



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



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

Técnicas, Entornos y Aplicaciones de Inteligencia Artificial

Planificación

Autor: Axel Guzmán Godia

Introducción

El objetivo de esta práctica es introducir el uso de planificadores y dominios definidos mediante lenguajes PDDL. Para ello, en la primera parte se pide al alumno que ejecute los planificadores proporcionados con problemas y dominios también proporcionados. En la segunda parte se pide implementar un dominio y un problema mediante el lenguaje PDDL.

Parte 1: Ejecución y comparación de planificadores

En este apartado presentaremos tablas comparativas de la ejecución de dos planificadores en diferentes problemas y dominios.

El primer planificador es el planificador lpg-td. Este es un planificador no determinista por lo que los resultados pueden variar. Para obtener una aproximación real, se ha ejecutado varias veces este planificador y se ha escogido el resultado mas común.

En cuanto al segundo planificador mips-xxl , se trata de un planificador que si es determinista. Cabe destacar que utiliza grafos de planificación con los efectos delete relajados.

A continuación se muestran tablas comparativas de la ejecución en diversos dominios.

Dominio pipes	Problema 1		Problema 2		Problema 3		Problema 4	
lpg-td	Número acciones plan	6	Número acciones plan	244	Número acciones plan	13	Número acciones plan	161
	Tiempo de ejecución del plan:	0.23s	Tiempo de ejecución del plan:	0.474s	Tiempo de ejecución del plan:	0.047s	Tiempo de ejecución del plan:	0.227s
	Tiempo acciones (makespan)	12	Tiempo acciones (makespan)	244	Tiempo acciones (makespan)	22	Tiempo acciones (makespan)	290
	Calidad de la solución	12	Calidad de la solución	244	Calidad de la solución	22	Calidad de la solución	290
MIPS-XXL Planner	Número acciones plan	5	Número acciones plan	14	Número acciones plan	9	Número acciones plan	13
	Tiempo de ejecución del plan:	0.006s	Tiempo de ejecución del plan:	0.009s	Tiempo de ejecución del plan:	0.045s	Tiempo de ejecución del plan:	0.106s
	Tiempo acciones (makespan)	6.02	Tiempo acciones (makespan)	28.14	Tiempo acciones (makespan)	14.06	Tiempo acciones (makespan)	22.10
	Calidad de la solución	5	Calidad de la solución	14	Calidad de la solución	9	Calidad de la solución	13

Dominio rovers	Problema 1		Problema 2		Problema 3		Problema 4	
lpg-td	Número acciones plan	11	Número acciones plan	10	Número acciones plan	17	Número acciones plan	11
	Tiempo de ejecución del plan:	0.047s	Tiempo de ejecución del plan:	0.022s	Tiempo de ejecución del plan:	0.21s	Tiempo de ejecución del plan:	0.018
	Tiempo acciones (makespan)	80	Tiempo acciones (makespan)	76	Tiempo acciones (makespan)	87	Tiempo acciones (makespan)	80
	Calidad de la solución	80	Calidad de la solución	76	Calidad de la solución	87	Calidad de la solución	80
MIPS-XXL Planner	Número acciones plan	10	Número acciones plan	8	Número acciones plan	13	Número acciones plan	8
	Tiempo de ejecución del plan:	0.005s	Tiempo de ejecución del plan:	0.007s	Tiempo de ejecución del plan:	0.025s	Tiempo de ejecución del plan:	0.004s
	Tiempo acciones (makespan)	57.05	Tiempo acciones (makespan)	47.04	Tiempo acciones (makespan)	67.05	Tiempo acciones (makespan)	38.03
	Calidad de la solución	10	Calidad de la solución	8	Calidad de la solución	13	Calidad de la solución	8

