Lunes 10 Enero 2022

PLANIFICACIÓN

Inteligencia Artificial
Universitat Politècnica de Catalunya
2021-2022 Q1

AUTORAS:
CARLA CAMPÀS GENÉ
BEATRIZ GOMES DA COSTA
LAIA VALLÈS GUILERA

1 INTRODUCCIÓN

Los hoteles reciben diferentes reservas con diferentes preferencias. Esta práctica tiene como objetivo evaluar las preferencias de los diferentes usuarios que han reservado en un hotel dado y crear la mejor asignación posible para las diferentes reservas.

Dado que la ponderación de las preferencias pueden venir definidas por diferentes criterios y diferentes variables en esta práctica crearemos cinco diferentes sistemas de planificación de las reservas de un hotel. De esta manera optimizaremos los resultados utilizando diferentes métodos y veremos la diferencia entre estos.

Para simplificar esta situación hemos optado por asumir que las reservas no tendrán ningún error que podrían prohibir la creación de una planificación. Esto sería por ejemplo, que hubiese reservas de un número de personas más grande que la habitación más grande. Además tenemos algunas restricciones dadas por el enunciado:

- Las habitaciones y reservas serán de 1 a 4 personas
- Los días irán de 1 a 30 (solo se puede reservar con un mes de antelación)

2 DOMINIO

A continuación describiremos en detalle el dominio que se ha creado para las diferentes extensiones. Haremos también distinción entre los usos de diferentes funciones y predicados en las diferentes extensiones.

2.1 Variables

2.1.1 Habitación

Representación de las habitaciones que tenemos en un hotel dado. Esta será de tipo objeto. Este objeto nos permitirá usar un identificador único para diferenciar las habitaciones. Mediante este objeto también vamos a poder crear relaciones de reservas. Además podremos obtener diferentes atributos de esta mediante funciones para distinguir las diferentes características que tiene cada habitación y crear ponderaciones de preferencias usando esta.

2.1.2 Reserva

Representación de las reservas hechas por usuarios al hotel donde se ubica nuestro sistema. Esta será de tipo objeto. Esta variable nos permitirá usar un identificador único para cada reserva que haya en el sistema. Mediante el objeto vamos a poder relacionar las reservas con diferentes preferencias y características que esta pueda tener en las diferentes implementaciones del sistema. Además también va a permitir relacionar una reserva con una habitación indicando que esta ha sido reservada.

2.2 Predicados

2.2.1 Visitada

El predicado visitado, tiene el siguiente formato ***(visitada ?r - reserva)***. Este se usará en todas las extensiones para ver si una visita ya se ha reservado en algún punto de la ejecución. En el nivel básico obviamos el predicado que reservada (que veremos a continuación) ya que todas las reservas que sean visitadas también deberán ser asignadas.

2.2.2 Reservada

El predicado reservada, tiene el siguiente formato ***(reservada ?r - reserva)***. Indica si la reserva está asignada a alguna habitación en cualquier momento de la ejecución. Este predicado se usa a partir de

la primera extensión y nos permite controlar qué reservas ya no podemos reservar y que reservas han siguen libres.

2.2.3 Habitación Asignada

El predicado habitación asignada tiene el siguiente formato ***(habitacion_asignada ?h - habitacion ?r - reserva)*** indica que la habitación h está asignada a la reserva r. Se plantea su uso a partir de la primera extensión donde tenemos que comprobar que reservas han sido asignadas a una habitación para poder comparar esas que se solapan.

2.2.4 Habitación Visitada

El predicado habitación visitada tiene el siguiente formato ***(habitacion_visitada ?h - habitacion ?r - reserva)***. Indica que en algún momento la habitación h se ha asignado a la reserva r. Se plantea su uso a partir de la primera extensión donde tenemos que comprobar que reservas han sido asignadas a una habitación en algún momento de la ejecución con tal de no volver a visitar esa habitación.

2.2.5 Usada

El predicado usada tiene el siguiente formato ***(usada ?h habitacion)***. Indica si la habitación h está asignada a alguna reserva. Este predicado se usa únicamente en la cuarta extensión donde también tenemos el objetivo de minimizar el número de habitaciones que pretendemos usar, por lo tanto usaremos este predicado para identificar si una habitación está, o no, en uso.

2.3 Funciones

2.3.1 Tamaño

La función tamaño tiene el siguiente formato ***(tamano ?x - objeto)***. Esta función la usaremos tanto en caso En el caso de las habitaciones indica la ocupación máxima que permite la habitación h. En el caso de las reservas indica el tamaño mínimo de una habitación para poder ser asignada a la reserva h. Esto se usa en todos los niveles de implementación.

2.3.2 Start Day

La función start day tiene el siguiente formato ***(start_day ?r - reserva)***. Indica el día de inicio de la reserva. Esta función se usa en todos los niveles de la implementación de esta manera podemos comparar los días de las reservas para comprobar que no se solapen.

2.3.3 End Day

La función end day tiene el siguiente formato ***(end_day ?r - reserva)***. Indica el día de finalización de la reserva. Esta función se usa en todos los niveles de la implementación de esta manera podemos comparar los días de las reservas para comprobar que no se solapen.

2.3.4 Reservas Libres

La función reservas libres tiene el siguiente formato ***(reservas_libres)***. Indica la cantidad de reservas que aún no han sido asignadas a la planificación final. Esto será usado a partir de la primera extensión para calcular la métrica de optimización. Queremos minimizar el número de reservas que se quedan fuera de la planificación en todo caso.

2.3.5 Orientación

La función orientación tiene el siguiente formato ***(orientacion ?x - objeto)***. Indica la orientación tanto de las habitaciones cómo de las reservas. Para las habitaciones esta puede tomar cuatro valores distintos; 0 es Norte, 1 es Sur, 2 es Este y 3 es Oeste. Y para las reservas puede tomar cinco valores distintos; -1 es Indiferente, 0 es Norte, 1 es Sur, 2 es Este y 3 es Oeste. Este se usa únicamente en la segunda extensión donde tenemos que comprobar si las habitaciones que les asignamos a las reservas cumplen las preferencias de la reserva.

2.3.6 Preferencia Orientación No Servida

La función preferencia orientación no servida tiene el siguiente formato ***(pref_orientacion_no_servida)***. Indica la cantidad de preferencias sobre la orientación no satisfechas. Se inicializa el número de reservas. Este se únicamente en la segunda extensión para generar el número de preferencias que han sido servidas, y por lo tanto optimizar también por este valor.

2.3.7 Cantidad Reservas

La función tamaño habitación tiene el siguiente formato ***(cantidad_reservas)***. Indica la cantidad de reservas que están asignadas a alguna habitación. Esto se usa en el nivel tres y el nivel cuatro.

2.3.8 Multiplicación Conjunto Ocupación

La función tamaño habitación tiene el siguiente formato ***(xctj_ocupacion)***. Indica la suma de los porcentajes de ocupación para cada asignación entre una reserva y una habitación. Esto se usa en la extensión tres y la extensión cuatro para crear el valor de optimización, explicado posteriormente.

2.3.9 Habitaciones Unused

La función tamaño habitación tiene el siguiente formato ***(habitaciones_unused)***. Indica el número de habitaciones que no están asignadas a ninguna reserva. Esta se usa en la cuarta extensión puesto que tenemos que minimizar el número de habitaciones que se usan, por lo tanto maximizar el número de habitaciones que no se usan en todo el mes.

2.4 Operadores

Cada extensión efectúa diferentes cambios a los efectos de los operadores para acomodar las diferentes necesidades que tendrán. En caso que existan cambios, se verán reflejados en la explicación de completado de nivel de cada nivel.

A continuación explicaremos la base de estos operadores, a partir de la cual las diferentes extensiones las podrán manipular.

2.4.1 Reservar

El operador reserva generará una asignación a partir de una reserva y una habitación. En el nivel básico está es muy básica, solo comprueba que la reserva no solape con ninguna previamente asignada y que la reserva no se haya visitado previamente. Posteriormente, a partir de la primera extensión deberemos comprobar que la habitación no esté en la reserva, que no solape con ninguna habitación que esté asignada a esa reserva y que la combinación de reserva, habitación no haya sido previamente visitada.

En cuanto al efecto, tiene los cambios más radicales. Todos los diferentes niveles de completado de la planificación darán nuevos efectos que se deben cumplir. En cuanto al nivel básico, dado que no hay factor de optimización y su nivel de complejidad és muy bajo solo deberemos cambiar el estado de visitado de la reserva, dado que ya estará en la planificación. A partir de la primera extensión deberemos asegurarnos de que marcamos la combinación de reserva, habitación como visitada y reservada además de cambiar el valor de optimización para reflejar la habitación reservada.

A continuación hacemos un resumen de lo expuesto:

Parámetros	*?h - habitacion, ?r - reserva*
Precondición	***nivel basico:*** $\neg visitado(r) \ \wedge \ \neg solapa(r)$ ***extension 1..4:*** $\neg visitada(r, h) \ \wedge \ \neg reservada(r) \ \wedge \ \neg solapa(h, r)$
Efecto	***nivel basico:*** $visitado(r)$ ***extension 1..4:*** $visitado(r, h) \ \wedge \ asignado\,(r, h) \ \wedge \ reservada(r) \ \wedge$ $modificarValorOptimizacion()$

2.4.2 Eliminar

En cuanto al operador de eliminación es un operador muy básico que se usa en la extensión dos y tres. Este cogerá una asignación de reserva, habitación y la quitará del conjunto. De esta manera podemos tener una planificación que no incluya todas las reservas pero se hayan probado diferentes combinaciones. Por lo tanto el efecto de esta quitará la asignación del conjunto de la planificación, pero no modificará el visitado (el conjunto reserva, habitación quedará como visitado).

A continuación hacemos un resumen de lo expuesto:

Parámetros	*?h - habitacion, ?r - reserva*
Precondición	$habitaciAssignada(h, r)$
Efecto	$\neg asignado\,(r, h) \ \wedge \ \neg reservada(r)$ $\wedge \ modificarValorOptimizacion()$

2.4.3 Cambio Habitación

El operador de cambio de habitación se implementa a partir de la segunda extensión, y su propósito es permitir en una sola acción cambiar la habitación que se le asigna a una reserva, por otra. De no estar, necesitaríamos dos aplicaciones del operador reservar y una de eliminar para conseguir el mismo efecto.

En esencia, lo que hace cambio_habitación es seleccionar una asignación reserva - habitación, así como una segunda habitación que pudiera resultar más adecuada. Si efectivamente, esta habitación cumple una serie de requisitos que veremos más abajo, el efecto que tiene este operador sobre el estado de nuestro programa es eliminar la asignación que la reserva tenía a la primera habitación, y asignarla a la segunda. La reserva continúa estando reservada, pues solamente modificamos el dónde está.

A continuación hacemos un resumen de lo expuesto:

Parámetros	?h - habitacion, ?h1 - habitacion, ?r - reserva
Precondición	$assignada(h,\ r)\ \wedge\ \neg visitada(h1, r)\ \wedge\ \neg solapa(r)$
Efecto	$visitado(r,\ h1)\ \wedge\ asignado\ (r,\ h1)\ \wedge\ asignado\ (r,\ h1)\ \wedge$ $modificarValorOptimizacion()$

2.4.4 Cambio Reserva

Finalmente, contamos con un operador cambio reserva que opera de manera similar al operador visto anteriormente, cambio habitación. La única diferencia, es que en este caso en lugar de contar con dos habitaciones y una sola reserva, contamos con dos reservas y una sola habitación. El propósito de este operador es, entonces, facilitar la eliminación de una reserva por otra que también pueda ser adecuada para esa habitación en concreto.

Este operador se usa a partir de la extensión 3, y opera de la siguiente manera. Coge como parámetros una asignación y una reserva. De no estar asignada a ninguna otra habitación y cumplir una serie de requisitos logísticos (no superación de la capacidad máxima de la habitación, posibles solapamientos, etc), la reserva inicialmente asignada a esta habitación dejará de estar reservada, y quien pasará a estarlo es esta segunda reserva.

Exponemos en la siguiente tabla un resumen del funcionamiento del operador:

Parámetros	?h - habitación, ?r - reserva, ?r1 - reserva
Precondición	$\neg visitada(?\,r1)\ \wedge\ assignada(h,\ r)\ \wedge)$ $\neg visitada(?\,h\,?\,r1)\ \wedge\ tamano(?\,h)\ >=\ tamano(?\,r1)$
Efecto	$visitada(?\,r1) \wedge\ \neg reservada(?\,r)\ \wedge$ $\neg assignada(?\,h\,?\,r)\ \wedge\ assignada(?\,h\,?\,r1)$

3 PROBLEMAS

3.1 Estado Inicial

El estado inicial, en general, de los diferentes niveles será compuesto de una serie de reservas y una serie de habitaciones. Las reservas deberán ir acompañadas de el día en el que empezará y el día en el

que acabará esta reserva. Además tanto las habitaciones como las reservas deberán ir acompañadas de un tamaño, en el caso de la habitación indicará la capacidad máxima y en el caso de la reserva la cantidad de personas.

También hay una serie de especificaciones tanto del estado inicial como del estado final por niveles:

- **Nivel Básico:** En el caso del nivel básico no habrá ningún ajuste al estado inicial.
- **Extensión 1:** En el caso de la primera extensión no habrá ningún ajuste al estado inicial.
- **Extensión 2:** En el caso de la segunda extensión implementamos una nueva preferencia para el usuario, siendo esta la orientación preferente de la habitación. En este caso pues el estado inicial también deberá incluir las preferencias de orientación de los usuarios y las diferentes orientaciones de la habitación.
- **Extensión 3:** En el caso de la tercera extensión no habrá ningún ajuste al estado inicial.
- **Extensión 4:** En el caso de la cuarta extensión no habrá ningún ajuste al estado inicial.

