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

Técnicas, Entornos y Aplicaciones de Inteligencia Artificial

Planificación

1. Introducción 2
2. Estructura de PDDL 3
2.1.- Especificación del Dominio 3
Dominio Zenotravel 4
2.2.- Especificación del Problema 6
Problema Zenotravel 7
3. Objetivos y contenido de la práctica 9
3.1.- Otros ejemplos de Dominios de Planificacion 9
3.2- Planificador a utilizar 9

Evaluación de la Práctica. Problema propuesto

1. Introducción

En esta práctica haremos uso de uno 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 solo 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 solo 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.

2.2.- 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 que no aparecen se inicializarán automáticamente con valor falso. En otras palabras, en el estado inicial **NO** se debe indicar la información negada; es decir, **no habrá** (not (predicadoX)).
    - 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:

Problema 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" ni utilizar en condiciones/efectos de las acciones, 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 un planificador para su resolución.
- Realizar modificaciones en los archivos PDDL y diseñar un nuevo dominio+problema para un caso concreto.

3.1.- Otros ejemplos de 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- Planificador a utilizar

Existen muchos planificadores disponibles públicamente, principalmente para su ejecución desde Linux. En esta práctica utilizaremos el planificador LPG, **recompilado para Windows** (siendo **necesaria la biblioteca cygwin1.dll**) de uso sencillo y que se comporta de forma eficiente en los dominios seleccionados.

Para usar lpg, basta copiar en una carpeta el ejecutable **lpg-td.exe**, junto con el fichero **cygwin1.dll**, sin ser necesaria ninguna instalación. Está disponible en Poliformat y también en https://lpg.unibs.it/lpg/.

LPG es un 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.exe –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. El coste indicado es el coste que supone la ejecución de dicha acción según la métrica de calidad del plan definida para el problema. 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.

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

Evaluación de la Práctica. Problema Propuesto

Antes de realizar el desarrollo de la práctica es imprescindible leer el documento anterior para comprender el modelado de problemas en PDDL y el funcionamiento de un planificador.

Brevemente, el objetivo de la práctica es el de modelar un escenario de planificación utilizando el lenguaje PDDL. Como objetivo adicional, utilizaremos el planificador proporcionado para familiarizarse con su uso y ejecución en algunos de los dominios facilitados.

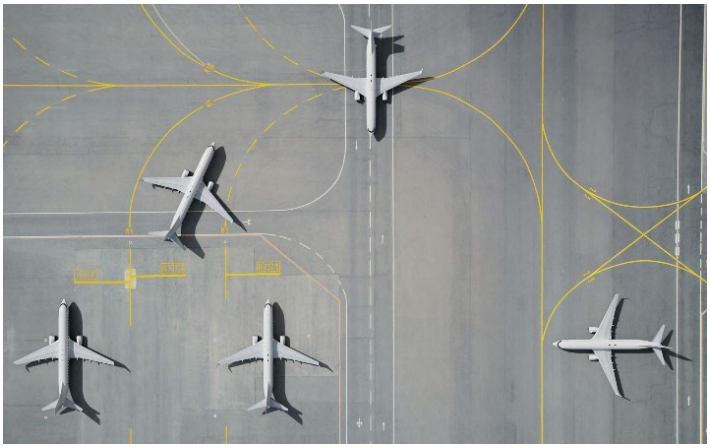
Tras el desarrollo de la práctica, su evaluación consistirá en rediseñar y evaluar pequeñas modificaciones del problema propuesto.

Nota 1: se facilitan los dominios (y problemas) de planificación de *rovers, storage y pipes* como muestra de ejemplo. Para hacerse una idea aproximada de los mismos y su solución, bastará con probar los cuatro primeros problemas (p01, p02, p03 y p04) de cada dominio.

Nota 2: recordad que en el caso de lpg-td, un valor elevado del parámetro "–n <valor>" puede hacer que el planificador nunca termine.

Nota 3: 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).

Ejercicio. Gestión inteligente de los movimientos de aviones en un aeropuerto



Utilizando técnicas de planificación inteligente, deseamos gestionar los movimientos de los aviones y los tractores, que remolcarán los aviones, a través de las distintas instalaciones de un aeropuerto: terminales y pistas (de rodaje y de aterrizaje). Cuando un avión aterriza, el flujo de movimiento es: pista de aterrizaje, pista/s de rodaje (recorriendo una o muchas) y aparcado en el finger (o pasarela de acceso) de una terminal. Cuando el avión desea despegar, el flujo será justo el inverso.

