Planificación Automática

PECL1

Curso 2021-2022

Isabel Blanco Martínez

Mario Pacheco Benito

Patricia Cuesta Ruiz

ÍNDICE

Parte 1: Planificación clásica con PDDL

Ejercicio 1.1. Logística de servicio de emergencias, versión inicial

Para la generación del dominio, hemos seguido el siguiente código que adjuntamos en capturas de pantalla y cuyo nombre de archivo es "drone-problem.pddl". El nombre del dominio se llamará "drone-dom".

```
(define (domain drone-dom)

    (:requirements
        :typing
    )

    Show hierarchy
    (:types
        drone human box content location - object
    )
```

Para comenzar, solamente hemos añadido la extensión ":typing", ya que sí está globalmente soportada por los distintos planificadores.

Los tipos de objetos que hemos declarado son:

- drone → que son los drones que van a intervenir para llevar cajas.
- human → el número de humanos que habrá repartidos por diferentes localizaciones.
- box → donde se guardará cierto contenido que pedirán los humanos.
- content → es el contenido que irá guardado en las propias cajas. Los humanos no pedirán cajas si no que pedirán contenidos que irán dentro de cajas.
- location → diferentes sitios donde se localizan los seres humanos. Además de esto, se encontrará el almacén donde se encuentra el dron y las cajas inicialmente.

Los predicados que hemos instanciado son los siguientes:

```
    (:predicates
        (drone-at ?d - drone ?l - location) 3👁 1☑ 1☒
        (drone-free ?d - drone) 1👁 1☑ 1☒
        (drone-carry ?d - drone ?b - box) 1👁 1☑ 1☒
        (human-at ?h - human ?l - location) 1👁
        (human-has ?h - human ?c - content) 1☑
        (box-at ?b - box ?l - location) 1👁 1☑ 1☒
        (box-has ?b - box ?c - content) 2👁
        (box-free ?b - box) 1👁 1☒
    )
```

- drone-at → que nos indicará dónde se encuentra el dron (en una localización del tipo loc(i) o en el almacén (warehouse).
- drone-free → nos devolverá si el dron está libre o por el contrario,
- drone-carry → el robot lleva una caja para llevarla a algún humano.
- human-at → devolverá la ubicación donde se encuentre el humano.
- human-has → nos servirá para comprobar si el humano necesita algún contenido.
- box-at → devolverá la ubicación de la caja.
- box-has → devolverá el contenido de la caja (para asegurarnos antes de tirarla).
- box-free → será útil porque necesitamos saber que la caja que hemos tirado ha sido usada para que el robot no intente buscarla para volver a usarla.

A contemplar, que teníamos un predicado llamado "human-needs" con lo que el humano necesita (medicina, comida, etc.). Esto es inútil ya que con el predicado human-has ya cubrimos la necesidad de saber si el humano necesita ese contenido o no al incluirlo en los *goals* de los problemas.

La acción "move" ayudará al robot a moverse desde una posición inicial a una posición final.

La única precondición necesaria es que el robot se encuentre en la posición desde la que nos queremos mover. El resultado será movernos a la posición que el planificador crea necesario.

```
(:action move
    :parameters (?d - drone ?from ?to - location)
    :precondition (and
        (drone-at ?d ?from)
    )
    :effect (and
        (drone-at ?d ?to)
        (not (drone-at ?d ?from))
    )
)
```

La acción que ayudará al dron a coger una caja será "pick-up". Necesitamos saber si la caja que va a coger está en esa ubicación y si el robot está vacío ya que si la caja no se encuentra en ese lugar o el robot va lleno, no va a ser posible coger la caja.

El resultado será indicar que la caja se encuentra en el robot y que el robot va con la caja que ha cogido.

```
    (:action pick-up
        :parameters (?d - drone ?b - box ?c - content ?l - location)
        :precondition (and
            (drone-at ?d ?l)
            (box-at ?b ?l)
            (drone-free ?d)
            (box-has ?b ?c)
            (box-free ?b)
        )
        :effect (and
            (drone-carry ?d ?b)
            (not (drone-free ?d))
            (not (box-at ?b ?l))
        )
    )
```

La acción contraria a coger una caja, será soltarla. La acción “drop” tendrá el resultado contrario que la acción “pick-up”. Si el dron contiene la caja, la soltará.

Aquí es importante remarcar que al humano que le soltemos la caja necesita el contenido que lleva la caja. Si todo esto es así, el humano recibe la caja y ya no necesitará más ese contenido.

```
    (:action drop
        :parameters (?d - drone ?b - box ?c - content ?l - location ?h - human)
        :precondition (and
            (drone-carry ?d ?b)
            (drone-at ?d ?l)
            (human-at ?h ?l)
            (box-has ?b ?c)
        )
        :effect (and
            (drone-free ?d)
            (box-at ?b ?l)
            (human-has ?h ?c)
            (not (drone-carry ?d ?b))
            (not (box-free ?b))
        )
    )
)
```

Para la generación del problema, hemos llamado al problema drone-problem1, ya que es la parte 1. Y el nombre del dominio, drone-dom, que coincide con lo puesto en el dominio. Este código se encuentra en el archivo “drone-problem.pddl”.

```
(define (problem drone-problem1) (:domain drone-dom)
    Show hierarchy
    (:objects
        drone1 - drone
        box1 box2 box3 box4 box5 box6 - box
        warehouse loc1 loc2 loc3 loc4 - location
        human1 human2 human3 human4 human5 - human
        clothes water food meds - content
    )
```

Por el momento hemos definido los objetos que se pueden ver solamente para comprobar que el planificador hace correctamente el problema.