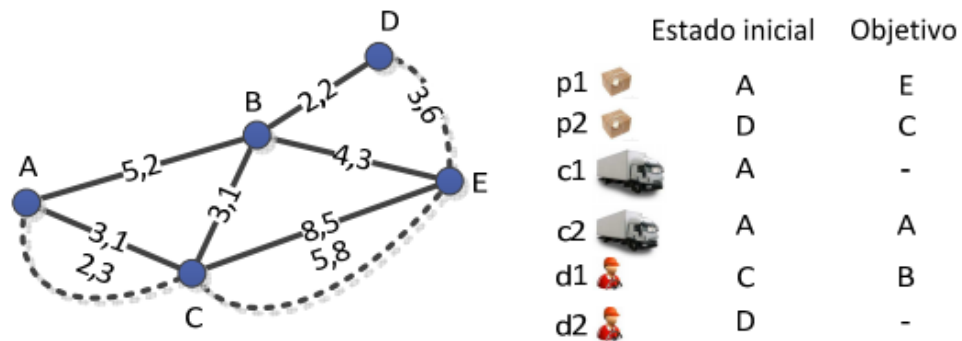
Dominio storage	Problema 1		Problema 2		Problema 3		Problema 4	
lpg-td	Número acciones plan	3	Número acciones plan	3	Número acciones plan	3	Número acciones plan	11
	Tiempo de ejecución del plan:	0.11s	Tiempo de ejecución del plan:	0.015s	Tiempo de ejecución del plan:	0.013s	Tiempo de ejecución del plan:	0.019s
	Tiempo acciones (makespan)	3	Tiempo acciones (makespan)	3	Tiempo acciones (makespan)	3	Tiempo acciones (makespan)	15
	Calidad de la solución	3	Calidad de la solución	3	Calidad de la solución	3	Calidad de la solución	15
MIPS-XXL Planner	Número acciones plan	3	Número acciones plan	3	Número acciones plan	3	Número acciones plan	8
	Tiempo de ejecución del plan:	0.002s	Tiempo de ejecución del plan:	0.003s	Tiempo de ejecución del plan:	0.001s	Tiempo de ejecución del plan:	0.003s
	Tiempo acciones (makespan)	3	Tiempo acciones (makespan)	3	Tiempo acciones (makespan)	3	Tiempo acciones (makespan)	10
	Calidad de la solución	3	Calidad de la solución	3	Calidad de la solución	3	Calidad de la solución	8

Análisis de resultados

Tras comparar los datos obtenidos se observa que el planificador Mips-xxl siempre obtiene un resultado mejor. En el peor de los casos obtiene el mismo resultado. Además también es computacionalmente más rápido.

Parte 2 : Modelado de un problema

En este apartado se pide modelar un problema en lenguaje PDDL. Para completar este apartado he decidido elegir uno de los escenarios propuestos. He elegido el primer escenario. Este escenario consiste en un problema de transporte de paquetes. Tenemos una serie de conductores , paquetes y camiones. El objetivo es que cada uno acabe en un destino . A continuación se muestra una figura explicativa sobre el problema.



A continuación vamos a explicar cómo hemos definido las dos partes necesarias : el dominio y el problema

Definición del dominio

La definición del dominio es la parte más importante. Una vez definido el dominio definir problemas diferentes es sencillo. Vamos a empezar por la cabecera: (véase código completo en el anexo)

```
; Dominio : Camiones  
  
; Autor Axel Guzman Godia  
  
(define (domain Camiones)  
  
(:requirements :typing :durative-actions :fluents)  
  
(:types camion conductor ciudad paquete) ; Especificamos nuestros objetos
```

Las dos primeras líneas son comentarios y no tienen ninguna implicación. Mediante define indicamos el nombre del dominio.

A continuación, mediante la cláusula requeriments indicamos los requisitos que debe soportar el planificador para trabajar con el problema. En esta caso en particular tenemos tres requisitos. :typing indica que se requieren predicados con tipo. :durative-actions indica que se necesitan acciones con duración. Por último con :fluents indicamos que vamos a hacer uso de funciones numéricas.

A continuación vamos a indicar los tipos que utilizaremos en nuestro problema. Tenemos cuatro objetos que modelar, las ciudades, los conductores , los paquetes y los camiones.

Para continuar modelando el problema necesitamos definir los predicados.

```
(:predicates (at ?x - (either conductor camion paquete) ?c - ciudad)
              (in ?x - (either paquete conductor) ?cam - camion)
              (vacio ?camion - camion)
              (conectado ?c1 - ciudad ?c2 - ciudad ))
```

Lo que estamos indicando aquí son predicados que nos servirán para comprobar condiciones. En cierto modo los predicados funcionan cómo los hechos en CLIPS. Por ejemplo el primer predicado “at” se utilizará para indicar que un conductor, un camión , o un paquete se encuentran en una ciudad. Siguiendo está filosofía el predicado in permitirá saber si un conductor o un paquete están en un camión. El predicado vacio servirá para saber si un camión tiene un conductor. De este modo podremos controlar que sólo haya un conductor por camión. Por último el predicado conectado servirá para modelar las conexiones entre las ciudades.