3.2 Estado Final

El estado final del problema (exceptuando el nivel básico que veremos a continuación) será una asignación de las reservas de manera que optimicemos la heurística de cada nivel.
También hay una serie de especificaciones tanto del estado inicial como del estado final por niveles:

- **Nivel Básico:** El estado final del nivel básico generará una solución solo en el caso de que esta exista (i.e. todas las reservas pueden ser asignadas). En caso que una asignación completa de las habitaciones no exista, no habrá solución.
- **Extensión 1:** La diferencia entre el nivel básico y la extensión 1 es que en esta se podrá tener una planificación donde no todas las reservas estén asignadas. Optimizaremos el número de reservas que quedaràn asignadas a la planificación final.
- **Extensión 2:** El estado final de la segunda extensión tendrá una asignación de reservas sin solaparse usando una ponderación entre el criterio de la primera extensión y la cantidad de reservas que tienen una habitación con su orientación preferente.
- **Extensión 3:** El estado final de la tercera extensión cambiará la forma de ponderación de la heurística. Pasaremos a considerar no solamente el número de reservas que quedan asignadas al final, sino también cómo de ajustadas quedan estas reservas a las habitaciones donde se asignan. De modo que en nuestra métrica optimizaremos la combinación de ambos factores.
- **Extensión 4:** En cuanto al estado final también cambiaremos la forma de ponderación de la heurística. Basándonos en la heurística implementada para la tercera extensión, ésta última se limita a añadir un factor más al conjunto de aspectos a optimizar. Siendo éste el número de habitaciones que han sido ocupadas alguna vez a lo largo del mes. Debido a que buscamos maximizar, lo que queremos optimizar es el número de habitaciones que no se han ocupado ni una vez.

3.3 Optimizaciones

3.3.1 Extensión 1

En la primera optimización la métrica es bastante sencilla. Solo tenemos un parámetro que queremos minimizar, esto es el número de reservas que quedarán libres. Por lo tanto empezaremos con el número total de reservas y cada vez que una nueva reserva quede asignada restamos uno a la función y cuando quitemos una reserva del conjunto añadiremos uno. Por lo tanto encontraremos el conjunto de asignaciones con menor reservas.

3.3.2 Extensión 2

En la segunda extensión optimizamos tanto por reservas como por preferencias. En cuanto a las reservas lo calcularemos de la misma forma que lo hemos calculado en la primera extensión. La cantidad de reservas que siguen las preferencias del usuario las contemplaremos de la misma manera. Empezaremos con la función ***pref_orient_no_servida*** al número de reservas y la iremos restando en cuando asignemos una habitación que sirva las reservas con una habitación que cumpla las preferencias del usuario o el usuario no haya indicado preferencia en orientación. Por lo tanto, también deberemos restar uno de la función ***pref_orient_no_servida*** si eliminamos una reserva que no tenga preferencias o estemos eliminando una reserva que estaba asignada a habitación que cumplia las preferencias de orientación.

En este caso también deberemos hacer una ponderación entre las dos cantidades que queremos minimizar. Le daremos más importancia a que más reservas se puedan crear que a la asignación de reservas que deberemos tener. Por esta razón deberemos pensar en una optimización que pondere estas dos de tal manera que nos sea preferente tener más reservas asignadas. Hemos hecho algunas pruebas que han indicado que la mejor ponderación entre una y es un ratio de uno a tres, donde ponderamos tres veces más alto la importancia de tener menos reservas libres que la preferencia de orientación. Claramente hay un rango, que para problemas más grandes se podría acortar pero con las limitaciones de este trabajo hemos visto que este ratio nos da un resultado óptimo en la mayoría de los casos.

3.3.3 Extensión 3

En ésta extensión, se nos pide encontrar una solución que optimice tanto el número de reservas que terminan siendo asignadas, como las plazas vacías que pueda dejar una reserva que ha sido asignada a una habitación de mayor capacidad.

En esencia, añadiremos al parámetro que controla las reservas que se asignan y que ya hemos visto en el apartado 3.3.1, uno nuevo. Éste parámetro llevará un control del sumatorio de porcentajes de ocupación de una reserva sobre una habitación que se le asigna. Entendemos como porcentaje de ocupación, el tamaño de la reserva sobre la capacidad máxima de la habitación, multiplicado por cien. De modo que intuitivamente, se trata de un valor entre 0 y 100. Y cuanto más cercano sea dicho valor al 100, más se ajustará la reserva a esa habitación.

Por lo tanto, inicialmente el valor de esta variable será 0, y a medida que vayamos añadiendo reservas, iremos calculando su porcentaje de ocupación y sumándole. Paralelamente, un cambio de reserva provoca la resta del porcentaje de ocupación de la reserva eliminada, y la adición del de la nueva. Y en cuanto al cambio de habitación, lo que se ve modificado es el dividendo en función del cual calculamos el porcentaje de ocupación de la reserva que estamos asignando a una habitación distinta. Esto es, restamos el tamaño dividido por la capacidad de la habitación inicial, y lo sustituiremos por la división entre ese mismo tamaño y la nueva habitación.

Finalmente, este valor será dividido por el número de reservas asignadas, para conseguir así una media de ésta ocupación. Media que, junto al número de reservas asignadas sobre las totales de las que disponemos, deseamos maximizar por tal de conseguir una solución donde se asignen todas las reservas posibles, y lo hagan a las habitaciones que más se ajusten a ellas.

3.3.4 Extensión 4

En la cuarta y última extensión de éste trabajo, se nos pide optimizar un nuevo factor, aparte de los que añadimos en las extensiones 1 y 3. Éste factor es el número de habitaciones usadas a lo largo del mes. Deseamos que éste número sea lo más pequeño posible, de modo que no será aceptable aquella solución que haga uso de *n* habitaciones, si existen soluciones viables que usen un número menor.

Por tal de llevar a cabo dicha optimización, y teniendo en cuenta que en la extensión tres hemos aborado el problema como uno de maximización de parámetros, lo primero que hacemos es declarar uno nuevo llamado *habitaciones_unused*, que determina a cuántas habitaciones no se le ha asignado ni una sola reserva a lo largo del mes. Éste valor se inicializará al número de habitaciones, e irá aumentando o disminuyendo en una unidad, a medida que vayamos aplicando los operadores. En el caso de reservar, si usamos una nueva habitación para asignar la reserva, decrementaremos el contador. Al hacer un cambio de reserva, el valor no se ve modificado en absoluto, puesto que en la asignación el único factor que varía es el de la reserva, y no el de la habitación. Finalmente, al aplicarse el operador de cambio de reserva, tendremos en cuenta el estado de la habitación que abandona nuestra reserva, así como el de la nueva que se le asigna y modificaremos el valor acorde con las circunstancias.

Es importante mencionar que en esta extensión hace falta añadir también un predicado que determine si efectivamente una habitación ha sido o no usada. Evidentemente, todas empiezan como no usadas, pero en los efectos de nuestras acciones, se irán usando, cuando tengan al menos una reserva asignada. Pueden retomar el valor falso si hacemos un cambio_habitación sobre una habitación que contaba con una única reserva, que ahora nos llevamos. Usaremos la información que nos proporciona este predicado para efectuar los cambios necesarios en nuestro parámetro.

Finalmente, en la métrica nos limitamos a añadir al conjunto de parámetros, el valor obtenido en la variable, en un estado solución, y a dividirlo por el número de habitaciones del que disponíamos. Como maximizamos, el resultado más óptimo será aquel que reserve el máximo número de reservas posibles, que aparte de ajustarse al máximo a las capacidades de las habitaciones, también se repartirán entre el menor número de habitaciones disponibles (maximizando las habitaciones sin usar).

4 NIVEL BÁSICO

En el nivel básico debíamos implementar un planificador que usase todas las reservas. En el caso que alguna de las reservas quede “excluida” no habrá planificación válida y por lo tanto nuestro sistema no debería encontrar ninguna planificación. Además deberemos seguir la restricción de que dos reservas no pueden solaparse entre sí.

4.1 Completado de Nivel

Con tal de implementar este nivel y usando las variables, funciones y predicados explicados anteriormente. A continuación veremos la implementación de esta en detalle.

Como hemos explicado anteriormente esta primera iteración del código es un sistema de planificación bastante sencillo. La cuestión más importante en este paso era crear un sistema extensible que nos permitiese la posterior manipulación del mismo para extender esta práctica con facilidad. Por lo tanto el código de este nivel es sencillo y conciso.

Como indicamos en la explicación del dominio, para esta sección y realmente para todas las posteriores usaremos las variables ***habitación*** y ***reserva***. En cuanto a funciones en este caso solo nos hará falta ***tamano***, ***start_day*** y ***end_day*** y para predicados ***visitada***.

Nos encontramos con un problema de síntesis dado que hay múltiples combinaciones de soluciones y estas no están acotadas. Por lo tanto crearemos una solución empezando por un conjunto de asignaciones de reservas a habitaciones vacías. Inicialmente usaremos una única acción ***reservar***, los parámetros, las precondiciones y los efectos están explicados anteriormente y no hay cambios significativos a este.

Se usará una única acción para este nivel dado que no tenemos que generar conjuntos de reservas que no usen alguna reserva. Por lo tanto con esta acción comprobaremos todas las combinaciones posibles que usen todas las reservas.

A continuación veremos los problemas (juegos de pruebas) que usaremos para comprobar la validez de este programa.

4.2 Juego de Prueba 1

4.2.1 Descripción

En este juego de prueba veremos una asignación posible. En este caso ninguna reserva se solapará con otra y por lo tanto nos generará una solución. Posteriormente hablaremos más del output que esperamos de esta elección. Además en este juego de pruebas tendremos dos tipos de reservas distintas. Unas tendrán un tamaño de dos personas y las otras un tamaño de una persona. Tendremos dos habitaciones, una de una persona y otra de dos. Por lo tanto nuestro juego de prueba también estará comprobando la correctitud de nuestro sistema en casos de reservas que se solapan peró que pueden ser acomodadas en diferentes habitaciones.

4.2.2 Elección

```
(define (problem nivel_basico) (:domain hotel)
   (:objects
       h0 h1 - habitacion
       r1 r2 r3 r4 r5 r6 r7 r8 r9 - reserva
       r12 r22 r32 r42 r52 r62 r72 r82 r92 - reserva
   )
   (:init
       (= (start_day r1) 1)
       (= (start_day r2) 2)
       (= (start_day r3) 3)
       (= (start_day r4) 4)
       (= (start_day r5) 5)
       (= (start_day r6) 6)
       (= (start_day r7) 7)
       (= (start_day r8) 8)
       (= (start_day r9) 9)
       (= (start_day r12) 1)
       (= (start_day r22) 2)
       (= (start_day r32) 3)
```

```
(= (start_day r42) 4)
(= (start_day r52) 5)
(= (start_day r62) 6)
(= (start_day r72) 7)
(= (start_day r82) 8)
(= (start_day r92) 9)

(= (end_day r1) 1)
(= (end_day r2) 2)
(= (end_day r3) 3)
(= (end_day r4) 4)
(= (end_day r5) 5)
(= (end_day r6) 6)
(= (end_day r7) 7)
(= (end_day r8) 8)
(= (end_day r9) 9)
(= (end_day r12) 1)
(= (end_day r22) 2)
(= (end_day r32) 3)
(= (end_day r42) 4)
(= (end_day r52) 5)
(= (end_day r62) 6)
(= (end_day r72) 7)
(= (end_day r82) 8)
(= (end_day r92) 9)

;(reservado r112 d1)
(= (tamano h0) 2)
(= (tamano h1) 1)
(= (tamano r1) 1)
(= (tamano r2) 1)
(= (tamano r3) 1)
(= (tamano r4) 1)
(= (tamano r5) 1)
(= (tamano r6) 1)
(= (tamano r7) 1)
(= (tamano r8) 1)
(= (tamano r9) 1)
(= (tamano r12) 2)
(= (tamano r22) 2)
(= (tamano r32) 2)
(= (tamano r42) 2)
(= (tamano r52) 2)
(= (tamano r62) 2)
(= (tamano r72) 2)
```

```
        (= (tamano r82) 2)
        (= (tamano r92) 2)
    )

    (:goal (or (forall (?res - reserva) (visitada ?res))))
)
```

4.2.3 Solución Esperada

En este juego de prueba esperamos una asignación total de las habitaciones. Todas las reservas con identificador r*2 deberán ir a h0 mientras que las reservas con identificador r* deberán ir a h1. No hay ninguna reserva que se solapa por lo tanto nos debería generar una posible planificación de las reservas.

4.2.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'NIVEL_BASICO' defined
 ... done.


translating negated cond for predicate VISITADA
no metric specified.

ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

advancing to goal distance:   18
                              17
                              16
                              15
                              14
                              13
                              12
                              11
                              10
                               9
                               8
                               7
                               6
                               5
                               4
                               3
                               2
                               1
                               0

ff: found legal plan as follows
step    0: RESERVAR H1 R9
        1: RESERVAR H0 R92
        2: RESERVAR H1 R8
        3: RESERVAR H0 R82
        4: RESERVAR H1 R7
        5: RESERVAR H0 R72
        6: RESERVAR H1 R6
        7: RESERVAR H0 R62
        8: RESERVAR H1 R5
```

```
        9: RESERVAR H0 R52
       10: RESERVAR H1 R4
       11: RESERVAR H0 R42
       12: RESERVAR H1 R3
       13: RESERVAR H0 R32
       14: RESERVAR H1 R2
       15: RESERVAR H0 R22
       16: RESERVAR H1 R1
       17: RESERVAR H0 R12

time spent:    0.00 seconds instantiating 0 easy, 27 hard action templates
               0.00 seconds reachability analysis, yielding 36 facts and 27 actions
               0.00 seconds creating final representation with 36 relevant facts, 0
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 91 states, to a max depth of 0
               0.00 seconds total time
```

Asignaciones Finales:
H0: r1, r2, r3, r4, r5, r6, r7, r8, r9
H1: r12, r22, r32, r42, r52, r62, r72, r82, r92

4.3 Juego de Prueba 2

4.3.1 Descripción

En este juego de pruebas vamos a implementar un problema donde haya alguna reserva que se solape y por lo tanto no pueda ocurrir la asignación de reservas prevista. Usaremos como base el mismo juego de pruebas visto en el primer juego de pruebas y cambiaremos el día de finalización de las reservas para que la primera reserva se solape con la segunda.

4.3.2 Elección

```
(define (problem nivel_basico) (:domain hotel)
   (:objects
       h0 h1 - habitacion
       r1 r2 r3 r4 r5 r6 r7 r8 r9 - reserva
       r12 r22 r32 r42 r52 r62 r72 r82 r92 - reserva
   )
   (:init
       (= (start_day r1) 1)
       (= (start_day r2) 2)
       (= (start_day r3) 3)
       (= (start_day r4) 4)
       (= (start_day r5) 5)
       (= (start_day r6) 6)
       (= (start_day r7) 7)
       (= (start_day r8) 8)
       (= (start_day r9) 9)
       (= (start_day r12) 1)
       (= (start_day r22) 2)
       (= (start_day r32) 3)
       (= (start_day r42) 4)
```

```
(= (start_day r52) 5)
(= (start_day r62) 6)
(= (start_day r72) 7)
(= (start_day r82) 8)
(= (start_day r92) 9)

(= (end_day r1) 2)
(= (end_day r2) 2)
(= (end_day r3) 3)
(= (end_day r4) 4)
(= (end_day r5) 5)
(= (end_day r6) 6)
(= (end_day r7) 7)
(= (end_day r8) 8)
(= (end_day r9) 9)
(= (end_day r12) 1)
(= (end_day r22) 2)
(= (end_day r32) 3)
(= (end_day r42) 4)
(= (end_day r52) 5)
(= (end_day r62) 6)
(= (end_day r72) 7)
(= (end_day r82) 8)
(= (end_day r92) 9)

;(reservado r112 d1)
(= (tamano h0) 2)
(= (tamano h1) 1)
(= (tamano r1) 1)
(= (tamano r2) 1)
(= (tamano r3) 1)
(= (tamano r4) 1)
(= (tamano r5) 1)
(= (tamano r6) 1)
(= (tamano r7) 1)
(= (tamano r8) 1)
(= (tamano r9) 1)
(= (tamano r12) 2)
(= (tamano r22) 2)
(= (tamano r32) 2)
(= (tamano r42) 2)
(= (tamano r52) 2)
(= (tamano r62) 2)
(= (tamano r72) 2)
(= (tamano r82) 2)
```

```
        (= (tamano r92) 2)
    )


    (:goal (or (forall (?res - reserva) (visitada ?res))))
)
```

4.3.3 Solución Esperada

En este caso no debería poder encontrar una solución viable para la planificación de todas estas reservas. Esto es porque no le damos ninguna vía de escape a quitar una reserva que se solape con otra. Por lo tanto debería darnos como resultado que no hay ninguna posible planificación.

4.3.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'NIVEL_BASICO' defined
 ... done.


translating negated cond for predicate VISITADA
ff: goal can be simplified to FALSE. No plan will solve it
```

Asignaciones Finales:
No hay asignaciones.

4.4 Juego de Prueba 3

4.4.1 Descripción

El problema siguiente ha sido obtenido mediante el generador. El generador de problemas aleatorios permite generar para el nivel básico, problemas de un tamaño dado por el usuario (número de habitaciones y número de reservas) con valores aleatorios para las fechas de inicio y finalización de las reservas y para los tamaños de las habitaciones y las reservas. En este caso, hemos generado un problema aleatorio con siete habitaciones y quince reservas. Para este nivel, es poco probable que se puedan generar problemas de gran tamaño que tengan solución, ya que si existe alguna reserva que no pueda ser asignada a ninguna habitación se considera que el problema no es solucionable, pero sí nos servirá para comprobar que la generación aleatoria de problemas para el nivel básico se realiza correctamente para ser usado como base para la generación de problemas de las siguientes extensiones.

4.4.2 Elección

```
(define (problem jp3) (:domain hotel)
      (:objects
            h0 h1 h2 h3 h4 h5 h6 - habitacion
            r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 - reserva
      )
      (:init
            (= (start_day r0) 13)
            (= (start_day r1) 21)
            (= (start_day r2) 29)
            (= (start_day r3) 27)
```

```
        (= (start_day r4) 24)
        (= (start_day r5) 23)
        (= (start_day r6) 26)
        (= (start_day r7) 29)
        (= (start_day r8) 6)
        (= (start_day r9) 8)
        (= (start_day r10) 17)
        (= (start_day r11) 20)
        (= (start_day r12) 10)
        (= (start_day r13) 19)
        (= (start_day r14) 22)
        (= (end_day r0) 27)
        (= (end_day r1) 26)
        (= (end_day r2) 30)
        (= (end_day r3) 29)
        (= (end_day r4) 25)
        (= (end_day r5) 24)
        (= (end_day r6) 29)
        (= (end_day r7) 30)
        (= (end_day r8) 12)
        (= (end_day r9) 18)
        (= (end_day r10) 21)
        (= (end_day r11) 21)
        (= (end_day r12) 23)
        (= (end_day r13) 30)
        (= (end_day r14) 23)
        (= (tamano h0) 3)
        (= (tamano h1) 3)
        (= (tamano h2) 1)
        (= (tamano h3) 1)
        (= (tamano h4) 3)
        (= (tamano h5) 1)
        (= (tamano h6) 1)
        (= (tamano r0) 1)
        (= (tamano r1) 2)
        (= (tamano r2) 2)
        (= (tamano r3) 2)
        (= (tamano r4) 2)
        (= (tamano r5) 1)
        (= (tamano r6) 3)
        (= (tamano r7) 1)
        (= (tamano r8) 2)
        (= (tamano r9) 2)
        (= (tamano r10) 3)
        (= (tamano r11) 3)
        (= (tamano r12) 3)
```

```
            (= (tamano r13) 2)
            (= (tamano r14) 3)
      )
      (:goal (or (forall (?res - reserva) (visitada ?res))))
)
```

4.4.3 Solución Esperada

Podemos ver que para este problema solo hay habitaciones de tamaño 1 y de tamaño 3, y reservas de tamaños 1, 2 y 3, por lo que todas las reservas de tamaño más grande o igual que 2 tendrán que asignarse a habitaciones de 3 plazas. De las 7 habitaciones existentes, solamente 3 de ellas tienen capacidad para 3 personas, por lo que si existen más de tres reservas con tamaño mayor que 1 que se solapan durante al menos una noche entonces la asignación será imposible y el problema no tendrá solución. Por ejemplo, se da esta situación con las reservas r10, r11, r12 y r13. Todas tienen tamaño mayor o igual a 2 e incluyen la noche del día 20 al día 21, por lo que el problema no puede tener solución.

4.4.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'GENERATED' defined
 ... done.


ff: goal can be simplified to FALSE. No plan will solve it
```

Asignaciones Finales:
No hay asignaciones.

5 EXTENSIÓN 1

En la primera extensión vamos a partir de la base creada en el nivel básico y añadir la posibilidad de crear planificaciones con alguna reserva no asignada. Las limitaciones propuestas en el nivel básico siguen existiendo por lo tanto no podremos asignar un grupo más grande a una habitación donde estos no quepan y no deberá haber solapamientos entre reservas en una misma habitación. Como pueden quedar reservas libres deberemos tener un criterio de optimización, en este caso se opta por minimizar el número de reservas sin asignar.

5.1 Completado de Nivel

La extensión de este nivel nos permite la reutilización del código ya generado en el primer apartado. Dado que en este caso usaremos más predicados y funciones deberemos cambiar la acción que ya teníamos para poder controlar estos. Además añadiremos una nueva acción eliminar.

No habrá ningún cambio en las variables que usemos para este apartado. En cuanto a predicados añadiremos ***reservada***, ***habitacion_asignada*** y ***habitacion_visitada*** explicados anteriormente estos nos permitirán la gestión de las reservas basadas en sus asignaciones a habitaciones en vez de

mediante reservas. El predicado visto en el nivel anterior aún nos hace falta, este mostrará que todas las habitaciones han sido visitadas en algún momento, y pues dándonos una solución. La eliminación de este implicaría la no solución ya que habría ciertas situaciones donde no visitamos cada habitación para cada reserva.

En cuanto a funciones nos hace falta añadir una que guarde el valor de nuestro criterio de optimización (i.e. la cantidad de reservas no asignadas) este será ***reservas_libres***. Este valor será modificado por las acciones de tal manera que si una reserva está en uso se restará uno del valor de ***reservas_libres***, y en caso de eliminar una reserva del conjunto pasará exactamente lo opuesto.

Dado que esta extensión si que puede tener planificaciones que no incluyan todas las reservas nos hará falta la incorporación de una nueva acción ***eliminar***. Esta nos permitirá quitar una reserva del mapa si hay otra reserva que requiere los días de una reserva ya asignada.

Además en esta fase también vemos que hay ciertos efectos de la acción ***reservar*** que no están en el nivel básico esto es una gestión de los nuevos predicados añadidos en esta extensión. Por lo tanto la acción ***reservar***, en el caso de que se cumpla la precondición deberá mantener el predicado ***habitacion_assignada*** al día poniendo las asignaciones que se hagan en esta. Deberá marcar una habitación como visitada en cuanto se le asigne esta y reservada si está en las reservas. Además de mantener los efectos que este ya tenía en el nivel básico. ***Eliminar*** hará exactamente lo explicado en el dominio.

5.2 Juego de Prueba 1

5.2.1 Descripción

En este juego de pruebas cojeremos ese presentado en el estado inicial (juego de pruebas 2) que nos ha provocado una no solución. A diferencia que en el nivel básico nos deberá generar una solución usando solo uno de los dos que se solapan en tiempo.

5.2.2 Elección

```
(define (problem jp1) (:domain hotel)
   (:objects
       h0 h1 - habitacion
       r1 r2 r3 r4 r5 r6 r7 r8 r9 - reserva
       r12 r22 r32 r42 r52 r62 r72 r82 r92 - reserva
   )
   (:init
       (= (reservas_libres) 18) ;num reservas totales

       (= (start_day r1) 1)
       (= (start_day r2) 2)
       (= (start_day r3) 3)
       (= (start_day r4) 4)
       (= (start_day r5) 5)
       (= (start_day r6) 6)
       (= (start_day r7) 7)
       (= (start_day r8) 8)
       (= (start_day r9) 9)
```

```
        (= (start_day r12) 1)
        (= (start_day r22) 2)
        (= (start_day r32) 3)
        (= (start_day r42) 4)
        (= (start_day r52) 5)
        (= (start_day r62) 6)
        (= (start_day r72) 7)
        (= (start_day r82) 8)
        (= (start_day r92) 9)

        (= (end_day r1) 2)
        (= (end_day r2) 2)
        (= (end_day r3) 3)
        (= (end_day r4) 4)
        (= (end_day r5) 5)
        (= (end_day r6) 6)
        (= (end_day r7) 7)
        (= (end_day r8) 8)
        (= (end_day r9) 9)
        (= (end_day r12) 1)
        (= (end_day r22) 2)
        (= (end_day r32) 3)
        (= (end_day r42) 4)
        (= (end_day r52) 5)
        (= (end_day r62) 6)
        (= (end_day r72) 7)
        (= (end_day r82) 8)
        (= (end_day r92) 9)

        ;(reservado r112 d1)
        (= (tamano h0) 2)
        (= (tamano h1) 1)
        (= (tamano r1) 1)
        (= (tamano r2) 1)
        (= (tamano r3) 1)
        (= (tamano r4) 1)
        (= (tamano r5) 1)
        (= (tamano r6) 1)
        (= (tamano r7) 1)
        (= (tamano r8) 1)
        (= (tamano r9) 1)
        (= (tamano r12) 2)
        (= (tamano r22) 2)
        (= (tamano r32) 2)
        (= (tamano r42) 2)
```

```
        (= (tamano r52) 2)
        (= (tamano r62) 2)
        (= (tamano r72) 2)
        (= (tamano r82) 2)
        (= (tamano r92) 2)
    )

    (:goal (or (forall (?res - reserva) (visitada ?res))))
    (:metric minimize (reservas_libres))
)
```

5.2.3 Solución Esperada

La solución final deberá ser una que tenga asignada o bien r1 o bien r2 a la habitación. En cuanto a las otras habitaciones deberán seguir de la misma manera como hemos explicado en el primer juego de pruebas del nivel básico. Las reservas r*2 deberán estar asignadas a la habitación H0 mientras que las identificadas como r* deberán ir asignadas a la habitación H1. Todas las reservas (exceptuando R1 o R2) estarán asignadas.

5.2.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'JP1' defined
 ... done.


translating negated cond for predicate VISITADA
translating negated cond for predicate HABITACION_ASSIGNADA
translating negated cond for predicate HABITACION_VISITADA
translating negated cond for predicate RESERVADA
warning: change on metric in wrong direction. no optimization done.

ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

advancing to goal distance:   18
                              17
                              16
                              15
                              14
                              13
                              12
                              11
                              10
                               9
                               8
                               7
                               6
                               5
                               4
                               3
                               2
                               1
                               0
```

```
ff: found legal plan as follows
step    0: RESERVAR H1 R9
        1: RESERVAR H1 R8
        2: RESERVAR H1 R7
        3: RESERVAR H1 R6
        4: RESERVAR H1 R5
        5: RESERVAR H1 R4
        6: RESERVAR H1 R3
        7: RESERVAR H1 R2
        8: ELIMINAR H1 R1 R2
        8: RESERVAR H1 R1
       10: RESERVAR H0 R92
       11: RESERVAR H0 R82
       12: RESERVAR H0 R72
       13: RESERVAR H0 R62
       14: RESERVAR H0 R52
       15: RESERVAR H0 R42
       16: RESERVAR H0 R32
       17: RESERVAR H0 R22
       18: RESERVAR H0 R12

time spent:    0.00 seconds instantiating 0 easy, 81 hard action templates
               0.00 seconds reachability analysis, yielding 198 facts and 72 actions
               0.00 seconds creating final representation with 180 relevant facts, 1
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.01 seconds searching, evaluating 217 states, to a max depth of 0
               0.01 seconds total time
```

Asignaciones Finales
H0: r1, r3, r4, r5, r6, r7, r8, r9
H1: r12, r22, r32, r42, r52, r62, r72, r82, r92

5.3 Juego de Prueba 2

5.3.1 Descripción

Este juego de pruebas tiene una habitación que se solapa con dos otras reservas. Las diferentes posibilidades que podría encontrar el sistema será o bien asignar la más larga o bien asignar las dos más cortas. De esta forma podremos comprobar el funcionamiento de la optimización en este problema.

