Dpto. Sistemas Informáticos y Computación
Escuela Técnica Superior de Ingeniería Informática
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Técnicas, Entornos y Aplicaciones de Inteligencia Artificial

Planificación

Evaluación de la Práctica

1. Introducción

En esta práctica haremos usos de algunos de los **planificadores** más conocidos que utilizan técnicas descritas en teoría.

Para ello definiremos un problema de planificación mediante el **lenguaje PDDL** (Planning Domain Definition Language), analizando su estructura y características principales. Este lenguaje se desarrolló en 1998 para la competición internacional de planificación (IPC, http://ipc.icaps-conference.org/) y, desde entonces, se ha utilizado como un estándar para generar una representación estándar y homogénea de un problema de planificación que pueda utilizarse por distintos planificadores.

PDDL ha evolucionado desde su creación y cada nueva versión extiende la anterior ofreciendo nuevas características y una mayor expresividad, aunque no todas *sobreviven*. A modo de resumen, las versiones de PDDL[1] son:

- PDDL 1 (1998). Representación proposicional basada en STRIPS y de acciones basadas en precondiciones/efectos. Se utilizan dos ficheros textuales: uno para el dominio de planificación con la definición de predicados y acciones; y otro para el problema de planificación (que se relaciona con un único dominio) con la definición de los objetos concretos, su estado inicial y objetivos.
- PDDL 2.1, con cinco niveles de expresividad (2001-02). Incluye acciones con duración y permite definir funciones numéricas para facilitar el manejo de recursos (uso de combustible, coste, utilidad de las acciones, etc.). También permite comparaciones en las precondiciones (<, >, <=, >=, =) y operadores de asignación en los efectos (:=, +=, -=, *=, /=). Adicionalmente se puede definir una función de optimización a maximizar/minimizar para que el planificador no sólo obtenga una solución sino también una orientada hacia la solución óptima.
- PDDL 2.2 (2003-04). Incluye la definición de predicados derivados (que ayudan a propagar nueva información; ej. si A está antes que B, y B antes que C, entonces A está antes que C) y literales exógenos anotados temporalmente (ej. un recurso está disponible sólo en una ventana de tiempo, independientemente de lo que se haga en el plan).
- PDDL 3.0 (2005-06). Da un mayor énfasis a la calidad del plan y en qué forma tiene. Se incluyen restricciones de trayectoria que deben cumplirse durante la ejecución del plan (ej. se debe pasar obligatoriamente por un estado dado, un estado tiene que ser anterior a otro, etc.) y la idea de preferencias, es decir restricciones aconsejables pero que se pueden violar con una penalización. De esta forma se pueden definir tanto restricciones de tipo *hard* como *soft*, lo que da una mayor variabilidad a los planes: la planificación no es sólo un problema de satisfactibilidad.
- PDDL 3.1 (2008-actualidad). Introduce variables de estado que reemplazan la representación booleana por una multi-estado mucho más compacta y eficiente (ej. estado_paquete={pendiente, enviado, recogido, perdido}).

[1] En el Poliformat de la asignatura se adjunta el documento explicativo de PDDL, donde aparece su gramática y se explican todos sus elementos.

2. Estructura de PDDL

El lenguaje PDDL2.1, que utilizaremos como ejemplo, tiene una estructura relativamente sencilla que comprende dos archivos de texto.

2.1.- Especificación del Dominio

En primer lugar se debe definir el archivo de dominio:

```
(define (domain <name>)
   (:requirements   <:req-1> … <:req-n>)          ; requisitos necesarios para entender el dominio
   (:types   <subtype-1>… <subtype-n>  –  <type-1> ……..  <type-n>)      ; subtipos y  tipos que se usan
   (:constants      <cons-1> ….  <cons-n>)                ; definición de constantes (si se van a usar)
   (:predicates   <p-1> <p-2>  … <p-n>)                   ; definición de predicados
   (:functions   <function-1> <function-2> … <function-n>)    ; definición de funciones

   (:durative-action   <name1>            ; acción con duración, también denominados operadores…
      :parameters (?par1 – <subtype1> ?par2 – <subtype2> ...)     ; parámetros de la acción
      :duration   <value>
      :condition (<condition1>… <conditionn>)  ; tres tipos de condiciones: "at start", "over all", "at end",
                                                 que representan las condiciones al inicio, durante toda la
                                                 ejecución y al final de la acción, respectivamente
      :effect (<effect1>… <effectn>)     ; dos tipos de efectos: "at start", "at end" que representan los
                                           efectos que suceden al inicio y al final de la acción, respectivamente
       )

   (:durative-action   <name2>     ) ; otras acciones…
)
```

Las partes principales para la especificación de un dominio son:

- ***Requerimientos*** de este dominio, es decir qué requisitos debe soportar el planificador para ser capaz de trabajar con este dominio. Los más habituales son:

:durative-actions	se requieren acciones durativas
:typing	se requieren predicados con tipo
:fluents	se requieren funciones numéricas

- ***Tipos y Subtipos*** de los objetos que se van a usar, y si se definirán constantes.

A continuación se definen los predicados y funciones que permitirán definir el estado actual del mundo en el problema de planificación.

- ***Predicados***, que permitirán representar información proposicional (booleana, solo puede tomar el valor cierto/falso) del estado actual en el problema de planificación. *Los predicados se especifican mediante sus parámetros y tipos.*
  En un cierto estado del problema se podrá indicar tanto el cumplimiento o incumplimiento de los predicados definidos. Ejemplo:

  "(at persona1 Valencia)" "(not (at persona1 Valencia))".