A continuación vamos a definir las funciones que utilizaremos. Para resolver este problema será suficiente con definir tres funciones. Utilizaremos estas funciones para obtener el coste total de un plan , la distancia entre dos ciudades y para conocer si se puede viajar entre una ciudad y otra. La definición es la siguiente:

```
(:functions (distancia ?c1 - ciudad ?c2 - ciudad) ;función numérica para conocer la distancia entre 2 ciudades
            (costeViaje ?c1 - ciudad ?c2 - ciudad) ;función numérica para conocer el coste de viajar entre 2 ciudades
            (coste) ;función para calcular el coste total
)
```


Mediante estas funciones podremos conocer la distancia entre dos ciudades y el coste de dicho viaje. La función coste servirá como acumulador para ir guardando el coste total.

Por último sólo queda definir las acciones del dominio. En este apartado haremos uso tanto de los predicados como de las funciones definidos anteriormente. Para modelar esta parte es necesario preguntarse que acciones se pueden llevar a cabo en el domino.

Para nuestro problema tenemos cuatro acciones que modelar:

- 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.

Veamos a continuación la implementación de cada función:

Cargar camión

```
(:durative-action cargarCamion ;Función para modelar la acción de cargar en un camion
:parameters (?paquete - paquete ?ciudad - ciudad ?camion - camion)
:duration (= ?duration 1)
:condition (and (at start ( at ?camion ?ciudad))(at start ( at ?paquete ?ciudad))
                (over all ( at ?camion ?ciudad)) )
:effect (and (at start(not (at ?paquete ?ciudad)))
              (at end (in ?paquete ?camion))(at end (increase (coste) 1)))
)
```

Para definir una función tenemos que indicar que parámetros vamos a usar. Para nuestra función cargar camión necesitaremos un camión un paquete y una ciudad (un camión sólo se puede cargar si esta en una ciudad). A continuación se indica la duración que tiene la acción. La siguiente parte consiste en definir las condiciones para que se pueda ejecutar el plan. Las condiciones precedidas de la cláusula “at start” son aquellas condiciones que deben darse antes de empezar la acción. Cómo ya hemos comentado vamos a exigir que el camión este en una ciudad. Asimismo también exigimos que haya un paquete en la misma ciudad. Nótese que para comprobar los requisitos estamos haciendo uso de los predicados que hemos definido anteriormente. Estos predicados se inicializarán en cada problema de manera independiente. A continuación tenemos un predicado precedido de la cláusula “over all”. Eso indica que es un predicado que debe cumplirse durante TODA la acción. Lo que indica al sistema es que durante toda la acción de cargar el camión el camión debe permanecer en la misma ciudad.

Para terminar , solo nos queda definir los efectos que tendrá la acción. De nuevo aparece la cláusula “at start” que indica los efectos inmediatos al empezar a realizar la acción. Análogamente la cláusula “at end” indica los efectos causados cuando se termine de llevar a cabo la acción.

Los efectos de nuestra acción cargar son tres. El primer efecto , de carácter inmediato es que el paquete deja de estar en la ciudad (de éste modo ningún otro camión intentará cargar dicho paquete). El segundo efecto, al final de la acción es que el paquete pasa a estar dentro del camión. Por último se incrementa el coste total de las acciones mediante el uso de “increase”.

Descargar camión

La definición de esta función es bastante similar a la de cargar camión:

```
(:durative-action descargarCamion ;Función para modelar la acción de descargar de un camion
:parameters (?paquete - paquete ?ciudad - ciudad ?camion - camion)
:duration (= ?duration 1)
:condition (and (at start ( at ?camion ?ciudad))
                (at start (in ?paquete ?camion))
                (over all ( at ?camion ?ciudad)))
:effect (and (at start(not (in ?paquete ?camion)))
             (at end (at ?paquete ?ciudad)) (at end (increase (coste) 1)))
)
```

En este caso exigiremos que haya un paquete en un camión y que dicho camión esté en una ciudad. Igual que en el caso anterior exigimos que el camión permanezca en la ciudad durante todo el proceso. Los efectos ahora son ,contrariamente al caso anterior, que el paquete deja de estar en el camión y pasa a estar en la ciudad. La última parte, es el incremento del coste total permanece igual.

A continuación se incluye la declaración del resto de acciones que siguen una filosofía similar a las explicadas.