Dado que hemos optado por minimizar en cuanto a reservas que no han sido reservadas en esa planificación este juego de pruebas deberá pues intentar minimizar el número de reservas que se quedan fuera de la planificación.

5.3.2 Elección

```
(define (problem jp1) (:domain hotel)
   (:objects
       h0 h1 - habitacion
       r1 r2 r3 - reserva
       r12 r22 r32  - reserva
   )
   (:init
       (= (reservas_libres) 6) ;num reservas totales
```

```
        (= (start_day r1) 1)
        (= (start_day r2) 2)
        (= (start_day r3) 3)
        (= (start_day r12) 1)
        (= (start_day r22) 2)
        (= (start_day r32) 3)

        (= (end_day r1) 3)
        (= (end_day r2) 2)
        (= (end_day r3) 3)
        (= (end_day r12) 1)
        (= (end_day r22) 2)
        (= (end_day r32) 3)

        (= (tamano h0) 2)
        (= (tamano h1) 1)
        (= (tamano r1) 1)
        (= (tamano r2) 1)
        (= (tamano r3) 1)
    )

    (:goal (or (forall (?res - reserva) (visitada ?res))))
    (:metric minimize (reservas_libres))
)
```

5.3.3 Solución Esperada

Como se ha discutido anteriormente este problema intenta minimizar el número de reservas que quedan fuera de la planificación por lo tanto nuestro conjunto final de reservas debería incluir ***r2*** y ***r3*** pero no ***r1***. De esta forma dos de las tres reservas quedan asignadas.

5.3.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'JP1' defined
 ... done.


translating negated cond for predicate VISITADA
translating negated cond for predicate HABITACION_ASSIGNADA
translating negated cond for predicate HABITACION_VISITADA
translating negated cond for predicate RESERVADA
warning: change on metric in wrong direction. no optimization done.

ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

advancing to goal distance:    6
                               5
```

```
                              4
                              3
                              2
                              1
                              0

ff: found legal plan as follows
step    0: RESERVAR H0 R12
        1: RESERVAR H1 R1
        2: ELIMINAR H1 R2 R1
        3: RESERVAR H1 R2
        4: RESERVAR H1 R3
        5: RESERVAR H0 R32
        6: RESERVAR H0 R22

time spent:    0.00 seconds instantiating 0 easy, 41 hard action templates
               0.00 seconds reachability analysis, yielding 66 facts and 36 actions
               0.00 seconds creating final representation with 60 relevant facts, 1
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 506 states, to a max depth of 0
               0.00 seconds total time
```

Asignaciones Finales
H0: r12, r22, r32
H1: r2, r3

5.4 Juego de Prueba 3

5.4.1 Descripción

El siguiente juego de prueba ha sido generado mediante el generador. Siguiendo el problema generado aleatoriamente para el nivel básico, se ha generado un problema aleatoriamente con 7 habitaciones y 15 reservas. Dado el tamaño del problema, es muy posible que se dé una situación en que hay alguna reserva que no se puede asignar. Esta prueba nos servirá para poder comparar el resultado con el que hemos obtenido en la prueba generada aleatoriamente del nivel básico.

5.4.2 Elección

```
(define (problem jp3) (:domain hotel)
      (:objects
            h0 h1 h2 h3 h4 h5 h6 - habitacion
            r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 - reserva
      )
      (:init
            (= (reservas_libres) 15)
            (= (start_day r0) 11)
            (= (start_day r1) 8)
            (= (start_day r2) 15)
            (= (start_day r3) 11)
            (= (start_day r4) 10)
            (= (start_day r5) 2)
            (= (start_day r6) 22)
            (= (start_day r7) 18)
            (= (start_day r8) 18)
```

```
        (= (start_day r9) 17)
        (= (start_day r10) 28)
        (= (start_day r11) 6)
        (= (start_day r12) 4)
        (= (start_day r13) 17)
        (= (start_day r14) 4)
        (= (end_day r0) 15)
        (= (end_day r1) 27)
        (= (end_day r2) 22)
        (= (end_day r3) 25)
        (= (end_day r4) 15)
        (= (end_day r5) 16)
        (= (end_day r6) 29)
        (= (end_day r7) 29)
        (= (end_day r8) 25)
        (= (end_day r9) 24)
        (= (end_day r10) 29)
        (= (end_day r11) 10)
        (= (end_day r12) 21)
        (= (end_day r13) 30)
        (= (end_day r14) 10)
        (= (tamano h0) 2)
        (= (tamano h1) 3)
        (= (tamano h2) 4)
        (= (tamano h3) 1)
        (= (tamano h4) 3)
        (= (tamano h5) 3)
        (= (tamano h6) 3)
        (= (tamano r0) 4)
        (= (tamano r1) 3)
        (= (tamano r2) 3)
        (= (tamano r3) 1)
        (= (tamano r4) 3)
        (= (tamano r5) 2)
        (= (tamano r6) 4)
        (= (tamano r7) 4)
        (= (tamano r8) 4)
        (= (tamano r9) 3)
        (= (tamano r10) 3)
        (= (tamano r11) 1)
        (= (tamano r12) 1)
        (= (tamano r13) 2)
        (= (tamano r14) 2)
    )
    (:goal (or (forall (?res - reserva) (visitada ?res))))
    (:metric minimize (reservas_libres))
```

```
)
```

5.4.3 Solución Esperada

Se puede observar que como pasaba en el caso del problema generado para el nivel básico, no se podrán asignar todas las reservas ya que sólamente se ha generado una habitación de tamaño 4 (h2) y hay al menos dos reservas de tamaño también 4 cuyas fechas están solapadas. Por ejemplo, existe un solapamiento entre el día 22 y el día 29 para las reservas r6 y r7, que tienen tamaño 4. En este caso, a diferencia del anterior, puede haber reservas sin asignar por lo que se espera que este problema, análogo al anterior, sí tenga una solución en este caso.

5.4.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
... done.
ff: parsing problem file
problem 'GENERATED' defined
... done.



no optimization required. skipping criterion.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on
1*g(s) + 5*h(s) where
   metric is  plan length

Cueing down from goal distance:   15 into depth [1]
                                  14            [1]
                                  13            [1]
                                  12            [1]
                                  11            [1]
                                  10            [1]
                                   9            [1]
                                   8            [1]
                                   7            [1][2]
                                   6            [1][2]
                                   5            [1][2]
                                   4            [1]
                                   3            [1][2]
```

```
                                  2            [1][2]
                                  1            [1]
                                  0
ff: found legal plan as follows

step    0: RESERVAR H6 R14
        1: RESERVAR H6 R13
        2: RESERVAR H5 R12
        3: RESERVAR H4 R11
        4: RESERVAR H5 R10
        5: RESERVAR H4 R9
        6: RESERVAR H1 R5
        7: RESERVAR H3 R3
        8: RESERVAR H2 R8
        9: ELIMINAR H2 R7 R8
       10: RESERVAR H2 R7
       11: ELIMINAR H2 R6 R7
       12: RESERVAR H2 R6
       13: ELIMINAR H5 R4 R12
       14: RESERVAR H2 R0
       15: RESERVAR H5 R4
       16: ELIMINAR H5 R2 R4
       17: RESERVAR H5 R2
       18: ELIMINAR H5 R1 R2
       19: RESERVAR H5 R1

time spent:    0.00 seconds instantiating 0 easy, 710 hard action
templates
               0.00 seconds reachability analysis, yielding 406 facts and
556 actions
               0.00 seconds creating final representation with 332
relevant facts, 1 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 53 states, to a max
depth of 2
               0.00 seconds total time
```

Asignaciones Finales:
H0:
H1: r5
H2: r6, r0
H3: r3
H4: r11, r9
H5: r10, r1
H6: r14, r13

6 EXTENSIÓN 2

Para la segunda extensión partiremos del código creado en la primera extensión y deberemos añadir la posibilidad de tener preferencias de orientación. Este puede ser Norte, Sur, Este y Oeste. Además deberemos modificar el criterio de optimización para incluir si la habitación donde se ha puesto una reserva cumple la preferencia de orientación del usuario. Esto no deberá contrarrestar el criterio de optimización de la primera extensión, sino que deberemos encontrar un equilibrio entre estos dos.

6.1 Completado de Nivel

Entre esta versión del código nos encontramos en una situación bastante similar a la de la primera extensión. Las únicas incorporaciones y cambios que vemos en este nivel són para incorporar las preferencias de orientación. Para esto vamos a incorporar las funciones ***orientacion*** que nos dira la preferencia de orientación de una reserva y la orientación de una habitación. Además incorporaremos otra función ***pref_orient_no_servida*** que nos contará las reservas que no estén en una habitación con sus preferencias para facilitar la optimización de esta.

Además vemos la necesidad de añadir una tercera acción que sea ***cambio_habitación***, se explica en más detalle en el dominio. Vemos la necesidad de esta acción dado que la acción ***eliminar*** mira únicamente si las reservas en una habitación se solapan. En el caso de cambio habitacion vamos a ver si una reserva puede cambiarse de habitación (no hay solapamientos) y que haya un incremento en nuestro nuevo parámetro de optimización (i.e. la habitación cumpla las preferencias que la otra no cumplía). Por lo tanto esta nueva acción nos va a permitir encontrar resultados más óptimos para nuestra nueva función de optimización.

En cuanto a las acciones previamente vistas, estas ahora van a tener que comprobar que la habitación cumpla las preferencias para actualizar la función ***pref_orient_no_servida*** que empezará con la suma de las reservas totales. Las acciones de ***reservar*** deberán restar una cada vez que la habitación que se le asigne a una reserva cumpla su preferencia de orientación. Mientras que en ***eliminar*** se sumará uno a ***pref_orient_no_servida*** si quitamos una reserva que previamente estaba en una habitación con su preferencia de orientación.

6.2 Juego de Prueba 1

6.2.1 Descripción

La primera prueba que realizaremos nos permitirá asegurar que cuando todas las reservas se puedan satisfacer en diferentes habitaciones elegiremos esa que cumpla la restricción de la preferencia de orientación. Para comprobar esto tendremos reservas que no se solapan con la misma preferencia de orientación. Además tendremos dos habitaciones con orientaciones diferentes, una de las cuales cumple las preferencias de las reservas y la otra que no.

De esta forma podremos ver el impacto que tiene nuestra métrica en cuanto a la decisión de las habitaciones.

6.2.2 Elección

```
(define (problem jp1) (:domain hotel)
   (:objects
       h0 h1 - habitacion
       r1 r2 r3 r4 r5 - reserva
   )
```

```
   (:init
       (= (reservas_libres) 5)
       (= (pref_orient_no_servida) 5)

       (= (start_day r1) 1)
       (= (start_day r2) 2)
       (= (start_day r3) 3)
       (= (start_day r4) 4)
       (= (start_day r5) 5)

       (= (end_day r1) 1)
       (= (end_day r2) 2)
       (= (end_day r3) 3)
       (= (end_day r4) 4)
       (= (end_day r5) 5)

       (= (tamano h0) 1)
       (= (tamano h1) 1)
       (= (tamano r1) 1)
       (= (tamano r2) 1)
       (= (tamano r3) 1)
       (= (tamano r4) 1)
       (= (tamano r5) 1)

       (= (orientacion h0) 0)
       (= (orientacion h1) 2)

       (= (orientacion r1) 0)
       (= (orientacion r2) 0)
       (= (orientacion r3) 0)
       (= (orientacion r4) 0)
       (= (orientacion r5) 0)
   )

   (:goal (or (forall (?res - reserva) (visitada ?res))))
   ;(:metric minimize (+ (dias_libres) (pref_orient_no_servida)))
   (:metric minimize (+ (pref_orient_no_servida) (* (reservas_libres) 3)))
   ;;; reservas libres en funcion de reservas
   ;;; pref_orient_no_servida esta en funcion de reservas
)
```

6.2.3 Solución Esperada

Esperamos que todas las reservas se puedan poner en una planificación, además nos encontramos en una situación donde pueden estar asignadas todas a una habitación o a la otra. Por lo tanto deberían quedar asignadas a H0 ya que cumple las preferencias de orientación.