- ***Funciones,*** que permitirán representar *información numérica* del estado actual, pudiendo tomar cualquier valor numérico. *Las funciones se especifican mediante sus parámetros y tipos.* En PDDL no es necesario indicar el dominio de valores de una función, pudiendo contener tanto valores enteros como reales. Por ejemplo:

  "*(fuel avion1)"* es el equivalente a una variable numérica que podría tomar un valor como 0, 50.3, 200.60, etc.

La segunda parte del archivo define la lista de ***acciones*** *u operadores:*

- Cada **acción** tiene un *nombre, una lista de parámetros, una duración, unas precondiciones y unos efectos.*

  La acción, por tanto, representa una plantilla personalizable para modelar el conocimiento y funciona de forma similar a una regla de un SBC: si se cumplen todas las precondiciones (parte izquierda de la regla), se generan los efectos (parte derecha de la regla).

- Las ***precondiciones*** pueden tener que cumplirse al principio de la acción (*at start <pred>*), durante toda la ejecución de la acción (*over all <pred>*) o solo al final de la acción (*at end <pred>*). Para indicar más de una precondición, se utiliza la conectiva 'and'.

  Ejemplo: :condition (and (at start (at ?p ?c)) (over all (at ?a ?c)))

- Los ***efectos*** representan las consecuencias 'add' y 'delete' en la ejecución de una acción. Estas consecuencias se puede generar nada más comenzar la acción (*at start <pred>*) o al finalizar la misma (*at end <pred>*). Es decir, una acción representa una transición con una duración determinada. Para representar la consecuencia 'delete' se utiliza la conectiva 'not' y para indicar más de un efecto se utiliza la conectiva 'and'.

  Ejemplo: :effect (and (at start (not (at ?a ?c1))) (at end (at ?a ?c2)))

En función de los parámetros de una acción se instanciarán más o menos acciones. Por ejemplo, si una durative-action tiene 1 único parámetro y hay 3 valores que hacen matching con ese parámetro se instanciarán 3 acciones que podrán ser planificadas.

Veamos a continuación un fragmento de un ejemplo comentado del dominio ZenoTravel, que plantea un escenario de transporte aéreo de pasajeros:

Dominio Zenotravel

; Dominio Zenotravel

***; Breve descripción del dominio: se dispone de personas, aviones y ciudades. Las personas pueden embarcar/desembarcar en/de los aviones, que vuelan entre distintas ciudades. Existen dos formas de volar, una rápida y una lenta con distintos consumos de combustible*.**

(define (domain zeno-travel) ***; nombre del dominio***

(:requirements :durative-actions :typing :fluents) ***; se requieren acciones durativas, predicados con tipo y funciones numéricas***

(:types aircraft person city - object) ***; existen tres tipos de objetos: avión, persona y ciudad***

```
(:predicates   (at  ?x - (either person aircraft)   ?c - city)
               (in   ?p  – person   ?a  - aircraft))          ; hay predicados que nos permiten conocer en qué
                                                              ciudad  está una persona o avión ("at"; ej. (at juan valencia)),  y otros
                                                              cuando una persona está dentro de un  avión ("in"; ej. (in juan avion1))

(:functions   (fuel ?a - aircraft)    ; función numérica para representar el nivel de combustible de un avión
              (distance ?c1 - city ?c2 - city)    ; función numérica para conocer la distancia entre 2 ciudades
              (slow-speed ?a - aircraft)    ; función numérica para la velocidad lenta de un avión
              (fast-speed ?a - aircraft)    ; función numérica para la velocidad rápida de un avión
              (slow-burn ?a - aircraft)    ; función numérica con el ritmo de consumo de combustible lento
              (fast-burn ?a - aircraft)    ; función numérica con el ritmo de consumo de combustible rápido
              …
              (total-fuel-used)    ; función numérica para la cantidad total de combustible consumido
              (boarding-time)    ; función numérica para el tiempo que se tarda en embarcar.
                                 NOTA: si se quisiera modelar un tiempo distinto por avión y por persona
                                       sería necesario definir una función de la forma
                                       (boarding-time ?p - person ?a - aircraft)
              (debarking-time)))  ; función numérica para el tiempo que se tarda en desembarcar

; A continuación vienen las acciones (también denominadas operadores)

(:durative-action board    ; acción de embarcar
  :parameters (?p - person ?a - aircraft ?c - city)  ; hay 3 parámetros: persona, avión y ciudad
  :duration (= ?duration (boarding-time))          ; la duración viene dada por la función "boarding-time"
  :condition (and (at start (at ?p ?c))
                  (over all (at ?a ?c)))    ; hacen falta dos condiciones: al principio de la acción ("at start")
                                              la persona tiene que estar en la ciudad; durante toda la ejecución
                                              de la acción ("over all") el avión debe permanecer en la ciudad
  :effect (and (at start (not (at ?p ?c)))
               (at end (in ?p ?a))))          ; se generan dos efectos: al principio de la acción ("at start")
                                              la persona deja de estar en la ciudad; al final de la acción ("at end")
                                              la persona pasa a estar dentro del avión. En otras palabras, se
                                              genera una transición en la que la persona pasa de estar en la
                                              ciudad a estar dentro del avión

(:durative-action fly      ; acción de volar a velocidad lenta
   :parameters (?a - aircraft ?c1 ?c2 - city)        ; 3 parámetros: avión, ciudad origen y ciudad destino
   :duration (= ?duration (/ (distance ?c1 ?c2) (slow-speed ?a)))  ; la duración se calcula como una expresión
                                                                   numérica, aunque también podía ser fija (ej: (:duration (= ?duration 5)))
   :condition (and (at start (at ?a ?c1))    ; al principio el avión debe estar en la ciudad origen
                   (at start (>= (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)))))  ; debe haber suficiente
                                                                                     combustible para cubrir toda la distancia – ritmo de consumo de combustible lento
   :effect (and (at start (not (at ?a ?c1)))          ; el avión ya no está en la ciudad origen
                (at end (at ?a ?c2))                  ; el avión está en la ciudad destino
                (at end (increase (total-fuel-used) (* (distance ?c1 ?c2) (slow-burn ?a))))  ; se incrementa el total
                                                                                              de combustible
                (at end (decrease (fuel ?a) (* (distance ?c1 ?c2) (slow-burn ?a)))))  ; se decrementa el
                                                                                      combustible del avión actual
```

```
(:durative-action zoom          ; acción de volar a velocidad rápida – análoga a "fly" pero con un mayor
                                ritmo de consume combustible
   :parameters (?a - aircraft ?c1 ?c2 - city)
   :duration (= ?duration (/ (distance ?c1 ?c2) (fast-speed ?a)))
   :condition (and (at start (at ?a ?c1))
                   (at start (>= (fuel ?a)  (* (distance ?c1 ?c2) (fast-burn ?a)))))
   :effect (and (at start (not (at ?a ?c1)))
                (at end (at ?a ?c2))
                (at end (increase (total-fuel-used) (* (distance ?c1 ?c2) (fast-burn ?a))))
                (at end (decrease (fuel ?a) (* (distance ?c1 ?c2) (fast-burn ?a))))))

……          ; también existen acciones para desembarcar (debark) y repostar (refuel) que se pueden
            observar en el dominio original zenotravel.pddl
```