Subir a un camión

```
(:durative-action subirCamion ;Función para modelar la acción de subir a un camion

:parameters (?conductor - conductor ?ciudad - ciudad ?camion - camion)

:duration (= ?duration 1)

:condition (and (at start (at ?conductor ?ciudad))
                (at start (vacio ?camion))
                (at start ( at ?camion ?ciudad))
                (over all ( at ?camion ?ciudad)) )

:effect (and (at start (not (at ?conductor ?ciudad)))
              (at start (not (vacio ?camion)))
              (at end (in ?conductor ?camion))
              (at end (increase (coste) 1)))

)
```

En esta acción vamos a exigir que un camión esté vacío. De este modo controlamos que en un camión sólo va un conductor. Los efectos al final de la acción son que el camión deja de estar vacío y que el conductor pasa a estar dentro del camión.

Bajar de un camión

```
(:durative-action bajarCamion ;Función para modelar la acción de bajar de un camion
:parameters (?conductor - conductor ?ciudad - ciudad ?camion - camion)
:duration (= ?duration 1)
:condition (and (at start ( at ?camion ?ciudad))
                (at start (in ?conductor ?camion))
                (over all ( at ?camion ?ciudad)))
:effect (and (at start(not (in ?conductor ?camion)))
             (at end (at ?conductor ?ciudad)) (at end (vacio ?camion))(at end (increase (coste) 1))))
```

La definición de esta función sigue la filosofía de la acción subirCamión . Podemos observar cómo al final de la acción el camión pasa a estar vacío.

Viajar con un camión

```
(:durative-action viajar                                     ;Función para modelar el desplazamiento entre ciudades

:parameters (?camion - camion ?origen ?destino - ciudad ?conductor - conductor)

:duration (= ?duration (distancia ?origen ?destino))

:condition (and (at start(at ?camion ?origen))

                (over all (in ?conductor ?camion))

                (over all (conectado ?origen ?destino))

                )

:effect (and (at start(not (at ?camion ?origen)))

             (at end (at ?camion ?destino))(at end (increase (coste) (costeViaje ?origen ?destino))))

)
```

Esta acción sirve para modelar el viaje con un camión entre dos ciudades. Para poder realizar un viaje un camión debe tener un conductor y debe haber una conexión entre dichas ciudades. Los efectos són que el camión pasará a estar en la ciudad destino. A diferencia de las otras funciones, la duración viene determinada por una función. La función distancia , definida anteriormente, devolverá el tiempo necesario para llevar a cabo el viaje , y , por tanto la duración del viaje. También se hace uso de una función para incrementar el coste. La función costeViaje nos devolverá el coste de viajar entre una ciudad y otra.

Viajar con un camión

```
(:durative-action autobus ;Función para modelar el desplazamiento d eun conductor en autobus
:parameters (?conductor - conductor ?origen ?destino - ciudad)
:duration (= ?duration 10)
:condition (and (at start (at ?conductor ?origen))
                (over all (conectado ?origen ?destino)))
:effect (and (at start (not (at ?conductor ?origen)))
             (at end (at ?conductor ?destino)) (at end (increase (coste) 3)))
)
```

La última acción de nuestro dominio. Esta acción es muy parecida a la acción de viajar con un camión . Sin embargo no es necesario estar en un camión . En este dominio se asume que las conexiones entre ciudades son también conexiones en autobús. Sin embargo a diferencia de la acción viajar, el coste y la duración de un viaje son fijos.

Llegados a este punto ya tenemos definido el dominio. A partir de aquí se pueden definir problemas que funcionen con las acciones que hemos especificado. A continuación vamos a ver cómo modelar el problema propuesto.

Modelado de un problema sobre el dominio

En este apartado vamos a mostrar cómo hemos modelado el problema del transporte de paquetes entre ciudades.

Para empezar el archivo, abrimos con las siguientes líneas:

```
(define (problem transporte)
```

```
(:domain Camiones)
```

Con la primera línea vamos a indicar el nombre del problema. La segunda línea indica que dominio vamos a usar. El nombre del dominio debe ser necesariamente el mismo que el que hay en el archivo dónde hemos definido el dominio.

A continuación tenemos que indicar los objetos que formarán parte del problema. Esto significa añadir la información sobre los camiones , los paquetes , los conductores y las ciudades:

```
(:objects                                ;Objetos que forman parte del problema
```

```
    camion1 - camion
```

```
    camion2 - camion
```

```
    conductor1 - conductor
```

```
    conductor2 - conductor
```

```
    paquete1 - paquete
```

```
    paquete2 - paquete
```

```
    ciudadA - ciudad
```

```
    ciudadB - ciudad
```

```
    ciudadC - ciudad
```


ciudadD - ciudad

ciudadE - ciudad)