Los aviones pueden moverse de forma autónoma entre pistas, bien sean pistas de rodaje o pistas de aterrizaje. Sin embargo, debido a su limitada maniobrabilidad, no podrán llegar por sí solos a las terminales, teniendo que ser remolcados por los tractores. Es decir, un avión necesitará siempre a un tractor que lo remolque desde una pista de rodaje a la terminal y, en sentido inverso, desde la terminal a la pista de rodaje.

Los tractores son eléctricos (con el fin de reducir las emisiones contaminantes) y sin conductor. Por lo tanto, hay que planificar de forma autónoma todos sus movimientos, bien sea remolcando aviones o moviéndose cuando están vacíos.

El sistema a modelar presenta las siguientes características:

- Hay dos tipos de vehículos: aviones y tractores. Cada vehículo tiene una determinada velocidad.

- Hay dos tipos de pistas: de aterrizaje y de rodaje. También se dispone de terminales y fingers. Una terminal puede tener varios fingers, que conectarán a cada una de las puertas de la terminal. Necesitaremos modelar las acciones de desplegar el finger, para que el pasaje pueda abandonar el avión tras su llegada a la terminal, y replegar el finger, para que el avión esté listo para abandonar el aeropuerto.

- Los aviones pueden moverse libremente y sin ninguna ayuda extra entre pistas de aterrizaje y de rodaje (en cualquier sentido), pero no pueden alcanzar/abandonar las terminales por sí solos. Para que un avión pueda alcanzar/abandonar una terminal necesitará obligatoriamente ser remolcado por un tractor. Un tractor podrá moverse vacío o remolcando a un avión, pero nunca podrá llegar a una pista de aterrizaje. En otras palabras, las pistas de aterrizaje son exclusivas de los aviones, las terminales son exclusivas de los tractores (remolcando o no aviones) y las pistas de rodaje son para el movimiento de cualquier vehículo (tractor o avión). Existe un modelo de conexiones del aeropuerto, indicando los elementos que están conectados, así como la distancia entre ellos.

- Evidentemente, un tractor solo puede remolcar un avión en cada momento. Una terminal puede dar servicio a múltiples aviones gracias a sus múltiples fingers, pero cada finger solo puede servir a un avión en cada momento.

- Los tractores funcionan con batería, que debe cargarse en un cargador libre, situado en una terminal. Por tanto, se deberá modelar el nivel de carga restante de la batería y planificar cuándo/dónde se debe recargar para que el tractor no se quede sin batería. Actualmente, cada terminal dispone de un solo cargador. Un cargador solo puede cargar a un tractor en cada momento. Cada tractor tiene una duración de carga.

- De momento, el sistema está en modo de pruebas, por lo que el número de tractores y cargadores es limitado. Se prevé que estos números aumenten considerablemente en el futuro.

El modelo de conexiones del aeropuerto es el mostrado en las siguientes tablas. Lógicamente, un vehículo solo podrá moverse entre dos posiciones que estén conectadas (las celdas en blanco representan ausencia de conexión). Por simplicidad, la distancia entre dos posiciones se considera simétrica, aunque **habrá que modelarla explícitamente en los dos sentidos**:

	pista-rodaje1	pista-rodaje2	pista-rodaje3
pista-aterrizaje1	8	12	
pista-aterrizaje2	16		8

	pista-rodaje1	pista-rodaje2	pista-rodaje3
pista-rodaje1		8	
pista-rodaje2	8		4
pista-rodaje3		4	

	terminal1	terminal2	terminal3
pista-rodaje1	20	20	24
pista-rodaje2		12	12
pista-rodaje3			20

	terminal1	terminal2	terminal3
terminal1		8	
terminal2	8		12
terminal3		12	

Existen varias acciones a planificar:

- **Mover** un avión entre dos pistas (origen y destino), pudiendo ser indistintamente de aterrizaje o de rodaje. El avión se mueve por sí solo y no debe estar siendo remolcado por un tractor. La duración de la acción viene dada por la distancia a recorrer dividida por la velocidad del avión.

- **Mover** un tractor vacío, sin remolcar un avión, entre una terminal y una pista de rodaje (en cualquier sentido). La duración de la acción viene dada por la distancia a recorrer entre el origen y el destino dividida por la velocidad del tractor. Esta acción consume 2 unidades de batería del tractor, por lo que antes de ejecutarla hay que asegurarse de que la carga restante sea al menos de 2 unidades.