Como puede observarse, la definición del dominio es **única** para modelar el conocimiento sobre un escenario determinado. En este caso se ha definido el conocimiento de la acción embarcar "board" en base a una serie de parámetros, precondiciones y efectos. Obviamente, no se ha indicado de cuántas personas, aviones y ciudades se dispone pues eso es independiente del comportamiento de esta acción y del dominio en general. Finalmente, si estamos ante un problema con 2 personas, 3 aviones y 5 ciudades se producirán 2*3*5=30 instanciaciones de "board", una para cada combinación de valores.

2.1.- Especificación del Problema

En segundo lugar se define el archivo del problema:

```
(define (problem <name>)
   (:domain <name >)              ; nombre del dominio al que pertenece este problema
   (:objects <obj1> - <type1> … <objn> - <typen>)          ; objetos y sus tipos
   (:init                                                 ; estado inicial
         (<predicate1>)  …  (<predicatei>)                ; parte proposicional. Predicados.
         (= <fuction-1> <value1>)… (= <function-n> <valuen>))  ; parte numérica. Funciones.
   (:goal                                                 ; objetivos
      (and  ((<predicate1>)  …  (<predicatei>)            ; objetivos proposicionales
            (<operator1>  <fuction-1>  <value1>) …        ; objetivos numéricos
            (<operatorn>  <fuction-n>  <valuen>)))
   (:metric minimize|maximize <expression>)  ; opcionalmente una métrica a min/max-imizar que
                                             representa la calidad del plan
)
```

En esencia, un problema en PDDL está formado por cuatro apartados.

- **Objetos:** En primer lugar, se define el número de objetos de cada tipo de que se dispone en el problema actual. Por ejemplo, se puede indicar que se dispone de un "avion1, avion2, avion3" de tipo "aircraft " (el tipo aircraft se definió en el dominio, de ahí que un problema deba trabajar con un dominio específico). A partir de estos objetos se podrán instanciar los predicados, funciones y acciones definidas previamente en el archivo de dominio.

- **Estado Inicial:** En segundo lugar se define la información inicial (init) para este problema concreto. Aquí se debe inicializar toda la información necesaria para resolver este problema de planificación. La definición de los predicados y de las funciones se realizó en el dominio.
    - En el caso de la información proposicional, basta con indicar los predicados que están en el estado inicial, es decir aquellos con valor cierto; los predicados no indicados se inicializarán con valor falso.
    - En el caso de la información numérica, habrá que inicializar **absolutamente** todas las funciones con un valor determinado. Por ejemplo: "(at persona1 Valencia)" indicará que la persona1 está en Valencia. Si no se indica este predicado entonces se considerará que la persona1 no está en Valencia. Por otro lado "(= (fuel avion1) 100.3)" asignará el valor inicial 100.3 a la función (fuel avion1).

- **Objetivos.** En tercer lugar se define el conjunto de objetivos (goal) a conseguir en este problema. En este caso podemos optar por objetivos proposicionales, numéricos o una mezcla de ellos. Por ejemplo, una definición de objetivo:

```
(:goal (and
            (at persona1 Madrid)
            (> (fuel avion1) 200)))
```

  significa que la persona1 debe acabar en Madrid y el nivel de combustible del avion1 ser superior a 200.

- **Métrica de evaluación:** Finalmente, en cuarto lugar se define una métrica para representar la calidad del plan. La métrica es una expresión a minimizar/maximizar y el plan será mejor cuanto mejor sea el valor de la expresión de la métrica (obviamente, distintos planificadores podrán devolver planes de distinta métrica/calidad).