Ahora que ya tenemos definidos los objetos que interactuarán en el problema, tenemos que definir el estado inicial del problema. Esto significa especificar dónde están los camiones, los paquetes, los conductores y las conexiones entre las ciudades. Para definir estos estados iniciales hacemos uso de los predicados y las funciones definidas en el dominio.

```
(:init                                     ;Estado inicial del problema
```

```
    ;Situacion inicial de los objetos
```

```
        (at camion1 ciudadA)
```

```
        (at camion2 ciudadA)
```

```
        (at conductor1 ciudadC)
```

```
        (at conductor2 ciudadD)
```

```
        (at paquete1 ciudadA)
```

```
        (at paquete2 ciudadD)
```

```
        (vacio camion1)
```

```
        (vacio camion2)
```

```
    ;Conexion entre ciudades
```

(conectado ciudadA ciudadB)

(conectado ciudadA ciudadC)

(conectado ciudadB ciudadA)

(conectado ciudadB ciudadC)

(conectado ciudadB ciudadD)

(conectado ciudadB ciudadE)

(conectado ciudadC ciudadA)

(conectado ciudadC ciudadB)

(conectado ciudadC ciudadE)

(conectado ciudadD ciudadB)

(conectado ciudadE ciudadB)

(conectado ciudadE ciudadC)

;Distancias entre ciudades

(= (distancia ciudadA ciudadB) 5)

(= (distancia ciudadA ciudadC) 3)

(= (distancia ciudadB ciudadA) 5)

(= (distancia ciudadB ciudadC) 3)

(= (distancia ciudadB ciudadD) 2)

(= (distancia ciudadB ciudadE) 4)

(= (distancia ciudadC ciudadA) 3)

(= (distancia ciudadC ciudadB) 3)

(= (distancia ciudadC ciudadE) 8)

(= (distancia ciudadD ciudadB) 2)

(= (distancia ciudadE ciudadB) 4)

(= (distancia ciudadE ciudadC) 8)

;Coste de viajar entre cada ciudad

(= (costeViaje ciudadA ciudadB) 2)

(= (costeViaje ciudadA ciudadC) 1)

```
(= (costeViaje ciudadB ciudadA) 2)
(= (costeViaje ciudadB ciudadC) 1)
(= (costeViaje ciudadB ciudadD) 2)
(= (costeViaje ciudadB ciudadE) 3)

(= (costeViaje ciudadC ciudadA) 1)
(= (costeViaje ciudadC ciudadB) 1)
(= (costeViaje ciudadC ciudadE) 5)

(= (costeViaje ciudadD ciudadB) 2)

(= (costeViaje ciudadE ciudadB) 3)
(= (costeViaje ciudadE ciudadC) 5)
```

```
);fin declaración estado inicial
```

Podemos distinguir dos tipos de líneas. Las que sirven para definir predicados y los que asignan valores a las funciones.

Para terminar sólo nos queda definir los objetivos del problema y la métrica de calidad. Los objetivos que hemos definido son los siguientes:

;Declaración de los objetivos dle problema

```
(:goal (and  
  (at paquete1 ciudadE)  
  (at paquete2 ciudadC)  
  (at camion2 ciudadA)  
  (at conductor1 ciudadB)))
```

La definición de los objetivos es bastante sencilla. Lo único que hacemos es indicar que predicados deben cumplirse para completar el problema. Finalmente la definición de la métrica de calidad es la siguiente:

;Métrica para medir la calidad del plan

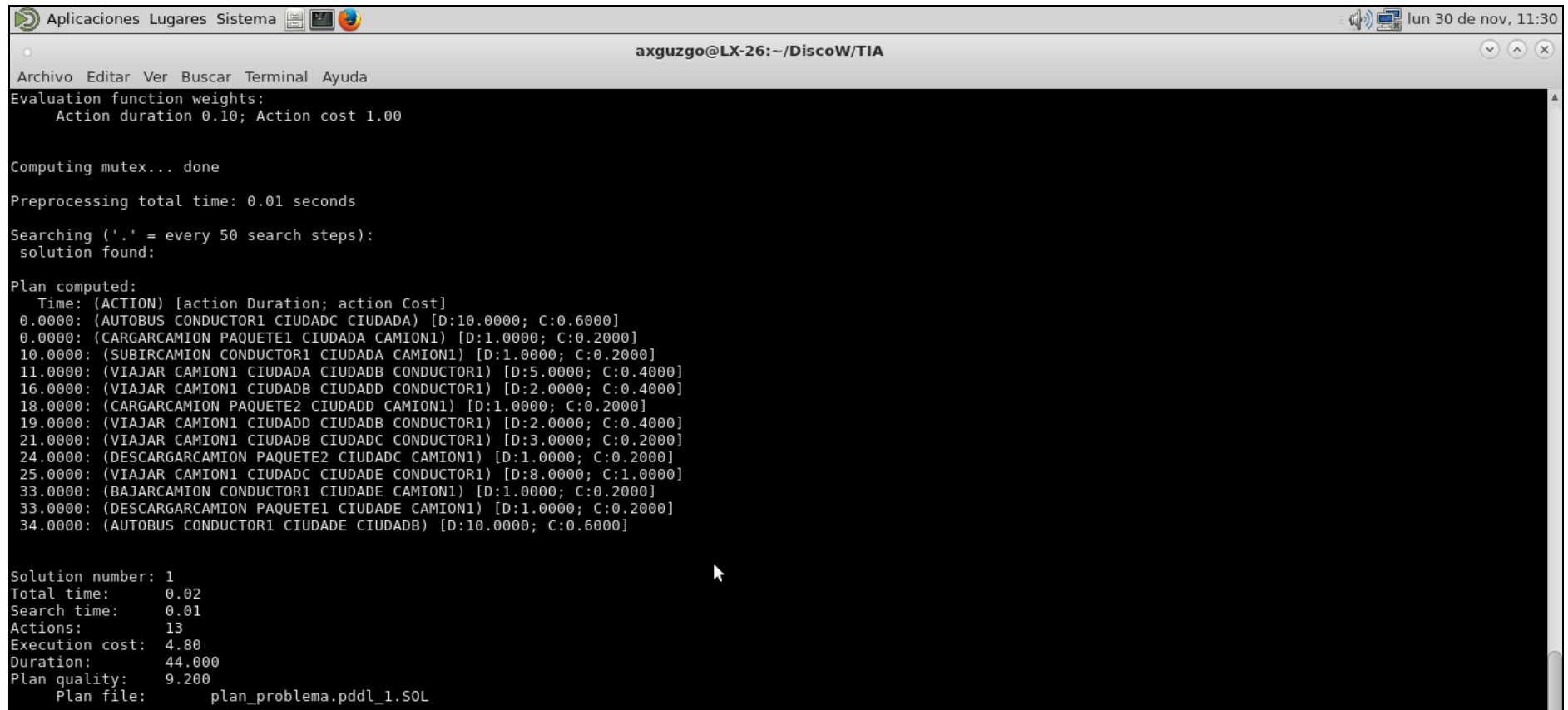
```
(:metric minimize (+ (* 0.1 (total-time)) (* 0.2 (coste))))  
)
```

Este apartado sirve para indicar cómo se mide la calidad del plan. Mediante “minimize” indicamos que queremos reducir el máximo posible la expresión indicada.

Ahora sólo queda realizar una ejecución y comprobar los resultados.

Ejecución del problema

Tras ejecutar el problema con el planificador obtenemos los resultados siguientes:



```
Aplicaciones Lugares Sistema axguzgo@LX-26:~/DiscoW/TIA
Archivo Editar Ver Buscar Terminal Ayuda
Evaluation function weights:
  Action duration 0.10; Action cost 1.00

Computing mutex... done

Preprocessing total time: 0.01 seconds

Searching ('.' = every 50 search steps):
  solution found:

Plan computed:
  Time: (ACTION) [action Duration; action Cost]
0.0000: (AUTOBUS CONDUCTOR1 CIUDADC CIUDADA) [D:10.0000; C:0.6000]
0.0000: (CARGARCAMION PAQUETE1 CIUDADA CAMION1) [D:1.0000; C:0.2000]
10.0000: (SUBIRCAMION CONDUCTOR1 CIUDADA CAMION1) [D:1.0000; C:0.2000]
11.0000: (VIAJAR CAMION1 CIUDADA CIUDADB CONDUCTOR1) [D:5.0000; C:0.4000]
16.0000: (VIAJAR CAMION1 CIUDADB CIUDADD CONDUCTOR1) [D:2.0000; C:0.4000]
18.0000: (CARGARCAMION PAQUETE2 CIUDADD CAMION1) [D:1.0000; C:0.2000]
19.0000: (VIAJAR CAMION1 CIUDADD CIUDADB CONDUCTOR1) [D:2.0000; C:0.4000]
21.0000: (VIAJAR CAMION1 CIUDADB CIUDADC CONDUCTOR1) [D:3.0000; C:0.2000]
24.0000: (DESCARGARCAMION PAQUETE2 CIUDADC CAMION1) [D:1.0000; C:0.2000]
25.0000: (VIAJAR CAMION1 CIUDADC CIUADE CONDUCTOR1) [D:8.0000; C:1.0000]
33.0000: (BAJARCAMION CONDUCTOR1 CIUADE CAMION1) [D:1.0000; C:0.2000]
33.0000: (DESCARGARCAMION PAQUETE1 CIUADE CAMION1) [D:1.0000; C:0.2000]
34.0000: (AUTOBUS CONDUCTOR1 CIUADE CIUDADB) [D:10.0000; C:0.6000]

Solution number: 1
Total time: 0.02
Search time: 0.01
Actions: 13
Execution cost: 4.80
Duration: 44.000
Plan quality: 9.200
Plan file: plan_problema.pddl_1.SOL
```