6.2.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'JP1' defined
 ... done.


translating negated cond for predicate HABITACION_ASSIGNADA
translating negated cond for predicate HABITACION_VISITADA
translating negated cond for predicate VISITADA
translating negated cond for predicate RESERVADA
warning: change on metric in wrong direction. no optimization done.

ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

advancing to goal distance:    5
                               4
                               3
                               2
                               1
                               0

ff: found legal plan as follows
step    0: RESERVAR H1 R5
        1: CAMBIO_HABITACION H1 H0 R5
        2: RESERVAR H1 R4
        3: CAMBIO_HABITACION H1 H0 R4
        4: RESERVAR H1 R3
        5: CAMBIO_HABITACION H1 H0 R3
        6: RESERVAR H1 R2
        7: CAMBIO_HABITACION H1 H0 R2
        8: RESERVAR H1 R1
        9: CAMBIO_HABITACION H1 H0 R1

time spent:    0.00 seconds instantiating 0 easy, 25 hard action templates
               0.00 seconds reachability analysis, yielding 60 facts and 25 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
               0.00 seconds searching, evaluating 55 states, to a max depth of 0
               0.00 seconds total time
```

Asignaciones Finales:
H0: r1, r2, r3, r4, r5
H1: -

6.3 Juego de Prueba 2

6.3.1 Descripción

En el siguiente juego de pruebas probamos la asignación de habitaciones usando tanto preferencias de orientación y asignaciones solapadas. En este caso tendremos cuatro diferentes reservas con cuatro diferentes preferencias de orientación, estas se solapan entre sí. Además tendremos tres habitaciones con diferentes orientaciones, esto quiere decir que una de las habitaciones se deberá quedar fuera. En este juego de pruebas deberemos comprobar que funcione la optimización de tal forma que nos quedemos con la reserva cuya preferencia de orientación no se pueda cumplir.

6.3.2 Elección

```
(define (problem jp2) (:domain hotel)
   (:objects
       h0 h1 h2 - habitacion
       r1 r2 r3 r4 - reserva
   )
   (:init
       (= (reservas_libres) 5)
       (= (pref_orient_no_servida) 5)

       (= (start_day r1) 5)
       (= (start_day r2) 5)
       (= (start_day r3) 5)
       (= (start_day r4) 5)

       (= (end_day r1) 10)
       (= (end_day r2) 10)
       (= (end_day r3) 10)
       (= (end_day r4) 10)

       (= (tamano h0) 1)
       (= (tamano h1) 1)
       (= (tamano h2) 1)
       (= (tamano r1) 1)
       (= (tamano r2) 1)
       (= (tamano r3) 1)
       (= (tamano r4) 1)

       (= (orientacion h0) 0)
       (= (orientacion h1) 1)
       (= (orientacion h2) 2)

       ; n = 0 / s = 1 / e = 2 / o = 3
       (= (orientacion r1) 0)
       (= (orientacion r2) 1)
       (= (orientacion r3) 2)
       (= (orientacion r4) 3)
   )

   (:goal (or (forall (?res - reserva) (visitada ?res))))
   ;(:metric minimize (+ (dias_libres) (pref_orient_no_servida)))
   (:metric minimize (+ (pref_orient_no_servida) (* (reservas_libres) 3)))
   ;;; reservas libres en funcion de reservas
   ;;; pref_orient_no_servida esta en funcion de reservas
)
```

6.3.3 Solución Esperada

Con el juego de pruebas que hemos mostrado anteriormente vemos que obligatoriamente una de las habitaciones se va a tener que quedar fuera. Con la minimización que hemos propuesto debería quedarse fuera la reserva R4 ya que no hay ninguna habitación que tenga esa preferencia. Las otras reservas deberían quedar asignadas a las habitaciones que tengan la preferencia de orientación adecuada.

6.3.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'JP2' defined
 ... done.


translating negated cond for predicate HABITACION_ASSIGNADA
translating negated cond for predicate HABITACION_VISITADA
translating negated cond for predicate RESERVADA
warning: change on metric in wrong direction. no optimization done.

ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

advancing to goal distance:    4
                               3
                               2
                               1
                               0

ff: found legal plan as follows
step    0: RESERVAR H0 R4
        1: RESERVAR H1 R3
        2: CAMBIO_HABITACION H0 H2 R4
        3: CAMBIO_HABITACION H1 H0 R3
        4: CAMBIO_HABITACION H2 H1 R4
        5: CAMBIO_HABITACION H0 H2 R3
        6: ELIMINAR H1 R1 R4
        7: RESERVAR H1 R1
        8: CAMBIO_HABITACION H1 H0 R1
        9: RESERVAR H1 R2

time spent:    0.00 seconds instantiating 48 easy, 48 hard action templates
               0.00 seconds reachability analysis, yielding 60 facts and 96 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
               0.16 seconds searching, evaluating 11413 states, to a max depth of 0
               0.16 seconds total time
```

Asignaciones Finales:

H0: r1

H1: r2

H2: r3

6.4 Juego de Prueba 3

6.4.1 Descripción

El siguiente juego de prueba ha sido generado mediante el generador. Siguiendo el problema generado aleatoriamente para el nivel básico, se ha generado un problema con 7 habitaciones y 15 reservas.

6.4.2 Elección

```
(define (problem jp3) (:domain hotel)
      (:objects
            h0 h1 h2 h3 h4 h5 h6 - habitacion
            r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 - reserva
      )
      (:init
            (= (reservas_libres) 15)
            (= (start_day r0) 15)
            (= (start_day r1) 16)
            (= (start_day r2) 5)
            (= (start_day r3) 22)
            (= (start_day r4) 13)
            (= (start_day r5) 25)
            (= (start_day r6) 16)
            (= (start_day r7) 1)
            (= (start_day r8) 6)
            (= (start_day r9) 19)
            (= (start_day r10) 12)
            (= (start_day r11) 10)
            (= (start_day r12) 23)
            (= (start_day r13) 8)
            (= (start_day r14) 25)
            (= (end_day r0) 25)
            (= (end_day r1) 17)
            (= (end_day r2) 6)
            (= (end_day r3) 25)
            (= (end_day r4) 14)
            (= (end_day r5) 26)
            (= (end_day r6) 28)
            (= (end_day r7) 6)
            (= (end_day r8) 18)
            (= (end_day r9) 25)
            (= (end_day r10) 21)
            (= (end_day r11) 19)
            (= (end_day r12) 25)
            (= (end_day r13) 26)
            (= (end_day r14) 27)
            (= (tamano h0) 2)
            (= (tamano h1) 1)
            (= (tamano h2) 4)
            (= (tamano h3) 1)
```

```
            (= (tamano h4) 4)
            (= (tamano h5) 1)
            (= (tamano h6) 4)
            (= (tamano r0) 4)
            (= (tamano r1) 3)
            (= (tamano r2) 4)
            (= (tamano r3) 3)
            (= (tamano r4) 1)
            (= (tamano r5) 3)
            (= (tamano r6) 4)
            (= (tamano r7) 3)
            (= (tamano r8) 2)
            (= (tamano r9) 2)
            (= (tamano r10) 2)
            (= (tamano r11) 3)
            (= (tamano r12) 1)
            (= (tamano r13) 2)
            (= (tamano r14) 4)
            (= (pref_orient_no_servida) 17)
            (= (orientacion h0) 0)
            (= (orientacion h1) 3)
            (= (orientacion h2) 1)
            (= (orientacion h3) 0)
            (= (orientacion h4) 2)
            (= (orientacion h5) 2)
            (= (orientacion h6) 3)
            (= (orientacion r0) -1)
            (= (orientacion r1) 0)
            (= (orientacion r2) 2)
            (= (orientacion r3) 1)
            (= (orientacion r4) 3)
            (= (orientacion r5) 3)
            (= (orientacion r6) 0)
            (= (orientacion r7) -1)
            (= (orientacion r8) -1)
            (= (orientacion r9) 1)
            (= (orientacion r10) 2)
            (= (orientacion r11) 3)
            (= (orientacion r12) -1)
            (= (orientacion r13) 0)
            (= (orientacion r14) 0)
      )
      (:goal (or (forall (?res - reserva) (visitada ?res))))
      (:metric minimize (+ (pref_orient_no_servida) (* (reservas_libres)
3)))
)
```

6.4.3 Solución Esperada

Como que el criterio de optimización para esta extensión penaliza más el hecho de que haya reservas libres que no que haya reservas asignadas a habitaciones que no coinciden con la preferencia sobre la orientación de la habitación, es de esperar que para ciertas asignaciones, la orientación de la habitación no coincida con la orientación indicada como preferente en la reserva.

6.4.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
... done.
ff: parsing problem file
problem 'GENERATED' defined
... done.



no optimization required. skipping criterion.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on
1*g(s) + 5*h(s) where
   metric is  plan length

Cueing down from goal distance:   15 into depth [1]
                                  14            [1]
                                  13            [1]
                                  12            [1]
                                  11            [1]
                                  10            [1]
                                   9            [1]
                                   8            [1]
                                   7            [1][2]
                                   6            [1]
                                   5            [1][2]
                                   4            [1][2]
                                   3            [1][2]
                                   2            [1][2]
                                   1            [1]
                                   0
```

```
ff: found legal plan as follows

step    0: RESERVAR H6 R14
        1: RESERVAR H4 R13
        2: RESERVAR H5 R12
        3: RESERVAR H6 R11
        4: RESERVAR H0 R10
        5: RESERVAR H6 R7
        6: RESERVAR H5 R4
        7: RESERVAR H4 R2
        8: RESERVAR H2 R9
        9: ELIMINAR H4 R8 R13
       10: RESERVAR H2 R8
       11: RESERVAR H4 R6
       12: ELIMINAR H4 R5 R6
       13: RESERVAR H4 R5
       14: ELIMINAR H4 R3 R5
       15: RESERVAR H4 R3
       16: ELIMINAR H4 R0 R3
       17: RESERVAR H4 R1
       18: ELIMINAR H4 R0 R1
       19: RESERVAR H4 R0


time spent:    0.00 seconds instantiating 0 easy, 573 hard action
templates
               0.00 seconds reachability analysis, yielding 384 facts and
491 actions
               0.00 seconds creating final representation with 288
relevant facts, 2 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 53 states, to a max
depth of 2
               0.00 seconds total time
```

Asignaciones Finales
H0: r10
H1:
H2: r9, r8
H3:
H4: r2, r0
H5: r12, r4
H6: r14, r11, r7

7 EXTENSIÓN 3

En la tercera extensión de nuestro problema, el objetivo es combinar las modificaciones obtenidas en en la primera extensión, con la minimización del número de plazas desperdiciadas en cada reserva. Por lo tanto, queremos conseguir reservar el máximo de asignaciones posible, minimizando siempre la diferencia entre la capacidad de la habitación y la del tamaño de la reserva.

7.1 Completado de Nivel

Para implementar esta extensión entra en juego, la variable *xctj_ocupacion*, que se irá modificando a lo largo de la ejecución junto con la variable que añadimos para la primera extensión, *cantidad_reservas*. De modo que, junto con las demás funciones ya explicadas en los apartados anteriores, nuestro **:functions** queda de la siguiente manera:

```
(:functions
       (tamano ?x - object)
       (start_day ?r - reserva)                ; dia que desean empezar - FIJO
       (end_day ?r - reserva)                  ; dia que desean acabar - FIJO

       (cantidad_reservas)
       (xctj_ocupacion)
   )
```

Por lo que respecta a los predicados, no se ven alterados en esta extensión. Y en cuanto a las acciones, contamos con cambio reserva, que equivale al eliminar, pero que reserva automáticamente la reserva que había causado un conflicto.

En esencia, lo que pretendemos hacer para implementar esta extensión, es efectuar un cambio de habitación siempre que una reserva que cause conflicto vaya a dejar menos camas desocupadas, que la reserva inicialmente asignada. El resto de precondiciones en esta acción, no difieren de las que encontramos en la acción eliminar.

```
(:action cambio_reserva
       :parameters (?h - habitacion ?r - reserva ?r1 - reserva)
       :precondition (and
           (not (visitada ?r1))
           (habitacion_assignada ?h ?r)
           (not (habitacion_visitada ?h ?r1))
           (>= (tamano ?h) (tamano ?r1))
           (< (- (tamano ?h) (tamano ?r1)) (- (tamano ?h) (tamano ?r)))
           (or
               (and
                   (>= (end_day ?r) (start_day ?r1))
                   (<= (end_day ?r) (end_day ?r1))
               )
               (and
                   (>= (start_day ?r) (start_day ?r1))
                   (<= (start_day ?r) (end_day ?r1))
               )
```

```
                (and
                    (<= (start_day ?r) (start_day ?r1))
                    (>= (end_day ?r) (end_day ?r1))
                )
                (and
                    (>= (start_day ?r) (start_day ?r1))
                    (<= (end_day ?r) (end_day ?r1))
                )
            )
        )
        :effect (and
            (visitada ?r1)
            (not (reservada ?r))
            (reservada ?r1)
            (not (habitacion_assignada ?h ?r))
            (habitacion_assignada ?h ?r1)
            (decrease (xctj_ocupacion) (* (/ (tamano ?r) (tamano ?h)) 100))
            (increase (xctj_ocupacion) (* (/ (tamano ?r1) (tamano ?h)) 100))

        )
    )
```

Como podemos ver en la porción de código anterior, hacer un cambio de reserva provoca una modificación en el valor de *xctj_ocupación,* que elimina el porcentaje de la reserva eliminada, y justo después añade el de la nueva. Similarmente, añadimos los siguientes efectos a las acciones *reservar* y *cambio_habitacion*, respectivamente.

Nótese como en el caso de *reservar*, añadimos el porcentaje de ocupación de la nueva reserva, y en *cambio_habitacion* sustituimos en el dividendo la capacidad de la habitación antigua por el de la nueva. El porcentaje total será dividido entre el número de reservas asignadas en la métrica, donde buscamos maximizar este valor medio.