  Por ejemplo, ante una métrica del tipo "minimize (total-time)" un plan será mejor cuanto menor sea el valor de la función por defecto "total-time", que representa la duración total.

  Es importante notar que la métrica es una expresión **deseable a optimizar**, pero prácticamente ningún planificador actual garantiza la solución óptima, ya que ello conlleva un proceso de búsqueda muy costoso.

De nuevo explicaremos un fragmento de un problema sencillo del escenario Zenotravel:

```
(define (problem ZTRAVEL-1-2)    nombre del problema

(:domain zeno-travel)    ; nombre del dominio – debe corresponderse con el definido en el dominio

(:objects    ; en este dominio hay 1 avión, 2 personas y 3 ciudades según los tipos definidos en el dominio
        plane1 - aircraft
        person1 - person
        person2 - person
        city0 - city
        city1 - city
        city2 – city)
```

(:init ***; estado inicial del problema actual.***

Importante: es necesario inicializar todos los predicados y detallar el valor de todas funciones que se van a usar en el problema ya que, de otra forma, el problema no estaría correctamente definido.

Si un predicado proposicional aparece en "init" se inicializa con el valor cierto, y falso en caso contrario. Si se trata de una función numérica hay que inicializarla con un valor numérico concreto

; NOTA: se deben inicializar los valores de todas las funciones que se vayan a utilizar. De lo contrario el comportamiento puede ser inesperado.

```
        (at plane1 city0)              ; el avión plane1 en city0 – información proposicional
        (= (slow-speed plane1) 198)    ; la velocidad lenta de plane1 – información numérica
        (= (fast-speed plane1) 449)    ; la velocidad rápida de plane1
        (= (fuel plane1) 3956)         ; combustible inicial de plane1
        (= (slow-burn plane1) 4)       ; ritmo lento de consumo de combustible de plane1
        (= (fast-burn plane1) 15)      ; ritmo rápido de consume de combustible de plane1
        (at person1 city0)             ; la persona person1 está inicialmente en city0
        (= (distance city0 city0) 0)   ; distancias entre pares de ciudades…
        (= (distance city0 city1) 678)
        (= (distance city0 city2) 775)
        ...
        (= (total-fuel-used) 0)        ; valor inicial del combustible acumulado
        (= (boarding-time) 0.3)        ; duración necesaria para embarcar
        (= (debarking-time) 0.6)       ; duración necesaria para desembarcar
)

(:goal (and                            ; objetivos a conseguir
        (at plane1 city1)              ; plane1 tiene que acabar en city1 – objetivo proposicional
        (at person1 city0)             ; person1 tiene que acabar en city0
        (at person2 city2)             ; person2 tiene que acabar en city2
        ; NOTA: un ejemplo de objetivo numérico podría ser: (< (total-fuel-used) 300)
))

(:metric minimize (+ (* 4 (total-time)) (* 0.005 (total-fuel-used)))) ; calidad (métrica) a optimizar
```

; La función a minimizar contempla tanto el tiempo total (duración del plan "total-time") como el consumo total del combustible.

Obviamente, se pueden definir otras funciones, como por ejemplo solo el (total-time) o el consumo de combustible de un avión en particular, por ej. (fuel plane1).

Los planificadores tratarán de optimizar esta función, pero difícilmente podrán garantizar la solución óptima por ser muy costoso

NOTA: (total-time) es una función definida por defecto, que NO hay que definir en ":functions" ni inicializar en ":init", ya que representa la duración total del plan y depende de cómo se posicionen las acciones en él. Obviamente, si todas las acciones del plan están en una secuencia la duración total será mayor que si hay acciones en paralelo.

3. Objetivos y contenido de la práctica

Los objetivos de esta práctica son básicamente:

- Conocer y analizar el formato PDDL. Nos centraremos en la versión temporal con acciones durativas introducida en PDDL 2.1, tal y como se ha descrito en el apartado anterior, pues es la que más se utiliza en la actualidad.
- Trabajar con una colección de dominios y problemas de planificación en PDDL, utilizando algunos de los planificadores más comunes para su resolución.
- Realizar modificaciones en los archivos PDDL y diseñar un nuevo dominio+problema para un caso concreto.

3.1.- Dominios de Planificacion

De la multitud de dominios de planificación utilizados como benchmarks, nos centraremos en tres de ellos:

Rovers. Dominio inspirado por las misiones del Mars Exploration Rover (MER) de la NASA. El objetivo es utilizar unos rovers para visitar distintos puntos de interés sobre un planeta y realizar una serie de muestreos para, posteriormente, comunicar los datos a su lanzadera. En el dominio se incluyen restricciones de navegación hacia los puntos de interés, de visibilidad de la lanzadera, de consumo de energía y de capacidad para realizar distintos tipos de muestras, ya que no todos los rovers están equipados para realizar todas las misiones.

Storage. Dominio utilizado para trasladar cajas desde unos contenedores a determinados depósitos/almacenes utilizando grúas. En cada depósito, cada grúa puede moverse siguiendo un determinado mapa espacial que conecta las distintas áreas del depósito. En el dominio se incluyen restricciones espaciales en los depósitos y zonas de carga, distinto número de depósitos, grúas disponibles, contenedores y cajas.

Pipes. Dominio utilizado para controlar el flujo de los derivados del petróleo a través de una red de tuberías. En el dominio se incluyen restricciones sobre las tuberías, sus respectivos segmentos y ocupación, compatibilidad e interferencia entre productos y capacidades de los tanques.

