Práctica 4 – Planificadores

Para la primera parte de la práctica vamos a hacer una comparación entre los dos planificadores que nos han proporcionado sobre los casos de prueba de Rovers, Storage y Pipes los resultados de los cuales vamos a mostrarlos en la tabla siguiente y posteriormente vamos a analizarlos.

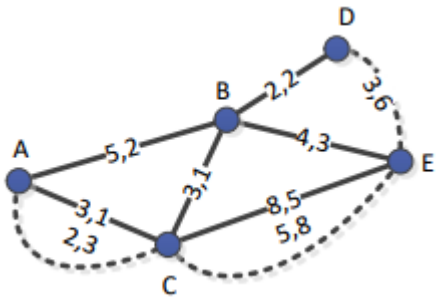
		Rover				Storage				Pipes			
		1	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
LPG	Acciones	10	9	11	8	3	6	3	24	6	312	19	19
	Tiempo	0.025	0.028	0.028	0.028	0.012	0.019	0.022	0.027	0.035	0.401	0.096	0.087
	Duración	75	71	80	60	3	6	3	40	12	606	34	34
MIPS	Acciones	10	8	13	8	3	3	3	8	5	14	9	13
	Tiempo	0.053	0.125	0.095	0.069	0.044	0.073	0.1	0.067	0.093	0.08	0.18	0.27
	Duración	57.05	47.04	67.05	38.03	3.00	3.00	3.00	10.00	6.02	24.11	14.06	22.1

La comparación de los resultados mostrados anteriormente es bastante fácil ya que si nos guiamos por la variable del numero de acciones se ve claramente que en el caso del Rover no se aleja mucho la una de la otra, pero cuando nos fijamos en el ejemplo de Pipes vemos que la reducción de LPG a MIPS es abismal (i.e. de 312 pasa a 14 acciones en el MIPS).

Si nos basamos en tiempo de ejecución o en la duración para algo muy parecido a lo descrito antes, pero en proporciones diferentes, con todo esto podemos tener claro que un planificador determinista hace eficientes los problemas planteados

Parte 2: Escenario de Logística

Vamos a modelar el siguiente problema para la gestión de unos camiones que tienen que transportar paquetes de ciudad en ciudad y gestionar tanto la carga y descarga del paquete como la subida y bajada de los conductores, controlando el tiempo y el coste de todas las acciones.



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

```
(define (domain transporte)
    (:requirements :durative-actions :typing :fluents)
    (:types paquete camion driver ciudad -object)
    (:predicates
        (at ?x - (either paquete camion driver) ?c - ciudad)
        (in ?p - (either paquete driver) ?c - camion))
    (:functions
        (distance ?c1-ciudad ?c2-ciudad)
        (coste ?p-ciudad ?c2-ciudad)
        (coste-total))
```

En la definición general marcamos las funciones que vamos a usar como el coste de ir de una ciudad a otra o la función distance que nos dará las unidades que tarde en ir de una ciudad de una a otra.

```
(:durative-action cargar
    :parameters (?p - paquete ?c - camion ?ciu - ciudad )
    :duration  (= ?duration 1)
    :condition (and (at start (at ?p ?ciu))
                    (over all (at ?c ?ciu)))
    :effect (and (at start (not(at ?p ?ciu)))
                 (at end (in ?p ?c))
                 (at end (increase (coste-total) 1))
                 )
    )

(:durative-action descargar
    :parameters (?p - paquete ?c - camion ?ciu - ciudad )
    :duration  (= ?duration 1)
    :condition (and (at start (in ?p ?c))
                    (over all(at ?c ?ciu)))
    :effect (and (at start (not(in ?p ?c)))
                 (at end (at ?p ?ciu))
                 (at end (increase (coste-total) 1))
                 )
    )
```