```
;acción reservar
(increase (xctj_ocupacion) (* (/ (tamano ?r) (tamano ?h)) 100))

;acción cambio_habitacion
(decrease (xctj_ocupacion) (* (/ (tamano ?r) (tamano ?h)) 100))
(increase (xctj_ocupacion) (* (/ (tamano ?r) (tamano ?h1)) 100))
```

7.2 Juego de Prueba 1

7.2.1 Descripción

El primer juego de pruebas presentado, tenemos 5 reservas y 4 habitaciones. Las reservas son para entre 1 y 4 personas. Y respecto a las habitaciones, tienen capacidades que van de 2 a 4. Además, las reservas se solapan completamente; es decir, que todas empiezan el día 1 y terminan el día 5.

El propósito principal de este juego de pruebas, es ver cómo se combinan ambos criterios en un solo problema. Por un lado, debido a la menor cantidad de habitaciones, tenemos que habrá al menos una

asignación que no podrá ser reservada. Y por otro lado, nos interesará ver cuál será la reserva, y cuantas camas habría dejado libres de haber sido elegida.

Vemos entonces que este juego pondrá a prueba los dos aspectos que trata de abordar esta extensión. Si hemos decidido hacer que todas las reservas sean exactamente para los mismos días, es para aislarla de aspectos externos, y ver cómo se comporta el programa cuando no le queda más remedio que repartirse las reservas entre las habitaciones a su disposición debido al solapamiento. En próximas extensiones pondremos ejemplos más generales, donde no todas las reservas se solapan, para ver que sigue funcionando perfectamente cuando añadimos características ajenas al objetivo de optimización de la extensión 3.

Finalmente también cabe destacar que para asegurarnos de que el programa no funciona por un golpe de suerte debido al orden en el que se escogen los pares habitación-reserva, nos hemos asegurado de que el tamaño de las reservas no siga un orden concreto.

7.2.2 Elección

```
(define (problem jp1) (:domain hotel)
   (:objects
       h1 h2 h3 h4 - habitacion
       r1  r2  r3  r4 r5 - reserva
      )
   (:init
       (= (cantidad_reservas) 0)
       (= (start_day r1) 1)
       (= (start_day r2) 1)
       (= (start_day r3) 1)
       (= (start_day r4) 1)
       (= (start_day r5) 1)

       (= ( end_day r1) 5)
       (= ( end_day r2) 5)
       (= ( end_day r3) 5)
       (= ( end_day r4) 5)
       (= ( end_day r5) 5)

       (= (tamano h1) 2)
       (= (tamano h2) 2)
       (= (tamano h3) 3)
       (= (tamano h4) 4)

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

```
  (:goal
      (or (forall (?res - reserva) (or (visitada ?res)))))
  )
```

7.2.3 Solución Esperada

Como podemos ver en el código proporcionado, las habitaciones h1 y h2 tienen una capacidad máxima de dos huéspedes, en h3 caben como máximo 3 personas, y en h4 caben 4. Intuitivamente podemos concluir que lo que esperamos ver en la solución final, es el siguiente:que la

- reserva r1 (3 huéspedes): habitación h3
- reserva r2 (4 huéspedes): habitación h4
- reserva r3 (2 huéspedes): habitaciones h1 o h2 (indistintamente)
- reserva r4 (1 huésped): sin asignar
- reserva r5 (2 huéspedes): habitaciones h1 o h2 (la complementaria a la que se le asigne a r3)

Si es r4 la reserva que debe quedarse sin asignar es porque a diferencia de las demás reservas, tiene un porcentaje de ocupación de como máximo el 50%, mientras que hay otras reservas con porcentajes máximos de hasta el 100% que por lo tanto nos interesa priorizar.

Asimismo, también es lógico que esperemos que se reserven el máximo de reservas posibles (cuatro) puesto que el hotel dispone de las habitaciones para ello y que reservara menos de cuatro no sería un resultado óptimo.

7.2.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'JP1' defined
 ... done.


translating negated cond for predicate HABITACION_ASSIGNADA
translating negated cond for predicate HABITACION_VISITADA
translating negated cond for predicate RESERVADA
warning: metric is no linear expression. defaulting to plan length.
no metric specified.

ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

advancing to goal distance:    5
                               4
                               3
                               2
                               1
```

```
                                0


ff: found legal plan as follows
step    0: RESERVAR H4 R1
        1: CAMBIO_RESERVA H4 R1 R2
        2: RESERVAR H3 R1
        3: RESERVAR H2 R3
        4: RESERVAR H1 R4
        5: CAMBIO_RESERVA H1 R4 R5

time spent:    0.00 seconds instantiating 0 easy, 49 hard action templates
               0.00 seconds reachability analysis, yielding 85 facts and 49 actions
               0.00 seconds creating final representation with 75 relevant facts, 2
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 46 states, to a max depth of 0
               0.00 seconds total time
```

Como vemos, hemos acertado con el resultado esperado. Se efectúan cuatro reservas, donde inicialmente r4 era una de ellas pero finalmente es sustituida por la reserva r5 puesto que nos ofrece un resultado mejor.

7.3 Juego de Prueba 2

7.3.1 Descripción

En este segundo juego de pruebas vamos a observar el comportamiento de nuestra tercera extensión en un problema más general. En este caso, contamos con cuatro habitaciones y nueve reservas distintas.

Esta vez, no todas las reservas se solapan, y nos vamos aprovechar de este hecho para observar en un mismo juego de pruebas, distintas situaciones. Ciertas reservas están pensadas para encajar perfectamente con las habitaciones disponibles, otras obligarán al programa a dejar cierto porcentaje de desocupación, y también veremos el caso opuesto; qué sucede cuando una sola reserva puede escoger entre todas las habitaciones, de todos los tamaños.

Pensamos que este juega de pruebas es adecuado dado que nos permite estudiar varios casos especiales que puedan darse, y nos permite ver cómo el programa los aborda.

Primero, con las reservas de tamaño exacto que encajan perfectamente con las habitaciones del juego de pruebas, queremos ver si efectivamente continúan asignándose correctamente en un problema más grande.

Por lo que respecta a las habitaciones que se solapan pero que no encajan a la perfección, queremos ver dos cosas; por un lado, cómo las asigna a las habitaciones de las que dispone, y por otro, qué hace con las reservas que no encajan perfectamente con las habitaciones restantes, y si es peor asignarlas con un bajo porcentaje de ocupación o no s asignarlas.

Finalmente, este juego de pruebas también nos permitirá ver cómo funciona nuestra implementación para el caso base en el que tenemos una sola reserva y todas las habitaciones, con sus distintas capacidades.

7.3.2 Elección

```
(define (problem jp1) (:domain hotel)
   (:objects
       h1 h2 h3 h4 - habitacion
       r1  r2  r3  r4  r5  r6  r7  r8 r11 - reserva
   )
   (:init
       (= (cantidad_reservas) 0)
       (= (xctj_ocupacion) 0)

       (= (start_day r1) 1)
       (= (start_day r2) 1)
       (= (start_day r3) 1)
       (= (start_day r4) 1)
       (= (start_day r5) 2)
       (= (start_day r6) 2)
       (= (start_day r7) 2)
       (= (start_day r8) 2)
       (= (start_day r11) 3)

        (= ( end_day r1) 1)
       (= ( end_day r2) 1)
       (= ( end_day r3) 1)
       (= ( end_day r4) 1)
       (= ( end_day r5) 2)
       (= ( end_day r6) 2)
       (= ( end_day r7) 2)
       (= ( end_day r8) 2)
       (= ( end_day r11) 3)

       (= (tamano h1) 1)
       (= (tamano h2) 2)
       (= (tamano h3) 3)
       (= (tamano h4) 4)

       (= (tamano r1) 3)
       (= (tamano r2) 2)
       (= (tamano r3) 1)
       (= (tamano r4) 4)
       (= (tamano r5) 2)
       (= (tamano r6) 1)
       (= (tamano r7) 1)
```

```
       (= (tamano r8) 3)
       (= (tamano r11) 3)
   )
  (:goal
       (or (forall (?res - reserva) (or (visitada ?res)))))
     (:metric maximize (+ (/ (xctj_ocupacion) (cantidad_reservas)) (/
(cantidad_reservas) 9))))
```

7.3.3 Solución Esperada

En este caso, esperamos encontrarnos con que las cuatro primeras habitaciones, que se solapan entre ellas, quedan asignadas a las habitaciones con la capacidad exacta. Es decir, la reserva r1 a h3, r2 con h2, r3 con h1 y finalmente r4 con h4.

Por lo que respecta a las siguientes cuatro reservas, nos encontramos con que r6 y r7 son para una sola persona, de modo que lo que esperamos ver es que, solo a una de las dos se le asigne la habitación h1. La otra, se verá obligada a ocupar la primera habitación no asignada, dejando un porcentaje de ocupación más alto. En este caso, como tenemos las reservas r5 y r8 para dos y tres personas respectivamente, la reserva que no sea asignada a la habitación h1, se verá obligada a quedar asignada en la habitación h4, dejando tres plazas libres. La otra opción sería que la reserva quedara sin asignar.

Finalmente, la reserva r11, tiene a su disposición las cuatro habitaciones debido a sus días de reserva, y esperamos que escoja la habitación h3, puesto que es la que mejor se ajusta a las necesidades de la reserva.

7.3.4 Resultado Obtenido

```
     (:metric maximize (+ (/ (xctj_ocupacion) (cantidad_reservas)) (/
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'JP1' defined
 ... done.

translating negated cond for predicate HABITACION_ASSIGNADA
translating negated cond for predicate HABITACION_VISITADA
translating negated cond for predicate RESERVADA
warning: metric is no linear expression. defaulting to plan length.
no metric specified.

ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

advancing to goal distance:    9
                               8
                               7
```

```
                                6
                                5
                                4
                                3
                                2
                                1
                                0

ff: found legal plan as follows
step    0: RESERVAR H4 R11
        1: CAMBIO_HABITACION H4 H3 R11
        2: RESERVAR H4 R5
        3: CAMBIO_RESERVA H4 R5 R8
        4: RESERVAR H2 R5
        5: CAMBIO_HABITACION H4 H3 R8
        6: RESERVAR H4 R6
        7: CAMBIO_RESERVA H4 R6 R7
        8: RESERVAR H1 R6
        9: RESERVAR H4 R1
       10: CAMBIO_RESERVA H4 R1 R4
       11: RESERVAR H3 R1
       12: RESERVAR H2 R2
       13: RESERVAR H1 R3

time spent:    0.00 seconds instantiating 0 easy, 132 hard action templates
               0.00 seconds reachability analysis, yielding 149 facts and 132
actions
               0.00 seconds creating final representation with 127 relevant facts,
2 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 193 states, to a max depth of 0
               0.00 seconds total time
```

Efectivamente, el resultado final nos deja la siguiente configuración de habitaciones:
h1: r6 r3
h2: r5 r2
h3: r11 r1 r8
h4: r7 r4

Como esperábamos, la reserva r7 ha quedado asignada a la habitación cuatro, que era la primera disponible que dejaba el menor espacio vacío posible

7.4 Juego de Prueba 3

7.4.1 Descripción

El siguiente juego de prueba ha sido generado mediante el generador. Para esta extensión se ha generado un problema con 6 habitaciones y 9 reservas. Esta prueba nos servirá para probar el funcionamiento de la extensión 3 en un problema de mayor tamaño, comprobando que se minimiza el número de plazas desperdiciadas.

7.4.2 Elección

```
(define (problem jp3) (:domain hotel)
      (:objects
            h0 h1 h2 h3 h4 h5 - habitacion
            r0 r1 r2 r3 r4 r5 r6 r7 r8 - reserva
      )
      (:init
            (= (cantidad_reservas) 0)
            (= (xctj_ocupacion) 0)
            (= (start_day r0) 26)
            (= (start_day r1) 20)
            (= (start_day r2) 10)
            (= (start_day r3) 14)
            (= (start_day r4) 25)
            (= (start_day r5) 10)
            (= (start_day r6) 10)
            (= (start_day r7) 15)
            (= (start_day r8) 1)
            (= (end_day r0) 27)
            (= (end_day r1) 29)
            (= (end_day r2) 24)
            (= (end_day r3) 18)
            (= (end_day r4) 26)
            (= (end_day r5) 21)
            (= (end_day r6) 28)
            (= (end_day r7) 28)
            (= (end_day r8) 25)
            (= (tamano h0) 4)
            (= (tamano h1) 2)
            (= (tamano h2) 4)
            (= (tamano h3) 2)
            (= (tamano h4) 2)
            (= (tamano h5) 4)
            (= (tamano r0) 2)
            (= (tamano r1) 4)
            (= (tamano r2) 3)
            (= (tamano r3) 1)
            (= (tamano r4) 4)
            (= (tamano r5) 2)
            (= (tamano r6) 3)
```

```
            (= (tamano r7) 1)
            (= (tamano r8) 2)
      )
      (:goal (or (forall (?res - reserva) (visitada ?res))))
      (:metric maximize (+ (/ (xctj_ocupacion) (cantidad_reservas)) (/
(cantidad_reservas) 9)))
)
```

7.4.3 Solución Esperada

Es de esperar que por la mayoría de asignaciones entre una habitación y una reserva, el tamaño de la reserva se acerque lo máximo posible al tamaño de la habitación para evitar el desperdicio de plazas. En este caso, dado que solamente hay habitaciones de tamaño 2 y de tamaño 4, es de esperar que la mayoría de reservas de tamaño 1 y 2 se asignen a una habitación de tamaño 2 y no de tamaño 4.

7.4.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
... done.
ff: parsing problem file
problem 'GENERATED' defined
... done.



no optimization required. skipping criterion.


no metric specified. plan length assumed.

checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on
1*g(s) + 5*h(s) where
   metric is  plan length

Cueing down from goal distance:    9 into depth [1]
                                   8           [1]
                                   7           [1]
                                   6           [1]
                                   5           [1]
                                   4           [1]
                                   3           [1]
                                   2           [1]
                                   1           [1]
                                   0
```

```
ff: found legal plan as follows

step    0: RESERVAR H4 R8
        1: RESERVAR H3 R7
        2: RESERVAR H5 R6
        3: CAMBIO_RESERVA H3 R7 R5
        4: CAMBIO_RESERVA H5 R6 R4
        5: RESERVAR H1 R3
        6: RESERVAR H5 R2
        7: CAMBIO_RESERVA H5 R2 R1
        8: RESERVAR H4 R0


time spent:    0.00 seconds instantiating 0 easy, 159 hard action
templates
               0.00 seconds reachability analysis, yielding 198 facts and
51 actions
               0.00 seconds creating final representation with 144
relevant facts, 2 relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 10 states, to a max
depth of 1
               0.00 seconds total time
```

Asignaciones Finales
H0: r0
H1: r3
H2:
H3: r5
H4: r8
H5: r4, r1

Tal y como se esperaba, la mayoría de reservas han sido asignadas a habitaciones del mínimo tamaño posible. En este caso la única excepción ha sido la asignación de la reserva r0 (2 plazas) a la habitación h0 (4 plazas).