3.2- Planificadores a utilizar

Existen muchos planificadores disponibles públicamente, principalmente para su ejecución desde Linux. En esta práctica utilizaremos dos planificadores de uso sencillo y que se comportan de forma eficiente en los dominios seleccionados:

Planificador lpg-td (http://zeus.ing.unibs.it/lpg/).

Planificador heurístico, basado en búsqueda local, que utiliza grafos de planificación relajados como se ha explicado en teoría para calcular sus estimaciones. Se trata de un ***planificador no determinista***, por lo que **no siempre** devuelve el mismo plan (aunque se le puede pasar un valor de semilla como parámetro para la repetición exacta de pruebas). Por tanto, es aconsejable invocarlo un número determinado de veces y quedarse con los valores de la mediana de las soluciones.

Un ejemplo de su ejecución es:

```
./lpg-td-1.0 –o dominio.pddl –f problema.pddl –n 3
```

En este caso se ejecutará el planificador con el dominio "dominio.pddl" y el problema "problema.pddl" y se le pide, si es posible, que obtenga hasta 3 soluciones, cada una mejor que la anterior (-n 3).
Excepcionalmente se puede sustituir la cadena "-n *valor*" por **"*-speed*"**, indicando que se obtenga un plan lo más rápido posible, o **"*-quality*",** indicando que se obtenga un plan tratando de que sea de buena calidad, es decir tratando de optimizar la métrica definida en "problema.pddl" (aunque no existe garantía de optimalidad del plan).

Un ejemplo del plan solución es:

```
0.0003: (POP-UNITARYPIPE S13 B1 A1 A3 B5 LCO OCA1 TA1-1-OCA1 TA3-1-LCO) [D:2.0000; C:0.1000]
2.0005: (PUSH-UNITARYPIPE S12 B5 A1 A2 B4 OCA1 LCO TA1-1-OCA1 TA2-1-LCO) [D:2.0000; C:0.1000]
4.0008: (PUSH-UNITARYPIPE S12 B0 A1 A2 B5 OC1B OCA1 TA1-1-OC1B TA2-1-OCA1) [D:2.0000; C:0.1000]
2.0010: (PUSH-UNITARYPIPE S13 B2 A1 A3 B1 GASOLEO LCO TA1-1-GASOLEO TA3-1-LCO) [D:2.0000; C:0.1000]
4.0012: (PUSH-UNITARYPIPE S13 B3 A1 A3 B2 RAT-A GASOLEO TA1-1-RAT-A TA3-1-GASOLEO) [D:2.0000; C:0.1000]
```

En cada línea aparece el instante de ejecución de la acción (antes de los ":"), la acción en sí con sus parámetros, y finalmente entre corchetes su duración y coste. Es importante darse cuenta de que las acciones mostradas no tienen que estar necesariamente ordenadas en el tiempo.

NOTA IMPORTANTE sobre el uso de lpg. El parámetro "–n <valor>" indica al planificador que trate de encontrar un determinado número "n" de soluciones. En algunos problemas hay que elegir dicho valor con sumo cuidado pues valores demasiados altos obligan a que el planificador dedique mucho tiempo en tratar de encontrar ese número de soluciones. Incluso, en algunos casos donde el problema no disponga de ese número de soluciones puede ocurrir que el planificador no termine nunca, ya que jamás será capaz de encontrar ese número de soluciones alternativas. Por lo tanto, para cada problema es aconsejable empezar con un valor pequeño (1 o 2) y, sucesivamente, ir incrementándolo.

Planificador mips-xxl (http://sjabbar.com/mips-xxl-planner).

Planificador heurístico que utiliza grafos de planificación con los efectos *delete* relajados. Además, utiliza técnicas de abstracción para obtener una representación de estados muy compacta que hace uso del espacio de disco en aquellos casos en los que el espacio de búsqueda supere la memoria disponible. Se trata de un ***planificador determinista***. Un ejemplo de su ejecución es:

```
./mips-xxl –o dominio.pddl –f problema.pddl –O
```

Al igual que en el planificador anterior, el dominio y el problema serán, respectivamente, los ficheros "dominio.pddl" y "problema.pddl". La opción "–O" indica que se tratará de obtener la solución óptima con respecto a la métrica definida en "problema.pddl". Si no existe dicha opción, la métrica a optimizar será simplemente la longitud del plan.

Un ejemplo del plan solución es:

```
0.00: (POP-UNITARYPIPE S13 B1 A1 A3 B5 LCO OCA1 TA1-1-OCA1 TA3-1-LCO ) [2.00]
2.01: (PUSH-UNITARYPIPE S12 B5 A1 A2 B4 OCA1 LCO TA1-1-OCA1 TA2-1-LCO ) [2.00]
4.02: (PUSH-UNITARYPIPE S13 B2 A1 A3 B1 GASOLEO LCO TA1-1-GASOLEO TA3-1-LCO ) [2.00]
6.03: (PUSH-UNITARYPIPE S13 B3 A1 A3 B2 RAT-A GASOLEO TA1-1-RAT-A TA3-1-GASOLEO ) [2.00]
8.04: (PUSH-UNITARYPIPE S12 B0 A1 A2 B5 OC1B OCA1 TA1-1-OC1B TA2-1-OCA1 ) [2.00]
```

El formato del plan es similar al de lpg-td. En este caso se muestra el instante de ejecución de la acción, la acción y sus parámetros y, finalmente, la duración entre corchetes.