Conclusiones

Realizar esta práctica resulta un poco difícil al principio por ser un modo nuevo de programar. Sin embargo los ejemplos proporcionados han sido de gran utilidad a la hora de comprender cómo funciona PDDL. Quizás estaría bien dar un poco más de información sobre el funcionamiento de los predicados y las funciones, ya que al principio es un poco confuso. Respecto a la implementación del código , he tenido algunos problemas según el planificador que usaba. Parece ser que el planificador MIPS-xxl es más estricto respecto al uso de paréntesis.

El resultado final ha sido el aprendizaje del funcionamiento de PDDL por lo que las sensaciones son positivas. A continuación se incluye el código completo de los dos ficheros como anexo

Anexo: código completo de los dos ficheros.

Fichero Camiones.pddl

```
; Dominio : Camiones
```

```
; Autor Axel Guzman Godia
```

```
(define (domain Camiones)
```

```
(:requirements :typing :durative-actions :fluents)
```

```
(:types camion conductor ciudad paquete) ; Especificamos nuestros objetos
```

```
(:predicates (at ?x - (either conductor camion paquete) ?c - ciudad)
```

```
          (in ?x - (either paquete conductor) ?cam - camion)
```

```
          (vacio ?camion - camion)
```

```
          (conectado ?c1 - ciudad ?c2 - ciudad )
```

```
)
```

```
(:functions (distancia ?c1 - ciudad ?c2 - ciudad) ;función numérica para conocer la distancia entre 2 ciudades
```

```
          (costeViaje ?c1 - ciudad ?c2 - ciudad) ;función numérica para conocer el coste de viajar entre 2 ciudades
```



```
(coste) ;función para calcular el coste total

)
```

```
(:durative-action subirCamion ;Función para modelar la acción de subir a un camion

:parameters (?conductor - conductor ?ciudad - ciudad ?camion - camion)

:duration (= ?duration 1)

:condition (and (at start (at ?conductor ?ciudad))
                (at start (vacio ?camion))
                (at start ( at ?camion ?ciudad))
                (over all ( at ?camion ?ciudad)) )

:effect (and (at start (not (at ?conductor ?ciudad)))
              (at start (not (vacio ?camion)))
              (at end (in ?conductor ?camion))
              (at end (increase (coste) 1)))

)
```

```
(:durative-action bajarCamion ;Función para modelar la acción de bajar de un camion

:parameters (?conductor - conductor ?ciudad - ciudad ?camion - camion)
```

```

:duration (= ?duration 1)

:condition (and (at start ( at ?camion ?ciudad))

                (at start (in ?conductor ?camion))

                (over all ( at ?camion ?ciudad)))

:effect (and (at start(not (in ?conductor ?camion)))

             (at end (at ?conductor ?ciudad)) (at end (vacio ?camion))(at end (increase (coste) 1))))

```

```

(:durative-action cargarCamion                                ;Función para modelar la acción de cargar en un camion

:parameters (?paquete - paquete ?ciudad - ciudad ?camion - camion)

:duration (= ?duration 1)

:condition (and (at start ( at ?camion ?ciudad))(at start ( at ?paquete ?ciudad))

                (over all ( at ?camion ?ciudad)) )

:effect (and (at start(not (at ?paquete ?ciudad)))

             (at end (in ?paquete ?camion))(at end (increase (coste) 1)))

)

```

```

(:durative-action descargarCamion                            ;Función para modelar la acción de descargar de un camion

:parameters (?paquete - paquete ?ciudad - ciudad ?camion - camion)

:duration (= ?duration 1)

```

```

:condition (and (at start ( at ?camion ?ciudad))
                (at start (in ?paquete ?camion))
                (over all ( at ?camion ?ciudad)))
:effect (and (at start(not (in ?paquete ?camion)))
             (at end (at ?paquete ?ciudad)) (at end (increase (coste) 1)))
)