8 EXTENSIÓN 4

8.1 Completado de Nivel

En la cuarta y última extensión del problema, se nos pide basarnos en la extensión 3, y extenderla asegurando que se ocupan el mínimo de habitaciones durante el mes. De modo que pasado el mes buscamos una solución que, aparte de implementar todo lo que ya conseguimos en la extensión 3, haya gastado el menor número de habitaciones posible. Proponemos abordar el problema de la siguiente manera.

Primero declaramos la variable *habitaciones_unused* que estará inicializada al número de habitaciones del que dispongamos. A medida que apliquemos los distintos operadores que tenemos, esto es, *reservar, cambio_reserva* y *cambio_habitacion,* la iremos aumentando o disminuyendo acorde.

Al reservar, comprobaremos si la habitación que se asigna ya tenía alguna reserva, o si por lo contrario es la primera. En el caso de que sí lo sea, disminuimos en una unidad el valor en la variable, mientras que de no ser la primera, el valor no se verá afectado.

Similarmente, al hacer un cambio de habitación hay que tener en cuenta el estado final causado por llevarse una reserva de la primera habitación a la segunda. En el caso de que ésta fuera la única reserva en la primera habitación, volvemos a incrementar el valor de la variable dado que ahora queda vacía. Del mismo modo, decrementar o no en una unidad la variable dependerá de si nuestra reserva es la primera en ser asignada a la habitación nueva o no.

En el caso del tercer operador, como simplemente cambiamos una reserva por otra, el número de habitaciones no se verá afectado, por lo que nuestra variable tampoco.

Para facilitar la tarea de determinar si una habitación estaba usada o no, tenemos un nuevo predicado *usada* que determina si ha habido al menos una reserva en la habitación en cuestión.

8.2 Juego de Prueba 1

8.2.1 Descripción

En este primer juego de pruebas, vamos a experimentar qué ocurre cuando, a pesar de tener a disposición varias habitaciones, pueden ser asignadas todas a la misma habitación. Así que en este primer juego de pruebas, vamos a comprobar la eficacia de la nueva funcionalidad, en cierto modo aislada de los demás factores.

Por lo tanto, dispondremos de cuatro habitaciones idénticas, y de ocho reservas que puedan encajar perfectamente en cualquiera de ellas. Además, para asegurar que todas puedan optar a quedar asignadas a cualquier habitación, evitaremos que se solapen las reservas.

8.2.2 Elección

```
(define (problem jp1) (:domain hotel)
   (:objects
       h1  h2  h3  h4 - habitacion
       r1  r2  r3  r4 r5 r6 r7 r8 - reserva
   )
```

```
   (:init
       (= (cantidad_reservas) 0)
       (= (xctj_ocupacion) 0)
       (= (habitaciones_unused) 4)
       (= (start_day r1) 1)
       (= (start_day r2) 2)
       (= (start_day r3) 3)
       (= (start_day r4) 4)
       (= (start_day r5) 5)
       (= (start_day r6) 6)
       (= (start_day r7) 7)
       (= (start_day r8) 8)
       (= ( end_day r1) 1)
       (= ( end_day r2) 2)
       (= ( end_day r3) 3)
       (= ( end_day r4) 4)
       (= ( end_day r5) 5)
       (= ( end_day r6) 6)
       (= ( end_day r7) 7)
       (= ( end_day r8) 8)
       (= (tamano h1) 1)
       (= (tamano h2) 1)
       (= (tamano h3) 1)
       (= (tamano h4) 1)
       (= (tamano r1) 1)
       (= (tamano r2) 1)
       (= (tamano r3) 1)
       (= (tamano r4) 1)
       (= (tamano r5) 1)
       (= (tamano r6) 1)
       (= (tamano r7) 1)
       (= (tamano r8) 1)
   )
   (:goal(or (forall (?res - reserva)(or (visitada ?res)))))
   (:metric maximize (+ (+ (/ (xctj_ocupacion) (cantidad reservas))(/
(habitaciones_unused) 4)) (/ (cantidad_reservas) 8 ))))
```

8.2.3 Solución Esperada

Como ya hemos dicho, esperamos que las ocho reservas queden asignadas a la misma habitación. No obstante, lo que tampoco podemos asegurar es a qué habitación se asignarán las reservas.

8.2.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
```

```
problem 'JP1' defined
 ... done.


translating negated cond for predicate HABITACION_ASSIGNADA
translating negated cond for predicate HABITACION_VISITADA
translating negated cond for predicate VISITADA
translating negated cond for predicate RESERVADA
translating negated cond for predicate USADA
warning: metric is no linear expression. defaulting to plan length.
no metric specified.

task contains conditional effects. turning off state domination.



ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

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

ff: found legal plan as follows
step    0: RESERVAR H4 R8
        1: RESERVAR H4 R7
        2: RESERVAR H4 R6
        3: RESERVAR H4 R5
        4: RESERVAR H4 R4
        5: RESERVAR H4 R3
        6: RESERVAR H4 R2
        7: RESERVAR H4 R1

time spent:    0.00 seconds instantiating 0 easy, 32 hard action templates
               0.00 seconds reachability analysis, yielding 168 facts and 32
actions
               0.00 seconds creating final representation with 168 relevant facts,
3 relevant fluents
               0.00 seconds computing LNF
```

```
                    0.00 seconds building connectivity graph
                0.00 seconds searching, evaluating 145 states, to a max depth of 0
                0.00 seconds total time
```

Finalmente las reservas quedan asignadas de la siguiente manera. Como bien esperábamos, las 8 reservas han quedado asignadas a la misma habitación, que ha resultado ser h4. Como ya hemos dicho podría haber sido cualquier otra habitación, dado que eran todas iguales.

Asignación final
h1:
h2:
h3:
h4: r4 r8 r7 r1 r5 r3 r6 r2

8.3 Juego de Prueba 2

8.3.1 Descripción

En este segundo juego de pruebas, vamos a combinar la funcionalidad que añadimos en esta extensión, con las que vimos en la pasada. De esta manera conseguiremos ver cómo se desenvuelven cuando hay más factores en juego.

Más concretamente, lo que pretendemos ver en éste juego son básicamente dos cosas. Por un lado si es es mayor el coste de no asignar una reserva o el de reservar teniendo que estrenar una habitación nueva, y por otro, si cuando no le queda más remedio, prefiere asignar una reserva a una habitación con menor porcentaje de ocupación o bien hacer uso de una habitación nueva debido a que se ajusta más a las necesidades de la reserva.

Contamos pues, con tres reservas que no se solapan, y dos habitaciones. Los tamaños de las reservas son dos para dos personas, y una para tres. Las habitaciones, son de capacidades dos y tres.

8.3.2 Elección

```
(define (problem jp1) (:domain hotel)
   (:objects
       h1  h2 - habitacion
       r1  r2  r3 - reserva
   )
   (:init
       (= (cantidad_reservas) 0)
       (= (xctj_ocupacion) 0)
       (= (habitaciones_unused) 4)

       (= (start_day r1) 1)
       (= (start_day r2) 2)
       (= (start_day r3) 3)

       (= ( end_day r1) 1)
       (= ( end_day r2) 2)
       (= ( end_day r3) 3)
```

```
        (= (tamano h1) 2)
        (= (tamano h2) 3)

        (= (tamano r1) 3)
        (= (tamano r2) 2)
        (= (tamano r3) 2)
    )
    (:goal
        (or (forall (?res - reserva)
                (or
                    (visitada ?res)
                )
            )
        )
    )
    (:metric maximize (+ (+ (/ (xctj_ocupacion) (cantidad_reservas)) (/
(habitaciones_unused) 4)) (/ (cantidad_reservas) 8 )))
)
```

8.3.3 Solución Esperada

Lo que esperamos ver con este juego de pruebas es que lógicamente, la reserva para tres personas r1se almacene en la habitación h2, y que las otras reservas se asignen a la habitación que más se le ajusta, en lugar de asignarse también a la habitación h2. Como hay dos opciones viables, h1 y h3, esperamos que ambas reservas de tamaño 2 se asignen ambas a una de las dos habitaciones.

8.3.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'JP1' defined
 ... done.

translating negated cond for predicate HABITACION_ASSIGNADA
translating negated cond for predicate HABITACION_VISITADA
translating negated cond for predicate VISITADA
translating negated cond for predicate RESERVADA
translating negated cond for predicate USADA
warning: metric is no linear expression. defaulting to plan length.
no metric specified.

task contains conditional effects. turning off state domination.
```

```
ff: search configuration is weighted A* with weight 5. plan length
COST MINIMIZATION DONE (WITH cost-minimizing relaxed plans).

advancing to goal distance:    3
                               2
                               1
                               0

ff: found legal plan as follows
step    0: RESERVAR H3 R3
        1: RESERVAR H3 R2
        2: RESERVAR H2 R1

time spent:    0.00 seconds instantiating 0 easy, 9 hard action templates
               0.00 seconds reachability analysis, yielding 46 facts and 5 actions
               0.00 seconds creating final representation with 38 relevant facts, 3
relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 10 states, to a max depth of 0
               0.00 seconds total time
```

Como esperábamos la habitación h2 ha quedado con una sola reserva asignada r1, mientras que las otras dos se han visto asignadas a la habitación h3, como podrían haberse añadido a la h1. Lo importante, es que efectivamente se han asignado a la misma habitación.

8.4 Juego de Prueba 3

8.4.1 Descripción

El siguiente juego de prueba ha sido generado mediante el generador. Para esta extensión se ha generado un problema con 7 habitaciones y reservas. Esta prueba nos servirá para probar el funcionamiento de la extensión 3 en un problema de mayor tamaño, comprobando que se minimiza el número de habitaciones usadas y el desperdicio de plazas.

8.4.2 Elección

```
(define (problem generated) (:domain hotel)
      (:objects
            h0 h1 h2 h3 h4 h5 h6 - habitacion
            r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 - reserva
      )
      (:init
            (= (cantidad_reservas) 0)
            (= (xctj_ocupacion) 0)
            (= (habitaciones_unused) 7)
            (= (start_day r0) 20)
```

```
        (= (start_day r1) 23)
        (= (start_day r2) 23)
        (= (start_day r3) 15)
        (= (start_day r4) 27)
        (= (start_day r5) 24)
        (= (start_day r6) 3)
        (= (start_day r7) 29)
        (= (start_day r8) 17)
        (= (start_day r9) 18)
        (= (start_day r10) 24)
        (= (start_day r11) 9)
        (= (start_day r12) 25)
        (= (start_day r13) 25)
        (= (start_day r14) 11)
        (= (end_day r0) 27)
        (= (end_day r1) 27)
        (= (end_day r2) 25)
        (= (end_day r3) 17)
        (= (end_day r4) 30)
        (= (end_day r5) 27)
        (= (end_day r6) 30)
        (= (end_day r7) 30)
        (= (end_day r8) 18)
        (= (end_day r9) 26)
        (= (end_day r10) 27)
        (= (end_day r11) 24)
        (= (end_day r12) 27)
        (= (end_day r13) 27)
        (= (end_day r14) 27)
        (= (tamano h0) 1)
        (= (tamano h1) 3)
        (= (tamano h2) 3)
        (= (tamano h3) 2)
        (= (tamano h4) 3)
        (= (tamano h5) 4)
        (= (tamano h6) 1)
        (= (tamano r0) 3)
        (= (tamano r1) 2)
        (= (tamano r2) 3)
        (= (tamano r3) 1)
        (= (tamano r4) 4)
        (= (tamano r5) 2)
        (= (tamano r6) 2)
        (= (tamano r7) 1)
        (= (tamano r8) 1)
        (= (tamano r9) 4)
```

```
            (= (tamano r10) 1)
            (= (tamano r11) 2)
            (= (tamano r12) 1)
            (= (tamano r13) 1)
            (= (tamano r14) 3)
      )
      (:goal (or (forall (?res - reserva) (visitada ?res))))
      (:metric maximize (+ (+ (/ (xctj_ocupacion) (cantidad_reservas))
(/ (habitaciones_unused) 7)) (/ (cantidad_reservas) 15)))
)
```

8.4.3 Solución Esperada

Para este problema, es de esperar que la solución generada asigne las reservas de manera que parte de las habitaciones queden vacías, teniendo en evitar el desperdicio de plazas y minimizando el número de reservas sin asignar.