Muy importante: mips-xxl devuelve por consola un plan que puede no ser tan paralelo como debería. El plan totalmente paralelizado (ejecutando tantas acciones simultáneas como sea posible) se genera siempre en un archivo denominado "ffPSolution.soln" –no confundir con "ffSolution.soln" que es el plan totalmente secuencial.

mips-xxl necesita que todas las funciones estén entre "()". Es decir, "total-time" se considerará un error, siendo lo correcto (total-time).

NOTA. La ejecución de ambos planificadores sin parámetros muestra un mensaje con todos los parámetros admitidos y su significado.

Evaluación de la Práctica

El objetivo de la práctica es utilizar los planificadores proporcionados para familiarizarse con su uso y ejecución en los dominios facilitados. Adicionalmente, se debe aprender a modelar un escenario de planificación utilizando el lenguaje PDDL.

La realización de esta práctica es individual y consistirá en dos partes:

Parte 1 (15%).

Utilizar los dos planificadores proporcionados para analizar y resolver los problemas de los tres dominios propuestos (rovers, storage y pipes). ***Basta con probar los cuatro primeros problemas (p01, p02, p03 y p04) de cada dominio.***

Se deberá entregar una **memoria** con una tabla comparativa de los resultados y un análisis de los mismos en términos de tiempo de ejecución y calidad de la solución.

Nota 1: recordad que en el caso de lpg-td, un valor elevado del parámetro "–n <valor>" puede hacer que el planificador nunca termine.

Nota 2: recordad que lpg-td es un planificador no determinista, y el plan obtenido de una ejecución a otra puede ser notablemente distinto. Lo que se suele hacer en este caso es invocarlo un determinado número de veces y quedarse con la solución que se encuentre en la mediana de las ejecuciones. Por ejemplo, se ejecuta 3 veces y nos quedamos con la solución que esté en la posición media (ni la mejor ni la peor).

Parte 2 (85%).

Modelar un dominio+problema en PDDL y evaluarlo. Para ello se utilizará uno de los enunciados propuestos. Dichos enunciados son orientativos y, por tanto, el desarrollo de esta parte está abierto a cualquier **aportación propia** del alumno, pudiendo extender e incorporar cualquier criterio/conocimiento que quiera añadir a fin de que el sistema incorpore más conocimiento. Estas aportaciones se valorarán positivamente en la nota del proyecto.

Se deberá entregar una **memoria** en la que se explique cómo se ha modelado el dominio+problema (con el código implementado), así como pruebas de su ejecución y conclusiones.

Existe **un plazo de entrega** para el entregable y memoria de las dos partes.

PARTE 1 (15%)

Analizad los tres dominios propuestos y ejecutad los dos planificadores para resolver distintos problemas (p01..p04). Comparad los resultados de los planificadores en términos de tiempo de ejecución y calidad de la solución obtenida, en términos de número de acciones y su duración (también conocido como *makespan*).

NOTA. Para conocer con mayor precisión el tiempo de ejecución de un programa se puede usar el comando de Linux "time".

```
time ./mips-xxl –o dominio.pddl –f problema.pddl –O
```

que devuelve algo de este estilo, con el tiempo real, de usuario (el que realmente nos interesa) y de sistema respectivamente:

```
real 0m0.084s
user 0m0.000s        --- este valor es el que realmente interesa medir
sys  0m0.008s
```

PARTE 2 (85%)

Elegid UNA de las siguientes propuestas

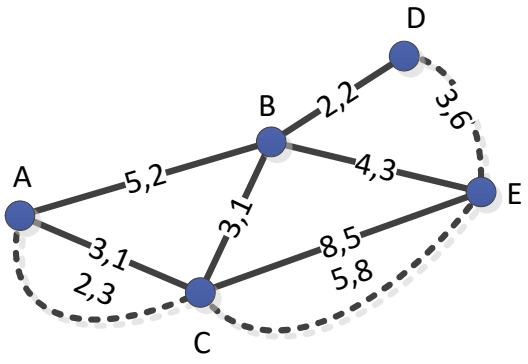
Para la evaluación de esta parte se debe elegir uno de los siguientes ejemplos o propuestas. Estos ejemplos pueden ser ampliados con otros casos/tipologías.

Ejemplo 1. Escenario de logística

Disponemos de una empresa de transporte de envío de paquetes entre ciudades conectadas por una red de carreteras. Cada camión debe ser conducido por un conductor. Los objetivos son que los paquetes, camiones y conductores acaben en determinadas ciudades. Básicamente, se necesitan las siguientes acciones:

- Cargar/descargar un paquete en un camión en una ciudad. La carga se puede hacer independientemente de que exista un conductor o no en el camión. Puesto que los paquetes son pequeños no existe limitación en el número de paquetes por camión.
- Subir/bajar un conductor en un camión situado en una ciudad. Como máximo solo un conductor puede estar en el camión en cada momento y, lógicamente, no puede estar en más de un camión a la vez.
- Conducir un camión por un conductor de una ciudad origen a una ciudad destino.
- Viajar en autobús para que un conductor pueda llegar de una ciudad origen a una destino. El autobús utiliza la misma red de carreteras que los camiones. No es necesario modelar explícitamente el autobús ni el hecho de subir/bajar de él.

