2 → Pau
- Nivel 3 → Joan
- Nivel 4 → Joan
- Generador de problemas → Jiabo
- Juegos de prueba → Conjuntamente
- Documentación → Conjuntamente

Realmente, hemos ido ayudándonos entre nosotros y no ha sido una repartición estricta.

8. Conclusiones

La práctica realizada nos ha permitido conocer cómo funciona el lenguaje PDDL y el funcionamiento práctico de un planificador. Así como hemos podido comprobar en los distintos juegos de pruebas el sistema funciona correctamente, cumple con los objetivos marcados inicialmente y muestra los resultados esperados. Hemos comprobado que el coste de nuestro sistema es exponencial, y por tanto, es costoso realizar la planificación sobre datos muy grandes. Una posible mejora sería la eficiència del problema.