8.4.4 Resultado Obtenido

```
ff: parsing domain file
domain 'HOTEL' defined
 ... done.
ff: parsing problem file
problem 'GENERATED' defined
 ... done.



no optimization required. skipping criterion.


no metric specified. plan length assumed.

task contains conditional effects. turning off state domination.



checking for cyclic := effects --- OK.

ff: search configuration is EHC, if that fails then  best-first on
1*g(s) + 5*h(s) where
    metric is  plan length

Cueing down from goal distance:   15 into depth [1]
                                  14            [1]
                                  13            [1]
                                  12            [1]
```

```
                                    11            [1]
                                    10            [1]
                                     9            [1]
                                     8            [1]
                                     7            [1]
                                     6            [1]
                                     5            [1]
                                     4            [1]
                                     3            [1]
                                     2            [1]
                                     1            [1]
                                     0

ff: found legal plan as follows

step    0: RESERVAR H5 R14
        1: RESERVAR H6 R13
        2: CAMBIO_RESERVA H6 R13 R12
        3: RESERVAR H4 R11
        4: CAMBIO_RESERVA H6 R12 R10
        5: CAMBIO_RESERVA H5 R14 R9
        6: RESERVAR H6 R8
        7: RESERVAR H6 R7
        8: CAMBIO_RESERVA H4 R11 R6
        9: CAMBIO_RESERVA H4 R6 R5
       10: RESERVAR H5 R4
       11: CAMBIO_RESERVA H6 R8 R3
       12: CAMBIO_RESERVA H4 R5 R2
       13: RESERVAR H3 R1
       14: CAMBIO_RESERVA H4 R2 R0


time spent:    0.07 seconds instantiating 0 easy, 784 hard action
templates
               0.00 seconds reachability analysis, yielding 436 facts
and 784 actions
               0.00 seconds creating final representation with 378
relevant facts, 3 relevant fluents
               0.02 seconds computing LNF
               0.02 seconds building connectivity graph
               0.01 seconds searching, evaluating 16 states, to a max
depth of 1
               0.12 seconds total time
```

Asignaciones Finales
H0:

H1:
H2:
H3: r1
H4: r0
H5: r4, r9
H6: r10, r3, r7

Hemos obtenido un resultado esperado, ya que efectivamente buena parte de las habitaciones han quedado libres de reservas. Además todas las reservas se han asignado a una habitación de igual tamaño, por lo que no se ha desperdiciado ninguna plaza.

9 Generador de Juego de Pruebas

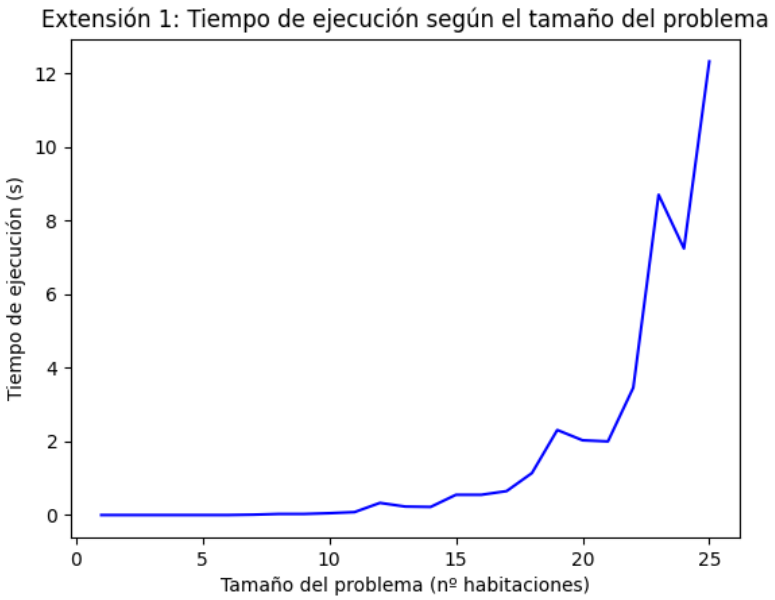
Hemos implementado un script en Python para generar juegos de prueba aleatorios. El script principal, *generator.py*, espera cuatro parámetros que se corresponden con la extensión, el número de habitaciones, el número de reservas y un último parámetro opcional para el nombre del fichero de salida. Este script genera un fichero con extensión *.pddl* que representa un problema de planificación que tiene el número de habitaciones y reservas indicados y además da valores aleatorios a las fechas de inicio y fin de las reservas, al tamaño de las habitaciones y de las reservas, a la orientación de la habitación, etc. También inicializa los valores para las funciones, el objetivo y las métricas en función de la extensión indicada. Se ha utilizado este script para generar un problema aleatorio para el nivel básico y cada una de las extensiones.

9.1 Tamaño de Problema Creciente

Para experimentar cómo evoluciona el tiempo de resolución con el tamaño del problema hemos implementado unos scripts en Python que usan reiteradamente el script anterior para generar problemas de tamaño creciente. De esa forma, *generateProblems.py* genera para cada extensión un problema con i = 1, 2, 3, … , N habitaciones, para una N dada en el fichero *constants.py*, y un número de reservas que depende de N. Otro script *run.py* se encarga de ejecutar todos los problemas con su dominio correspondiente y generar un fichero *.txt* con la salida de la ejecución. Finalmente, *generateResultsCSV.py* se encarga de leer estos ficheros con los resultados para obtener el tiempo de cada ejecución y escribirlo en un fichero .csv, y *plot.py* se encarga de leer los ficheros *.csv* y generar los gráficos que se encuentran a continuación mediante la librería *matplotlib*.

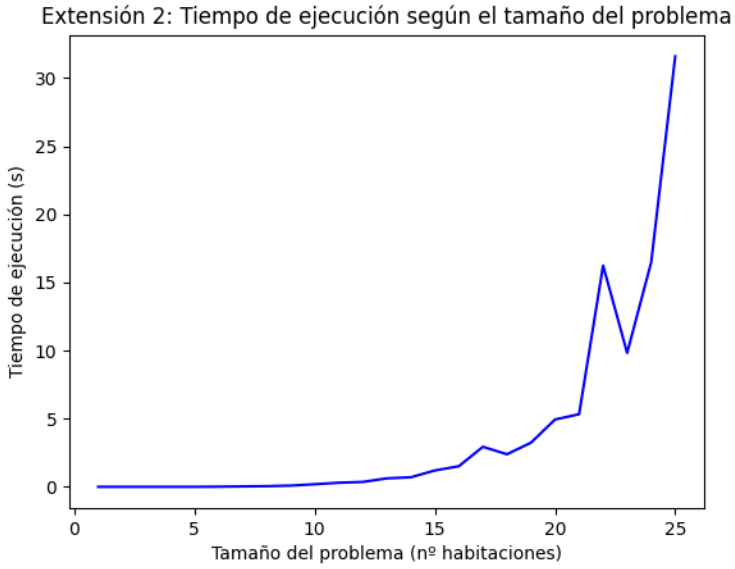
9.1.1 Extensión 1

Para la extensión 1 hemos generado cincuenta problemas con 1, 2, …, 25 habitaciones y cuatro reservas por cada habitación. En el siguiente gráfico pueden verse los resultados obtenidos: en el eje horizontal tenemos los distintos valores para el tamaño del problema (entre 1 y 25) y en el eje vertical tenemos el tiempo de ejecución resultante para cada tamaño. Se puede observar que el tiempo sigue una tendencia creciente en relación al tamaño del problema, acentuándose a partir de las 20 habitaciones y las 80 reservas.



9.1.2 Extensión 2

Para la extensión 2 hemos seguido el mismo procedimiento que para la extensión 1. Eso es, 25 problemas de entre 1 y 25 habitaciones con 4 reservas por habitación. El siguiente gráfico muestra los resultados obtenidos. Se puede apreciar que el tiempo sigue una clara tendencia creciente en relación al tamaño del problema, que se acentúa a partir de 20 habitaciones y 80 reservas, igual que para la extensión anterior.

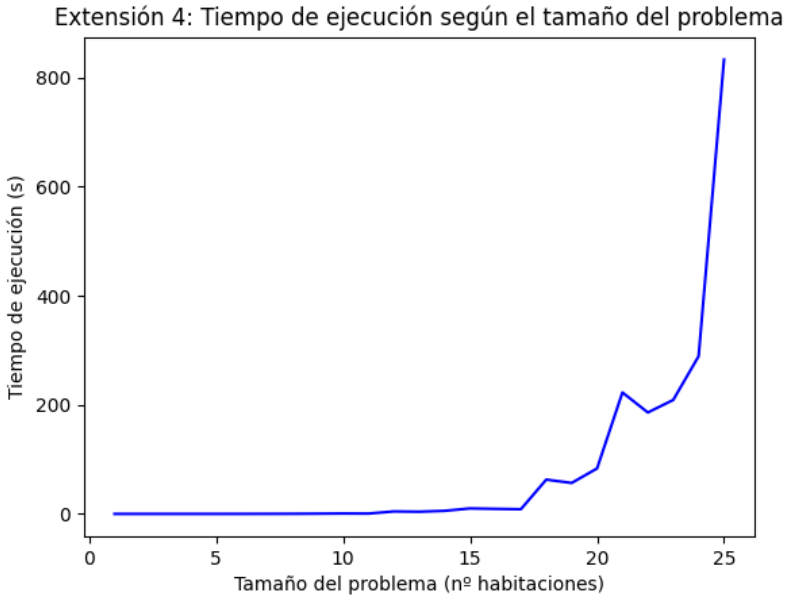


9.1.3 Extensión 3

Para la extensión 3 hemos generado problemas de entre 1 y 25 habitaciones con cuatro reservas por habitación. El siguiente gráfico muestra los resultados obtenidos. Se puede apreciar un fuerte crecimiento en el tiempo de ejecución a partir de las 15 habitaciones y 60 reservas.

9.1.4 Extensión 4

Para la extensión 4 hemos experimentado sobre problemas de tamaño entre 1 y 25 habitaciones, con 4 reservas por habitación. Una vez más, puede verse en la gráfica siguiente el resultado del experimento. Cabe destacar que el tiempo de ejecución aumenta drásticamente a partir de las 20 habitaciones y 80 reservas.



10 Conclusión

En éste trabajo sobre planificación, hemos hecho uso de la herramienta Metric FF, para conseguir ofrecer una serie de asignaciones de reservas a habitaciones de hoteles en base a la optimización de distintos parámetros. Hemos realizado esta práctica programando en pddl, un lenguaje que gracias a nuestro previo conocimiento adquirido en CLIPS no nos ha resultado difícil de entender.

Contábamos con una serie de extensiones cuya intención era optimizar una serie de parámetros, añadiendo ciertas características al problema, como la orientación de las habitaciones, su capacidad, el tamaño de las reservas, etc. La gran variedad de demandas en las extensiones nos ha permitido aplicar varios aspectos del lenguaje para su solución, así como abordar un problema de optimización de una forma muy distinta a como estamos acostumbrados. También nos ha permitido crear una serie de scripts para generar problemas de manera rápida y sencilla, ajustados a cada una de las extensiones implementadas.

Por lo que respecta a los juegos de pruebas, se proponen mostrar en cada caso cómo funcionan las implementaciones de las extensiones. La ejecución del programa para cada uno de los juegos de pruebas devuelve en todos los casos unos resultados bastante satisfactorios y siempre esperados. Y gracias al generador mencionado anteriormente, hemos podido comprobar cómo aumenta el tiempo de ejecución con el tamaño del problema, dado a la vastedad del espacio de búsqueda en los problemas grandes.

Si bien el lenguaje nos ha presentado ciertas dificultades en algunos momentos, hemos sabido superarlas y conseguir unos resultados satisfactorios, en los cuales se ve claramente que, para los problemas proporcionados, el programa es capaz de optimizar aquello que nos interese en cada momento.

Como posibles extensiones al trabajo, podríamos sugerir una extensión que unificase todas las preferencias que el usuario que hemos visto e incluso añadiera algunas más. Así que tendríamos un dominio encargado de encontrar la solución óptima para una serie de reservas con un intervalo de días concreto, un tamaño, así como una orientación u orientaciones preferentes.

Por otro lado, otro aspecto del cual nos hemos percatado es que no hacemos un control exhaustivo de errores; casos como qué ocurre cuando una reserva no podrá ser asignada a ninguna de las habitaciones disponibles por tener un tamaño demasiado grande, un rango de días incoherente o que supere la cantidad máxima de días que tenemos en cuenta. De modo que el funcionamiento de nuestra implementación depende en gran medida de que el input proporcionado sea correcto, y no caiga en ninguno de los errores mencionados.

En conclusión, podemos afirmar haber implementado correctamente todas las extensiones presentadas. Gracias a los juegos de pruebas, tanto los hechos a mano como los generados por nuestro script, nos demuestran que cada extensión optimiza adecuadamente los parámetros necesarios. Esto nos confirma que hemos declarado los predicados acertados, y que nuestras acciones exploran adecuadamente el espacio de búsqueda de cada problema que le damos como entrada.

11 Bibliografia

- https://planning.wiki/ref/pddl/domain
- http://editor.planning.domains/#
- https://github.com/tatsubori/Metric-FF