Las acciones de cargar/descargar y subir/bajar tienen una duración y coste de 1 unidad. Las acciones de viajar en autobús tienen una duración de 10 unidades y un coste de 3 unidades. Las ciudades, red de carreteras y duración/coste de las acciones de conducir se muestran en la Figura 1. Adicionalmente, en la Figura 1 se muestra también el estado inicial y objetivo para los paquetes, camiones y conductores



	Estado inicial	Objetivo
p1	A	E
p2	D	C
c1	A	-
c2	A	A
d1	C	B
d2	D	-

Figura 1. Red de carreteras con su duración y coste asociado (por ejemplo, la duración de conducir entre A-B y B-A es de 5 unidades y su coste es 2). Las líneas discontinuas representan las autopistas enunciadas en el Ejemplo 2 propuesto (por ejemplo, la duración de conducir por autopista entre A-C y C-A es de 2 unidades y su coste es 3).

Se desea resolver el problema de planificación anterior utilizando la siguiente métrica para evaluar la calidad:
(:metric minimize (+ (* 0.1 (total-time)) (* 0.2 (coste-total))))

Ejemplo 2. Escenario de logística modificado

Se desea modelar el mismo problema que en el Ejemplo 1 pero con un par de modificaciones:

- No existe la posibilidad de que un conductor viaje en autobús, sino que para desplazarse entre ciudades siempre deberá utilizar los camiones de la empresa de transporte. En tal caso, los conductores d1, d2 **deberán estar** inicialmente en A.
- Se desea añadir una nueva acción "conducir por autopista un camión por un conductor de una ciudad origen a una ciudad destino". Esta nueva acción es equivalente a la acción original de conducir del Ejemplo 1, pero requiere que haya una autopista entre las dos ciudades. Esta acción será de menor duración que la acción original pero más costosa económicamente. Las autopistas entre ciudades se muestran con trazo discontinuo en la Figura 1, junto con su duración y coste asociado.

Se desea resolver el problema utilizando la métrica siguiente para evaluar la calidad del plan:
(:metric minimize (+ (* 0.2 (total-time)) (* 0.3 (coste-total))))

Ejemplo 3. Escenario del viajante de comercio (TSP)

Se desea modelar el típico problema del viajante de comercio en el que disponemos de un conjunto de ciudades conectadas unas con otras (asumiremos un grafo completo, es decir aquél en el que cada vértice está conectado con todos los demás vértices). Se disponen de dos acciones:

- Viajar de una ciudad origen a una destino.
- Realizar-visita en una ciudad dada.

El objetivo del problema es visitar todas las ciudades una única vez de forma que el viajante de comercio salga y regrese a la ciudad de origen. Se pide modelar distintos escenarios con 4, 8 y 10 ciudades, respectivamente, y analizar los resultados obtenidos.

NOTA: se deja a elección del alumno establecer las duraciones de las distintas acciones "viajar" y "visitar". En principio, asumimos que el grafo formado por las ciudades es no dirigido, lo que implica que la duración entre dos ciudades es simétrica.

Adicionalmente, incluid también las dos siguientes modificaciones (explicando detalladamente los cambios realizados en el archivo de dominio PDDL):

- No se necesita recorrer todas las ciudades ni es necesario que el viajante regrese a la ciudad de origen.
- Las distancias entre ciudades no son simétricas (grafo dirigido). Es decir, la distancia entre ciudadA→ciudadB no tiene que coincidir con la distancia ciudadB→ciudadA.

Se desea resolver el problema minimizando la duración del plan (total-time). La ciudad de origen y de destino se deja a libertad del alumno.

Ejemplo 4. Gestión de grúas de un puerto de contenedores

Deseamos modelar un problema para gestionar las grúas de un puerto que se dedica al movimiento (carga y descarga) de contenedores. Concretamente, en nuestro caso disponemos de dos grúas como las que se muestra en la Figura 2. Cada grúa puede realizar las siguientes acciones:

- Cargar un contenedor de una plataforma temporal de almacenamiento. Análogamente, se dispone de la acción para descargar un contenedor del brazo de la grúa a la plataforma de almacenamiento. Actualmente se dispone únicamente de dos espacios de almacenamiento compartidos por las dos grúas, pero se prevé ampliar dicho número a cuatro espacios de almacenamiento. Por tanto, el modelo PDDL debe facilitar dicha ampliación de una forma sencilla. Las acciones de carga/descarga, al hacer uso de la plataforma de almacenamiento, tienen un coste de 5 unidades cada una y una duración de 10 unidades.
- Apilar un contenedor ya sujetado por el brazo sobre un contenedor existente. Análogamente, se dispone de la acción para desapilar un contenedor que se encuentra sobre otro contenedor. Las acciones de apilar/desapilar tienen un coste de 2 unidades y una duración de 5 unidades.

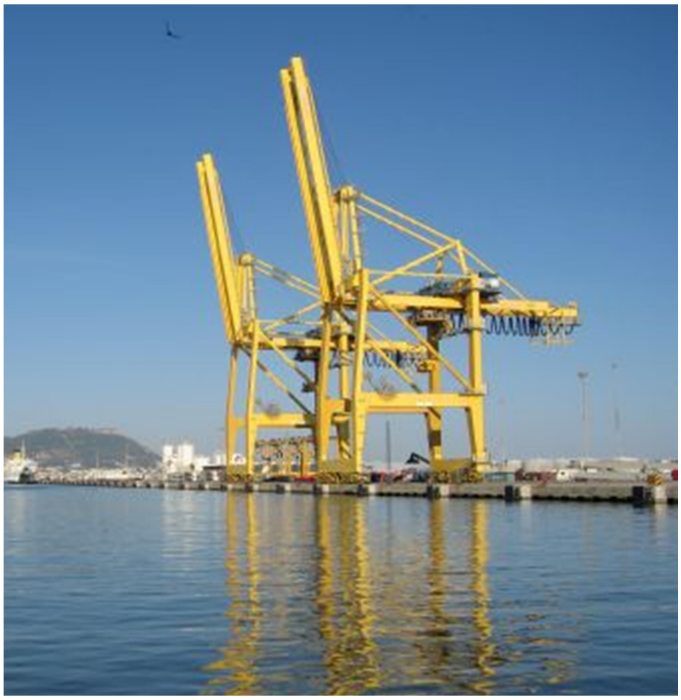


Figura 2. Las dos grúas del terminal de contenedores del problema a resolver.