Las funciones cargar y descargar que comprueban si el paquete esta en la ciudad y después pone o quita el paquete en el camión después incrementa las variables de tiempo y coste en el numero de unidades que nos indica la práctica.

```
(:durative-action subir
    :parameters (?c - camion ?d - driver ?ciu - ciudad)
    :duration  (= ?duration 1)
    :condition (and (at start (at ?c ?ciu))
                    (at start (at ?d ?ciu)) )
    :effect (and (at start (not (at ?d ?ciu)))
                 (at end (in ?d ?c))
                 (at end (increase (coste-total) 1))
                 )
    )

(:durative-action bajar
    :parameters (?c -camion ?d -driver ?ciu - ciudad)
    :duration  (= ?duration 1)
    :condition (and (at start (in ?d ?c))
                    (at start (at ?c ?ciu)) )
    :effect (and (at start (not(in ?d ?c)))
                 (at end (at ?d ?ciu))
                 (at end(increase (coste-total) 1))
                 )
```

El método de subir controla que todo este en la misma ciudad e indica que a partir de ahora el conductor estará en ese camión y no en la ciudad, volvemos a incrementar las variables de coste y tiempo como en la parte anterior.

```
(:durative-action viajar
    :parameters (?d - driver ?c1 - ciudad ?c2 - ciudad)
    :duration  (= ?duration 10)
    :condition (at start (at ?d ?c1))
    :effect (and (at start (not(at ?d ?c1)))
                 (at end (at ?d ?c2))
                 (at end (increase (coste-total) 3))
                 )
    )

(:durative-action coducir
    :parameters (?c - camion ?d - driver ?c1 - ciudad ?c2 - ciudad)
    :duration  (= ?duration (distance ?c1 ?c2))
    :condition (and (at start (at ?c ?c1)) (at start (in ?d ?c)))
    :effect (and (at start (not(at ?c ?c1)))
                 (at end (at ?c ?c2))
                 (at end (increase (coste-total) (coste ?c1 ?c2)))
                 )
    )
```

Y finalmente las funciones de viajar en autobús y conducir el camión, la primera tiene una duración de 10 unidades y comprueba que el camión está en nuestra misma ciudad después elimina el camión de la ciudad y al final le indica que el bus está en otra ciudad, finalmente incrementamos las variables con el coste y el tiempo fijos que nos da la definición del problema. La segunda es la misma que la primera, pero en el calculo de el coste y el tiempo le añadimos la función que nos devuelve las unidades de cada variable de una determinada carretera de una ciudad a otra.

Pruebas:

Planificador	Acciones	Tiempo	Duración
LPG	10	0.431	14.00
MIPS			

Las pruebas sobre el LPG están hechas con el siguiente código, pero como puedes ver las pruebas del MIPS no han podido hacerse ya que me decía que no había solución posible, he intentado cambiar las pruebas y los *"goals"* para ver que era lo que podía pasar y me seguía dando error.

```
(define (problem problema-logistica)
    (:domain transporte)
    (:objects
        p1 p2 - paquete
        c1 c2 - camion
        d1 d2 - driver
        A B C D E - ciudad
    )
    (:init
        (at p1 A)
        (at p2 D)
        (at c1 A)
        (at c2 A)
        (at d1 C)
        (at d2 D)
        (= (distance A A) 0)
        (= (distance A B) 5)
        (= (distance A C) 3)

        (= (distance B B) 0)
        (= (distance B A) 5)
        (= (distance B C) 3)
        (= (distance B D) 2)
        (= (distance B E) 4)

        (= (distance C A) 3)
        (= (distance C B) 3)
        (= (distance C C) 0)
        (= (distance C E) 8)

        (= (distance D D) 0)
        (= (distance D B) 2)

        (= (distance E C) 8)
        (= (distance E B) 4)
        (= (distance E E) 0)

        (= (coste A A) 0)
        (= (coste A B) 2)
        (= (coste A C) 1)

        (= (coste B B) 0)
        (= (coste B A) 2)
        (= (coste B C) 1)
        (= (coste B D) 2)
        (= (coste B E) 3)

        (= (coste C A) 1)
        (= (coste C B) 1)
        (= (coste C C) 0)
        (= (coste C E) 5)

        (= (coste D D) 0)
        (= (coste D B) 2)

        (= (coste E C) 5)
        (= (coste E B) 3)
        (= (coste E E) 0)


        (= (coste-total) 0)
    )
    (:goal (and (at p1 E)
        (at p2 C)
        (at c2 A)
        (at d1 B))
    )
    (:metric minimize (+ (* 0.1 (total-time))(* 0.2 (coste-total))))
)
```