- **Remolcar** un avión por un tractor. Esta acción es análoga a la anterior, pero ahora remolcando un avión: el tractor remolcará al avión desde un origen a un destino, de forma que el avión pasará también del origen al destino. Un tractor solo puede remolcar un avión en cada momento. La duración de la acción viene dada por el doble de la distancia a recorrer dividida por la velocidad del tractor. Esta acción consume 4 unidades de batería del tractor, por lo que antes de ejecutarla hay que asegurarse de que la carga restante sea al menos de 4 unidades. Es decir, remolcar dura el doble que mover un tractor vacío y consume el doble de batería.

- **Desplegar** el finger de una terminal para que un avión pase a estar aparcado en ese finger. En un momento dado, el finger solo puede dar servicio a un avión. Hay que modelar que el avión esté aparcado en ese finger, porque no puede estar aparcado en varios fingers simultáneamente. Para desplegar el finger, el avión debe estar en esa terminal y no puede estar siendo remolcado. La duración de esta acción es de 2 unidades. Actualmente, disponemos de 4 fingers: finger1 en terminal1, finger2 en terminal2; y finger3 y finger4 en terminal3.

- **Replegar** el finger de una terminal para que un avión deje de estar aparcado en ese finger. Esta acción es la inversa de la anterior y también dura 2 unidades.

- **Recargar** la batería de un tractor en el cargador de la terminal. El cargador debe estar disponible para poder cargar la batería (solo se puede cargar un tractor en cada momento). Esta acción solo se llevará a cabo si la carga restante del tractor es menor o igual a 6 unidades. Tras la recarga, la carga restante del tractor pasará a ser de 10 unidades. También nos interesa conocer el número de recargas que se realizan en total, por lo que deberemos modelar dicho número. Cada vez que se recargue un tractor, se incrementará el número de recargas en una unidad. La duración de esta acción depende de la duración de la carga del tractor. Solo existen 3 cargadores, uno por terminal.

Desde el punto de vista de optimización, nos interesa minimizar la duración del plan y el número de recargas realizadas, utilizando la siguiente métrica para evaluar la calidad del plan:

(:metric minimize (+ (* 0.5 (total-time)) (* 10 (numero-recargas))))

Queremos obtener el plan para dar servicio a 6 aviones utilizando 2 tractores. En la siguiente tabla se muestra la información necesaria para cada vehículo:

Vehículo	velocidad	posición inicial	posición objetivo	carga restante tractor	duración carga tractor
avion1	2	pista-aterrizaje1	finger1	-	-
avion2	2	pista-aterrizaje2	finger3	-	-
avion3	4	finger1	pista-aterrizaje1	-	-
avion4	4	finger2	pista-aterrizaje2	-	-
avion5	2	finger3	pista-aterrizaje2	-	-
avion6	2	finger4	pista-aterrizaje1	-	-
tractor1	1	terminal2	terminal1	4	10
tractor2	1	pista-rodaje2	terminal2	2	14

Todos los tractores y cargadores están inicialmente disponibles.

Probad el planificador LPG en sus distintas opciones -n, -speed, -quality y estudiad/analizad los distintos planes solución (y métricas). LPG no muestra explícitamente el valor correcto de la métrica, por lo que si queremos conocerla debemos calcularla manualmente. El único valor correcto que proporciona LPG es el de la duración, pues LPG añade un coste ficticio a todas las acciones que no tienen coste asociado (típicamente 0.1). Por lo tanto, el número de las recargas y de la métrica se tiene que obtener a mano sobre el plan resultante.

NOTA. La mejor solución que hemos obtenido con LPG para este problema es un plan de duración 146, incluyendo 4 acciones de recarga. Hemos obtenido dos planes con esa misma métrica, uno de 29 acciones y otro de 31. Estos planes no tienen que corresponderse necesariamente con los valores de la solución óptima, sino que se corresponden con la mejor solución que hemos encontrado en nuestras pruebas.

A modo de ejemplo, para el avión1 se necesitará: 1) moverlo de la pista-aterrizaje1 a una pista de rodaje, 2) remolcarlo por un tractor desde esa pista hasta la terminal1 (es posible que se necesiten pistas de rodaje intermedias), y 3) desplegar el finger1 para que el pasaje pueda desembarcar en la terminal1.