NOTA. En nuestro problema consideraremos que no existe límite en la altura máxima de contenedores apilados, aunque se deja como ampliación que el alumno lo modele.

Se desea resolver el problema de dos formas distintas para así comparar los planes obtenidos: 1) minimizando la duración del plan (total-time), y 2) minimizando el coste total. Se deja a libertad del alumno modelar el número de contenedores de los que se dispone, así como la configuración del estado inicial/objetivo de los contenedores (es decir, las pilas de las que hay que partir y a las que hay que llegar). Se recomienda analizar distintas configuraciones.

Ejemplo 5. Juego infantil del mono

Se desea modelar un juego infantil en el que hay que alimentar a un mono dándole plátanos y agua. En el problema existen varias habitaciones que el mono puede recorrer; en una de ellas hay unos plátanos colgados del techo y en otra una fuente de agua. Para conseguir los plátanos el mono necesita una caja sobre la que subirse, tal y como se muestra en la Figura 3, y unas tijeras para cortar los plátanos. Para conseguir el agua el mono necesita un vaso y estar en la habitación donde está la fuente.

Inicialmente, los objetos (mono, plátanos en el techo, tijeras, caja, vaso de agua y fuente) están distribuidos por las distintas habitaciones y se tiene que generar el plan adecuado para que el mono consiga comerse los plátanos y beber el agua. Se pide modelar tanto dichos objetos como las acciones necesarias para este juego. Se deja a libertad del alumno establecer el número de habitaciones del problema (con un mínimo de cinco), en qué posiciones están inicialmente los objetos y las duraciones de las tareas. Se desea resolver el problema minimizando la duración del plan (total-time) y analizar distintas configuraciones de habitaciones y posición de los objetos.

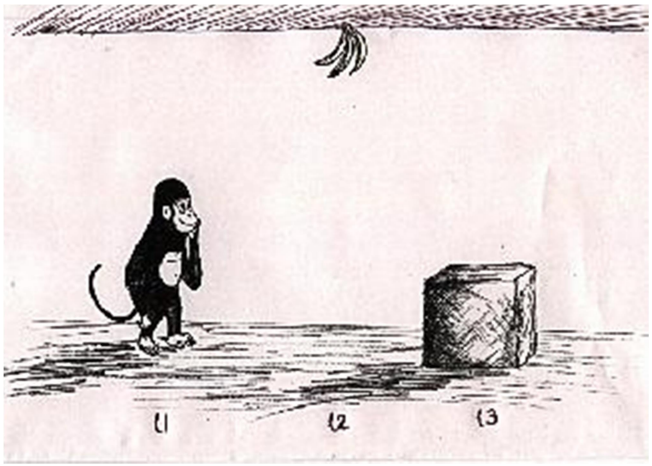


Figura 3. Juego infantil que consiste en alimentar a un mono con plátanos y agua.

Ejemplo 6. Planificación en ascensores inteligentes

Estamos trabajando en un edificio de 10 plantas que desea contar con ascensores inteligentes, de forma que los empleados puedan utilizarlos de la forma más eficiente posible. Para ello, de cada empleado se conoce en todo momento en qué planta del edificio está y, mediante un dispositivo de comunicación, a qué planta desea ir. Obviamente, se conoce en qué planta está cada ascensor. Disponemos actualmente de dos ascensores pero deseamos realizar un modelo PDDL que sea fácilmente extensible a un mayor número de ascensores. Las acciones a modelar son:

- Entrar/salir un empleado a/de un ascensor. Lógicamente, el empleado y el ascensor debe estar en la misma planta para que dicha acción sea posible. La duración de estas acciones es 1.
- Mover el ascensor de una planta a otra. No es necesario que el ascensor pare siempre en todas las plantas, pues dependerá de si es necesario que entren/salgan los empleados en ellas. La duración de mover depende de la distancia entre las plantas. Por simplicidad,

asumiremos que la duración es de 2 unidades por cada planta de separación. Por ejemplo, una acción directa de mover entre la planta 10 y 6 tardará (10-6)*2=8 unidades de tiempo. NOTA: PDDL no admite parámetros numéricos en las acciones, por lo que las plantas tendrán que modelarse como objetos de la forma planta01, planta02, etc.

Como prueba piloto se desea modelar 20 empleados que se quieren mover de una planta inicial a otra (se deja a libertad del alumno establecer estas plantas, aunque se recomienda hacer pruebas con distintos valores). El problema consistirá en obtener el plan que mueva a todos los empleados minimizando su duración total (expresión "(total-time)").

El alumno podrá proponer otro ejemplo de trabajo (se valorará especialmente el conocimiento adicional aportado al ejemplo propuesto)