```

```

(:durative-action viajar                                     ;Función para modelar el desplazamiento entre ciudades
:parameters (?camion - camion ?origen ?destino - ciudad ?conductor - conductor)
:duration (= ?duration (distancia ?origen ?destino))
:condition (and (at start(at ?camion ?origen))
                (over all (in ?conductor ?camion))
                (over all (conectado ?origen ?destino))
                )
:effect (and (at start(not (at ?camion ?origen)))
             (at end (at ?camion ?destino))(at end (increase (coste) (costeViaje ?origen ?destino))))
)

```

```

(:durative-action autobus                                   ;Función para modelar el desplazamiento d eun conductor en autobus

```

```

:parameters (?conductor - conductor ?origen ?destino - ciudad)

:duration (= ?duration 10)

:condition (and (at start (at ?conductor ?origen))
                (over all (conectado ?origen ?destino)))

:effect (and (at start (not (at ?conductor ?origen)))
             (at end (at ?conductor ?destino)) (at end (increase (coste) 3)))

)

)

```

Fichero Problema.pddl

;Aqui se implementa el problema de transportes

```
(define (problem transporte)
```

```
(:domain Camiones)
```

```
(:objects ;Objetos que forman parte del problema
```

```
camion1 - camion
camion2 - camion
conductor1 - conductor
conductor2 - conductor
paquete1 - paquete
paquete2 - paquete
ciudadA - ciudad
ciudadB - ciudad
ciudadC - ciudad
ciudadD - ciudad
ciudadE - ciudad)
```

```
(:init                                     ;Estado inicial del problema
;Situacion inicial de los objetos
    (at camion1 ciudadA)
    (at camion2 ciudadA)
    (at conductor1 ciudadC)
    (at conductor2 ciudadD)
    (at paquete1 ciudadA)
```

(at paquete2 ciudadD)

(vacio camion1)

(vacio camion2)

;Conexion entre ciudades

(conectado ciudadA ciudadB)

(conectado ciudadA ciudadC)

(conectado ciudadB ciudadA)

(conectado ciudadB ciudadC)

(conectado ciudadB ciudadD)

(conectado ciudadB ciudadE)

(conectado ciudadC ciudadA)

(conectado ciudadC ciudadB)

(conectado ciudadC ciudadE)

(conectado ciudadD ciudadB)

```
(conectado ciudadE ciudadB)
```

```
(conectado ciudadE ciudadC)
```

```
;Distancias entre ciudades
```

```
(= (distancia ciudadA ciudadB) 5)
```

```
(= (distancia ciudadA ciudadC) 3)
```

```
(= (distancia ciudadB ciudadA) 5)
```

```
(= (distancia ciudadB ciudadC) 3)
```

```
(= (distancia ciudadB ciudadD) 2)
```

```
(= (distancia ciudadB ciudadE) 4)
```

```
(= (distancia ciudadC ciudadA) 3)
```

```
(= (distancia ciudadC ciudadB) 3)
```

```
(= (distancia ciudadC ciudadE) 8)
```

```
(= (distancia ciudadD ciudadB) 2)
```

```
(= (distancia ciudadE ciudadB) 4)
```

```
(= (distancia ciudadE ciudadC) 8)
```

```
;Coste de viajar entre cada ciudad
```

```
(= (costeViaje ciudadA ciudadB) 2)
```

```
(= (costeViaje ciudadA ciudadC) 1)
```

```
(= (costeViaje ciudadB ciudadA) 2)
```

```
(= (costeViaje ciudadB ciudadC) 1)
```

```
(= (costeViaje ciudadB ciudadD) 2)
```

```
(= (costeViaje ciudadB ciudadE) 3)
```

```
(= (costeViaje ciudadC ciudadA) 1)
```

```
(= (costeViaje ciudadC ciudadB) 1)
```

```
(= (costeViaje ciudadC ciudadE) 5)
```

```
(= (costeViaje ciudadD ciudadB) 2)
```

```
(= (costeViaje ciudadE ciudadB) 3)
```



```
(= (costeViaje ciudadE ciudadC) 5)
```

```
);fin declaración estado inicial
```

```
;Declaración de los objetivos dle problema
```

```
(:goal (and
```

```
    (at paquete1 ciudadE)
```

```
    (at paquete2 ciudadC)
```

```
    (at camion2 ciudadA)
```

```
    (at conductor1 ciudadB)))
```

```
;Métrica para medir la calidad del plan
```

```
(:metric minimize (+ (* 0.1 (total-time)) (* 0.2 (coste))))
```

```
)
```