Al inicializarlo, indicamos que tanto el robot como todas las cajas se encuentran en el almacén. Es importante indicar que el robot no contiene ninguna caja (porque inicialmente está vacío).

También es importante indicar el contenido de las cajas con lo que nosotros hemos considerado, teniendo en cuenta que en base a lo que haya va a ser lo que va a repartir. Que las cajas estén “libres” nos indica que todavía no han sido asignadas a ningún usuario y que por tanto pueden ser cogidas por el robot para entregarlas.

```
    View
    (:init
        ;todo: put the initial state's facts and numeric values here
        (drone-at drone1 warehouse)
        (drone-free drone1)
        (box-at box1 warehouse)
        (box-at box2 warehouse)
        (box-at box3 warehouse)
        (box-at box4 warehouse)
        (box-at box5 warehouse)
        (box-at box6 warehouse)
        (box-has box1 meds)
        (box-has box2 water)
        (box-has box3 clothes)
        (box-has box4 food)
        (box-has box5 water)
        (box-has box6 meds)
        (box-free box1)
        (box-free box2)
        (box-free box3)
        (box-free box4)
        (box-free box5)
        (box-free box6)
```

Por último, indicar en qué localización se encontrarán las personas (teniendo en cuenta que es imposible que se estén en el almacén).

```
        (human-at human1 loc1)
        (human-at human2 loc1)
        (human-at human3 loc2)
        (human-at human4 loc3)
        (human-at human5 loc4)
    )
```

Los objetivos están claros: que todos los usuarios queden abastecidos y que el robot acabe en el almacén.

La segunda cosa es fácil, pero la primera debemos de introducirlas en función de lo que hemos inicializado en el init.

```
    (:goal (and
        ;todo: put the goal condition here
        (drone-at drone1 warehouse)
        (human-has human1 meds)
        (human-has human1 water)
        (human-has human2 clothes)
        (human-has human3 food)
        (human-has human4 water)
        (human-has human5 meds)
    ))

)
```

Ejercicio 1.2. Generador de problemas en Python

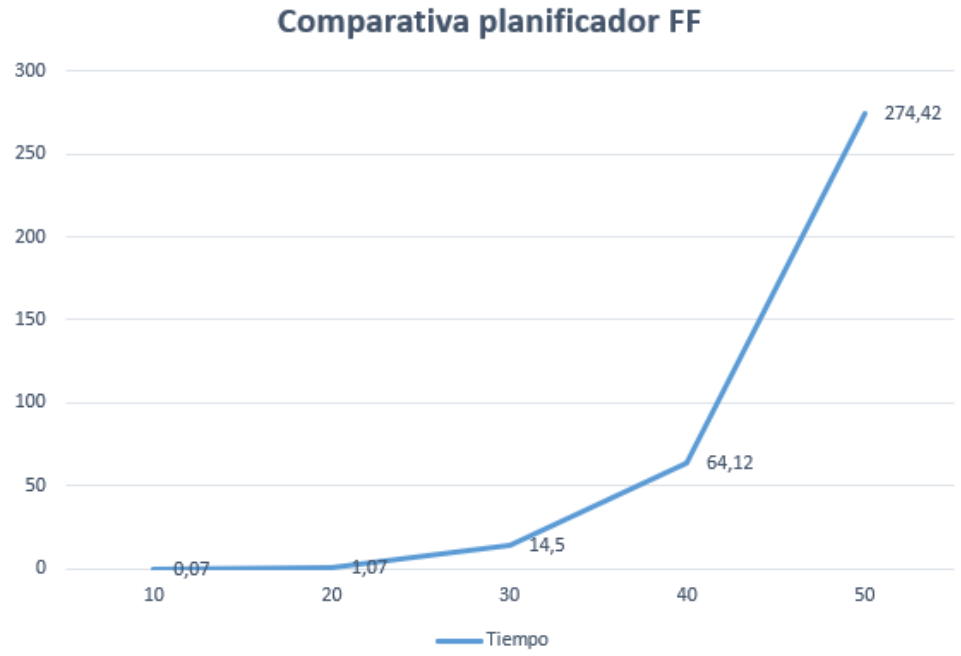
Mediante el siguiente comando variamos la generación de problemas:

```
python ./generate-problem.py -d X -r X -l X -p X -c X -g X
```

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	0	10	10	10	10	0.07
1	0	20	20	20	20	1.07
1	0	30	30	30	30	14.50
1	0	40	40	40	40	64.12
1	0	50	50	50	50	274.42

Tras hacer pruebas cambiando los parámetros e incrementando en un orden de 10, hemos podido observar que para que tenga un tiempo de ejecución aproximado de 10 segundos, los posibles parámetros son **-d 1 -r 0 -l 30-p 30 -c 30 -g 30**. Tarda un tiempo total de **14.50 segundos**. En el caso de disponer un tiempo máximo de 5 minutos, hemos podido comprobar que los parámetros ideales para que no sobrepase el tiempo son **-d 1 -r 0 -l 50 -p 50 -c 50 -g 50**. La duración total es de 274.42 segundos, es decir, **4.57 minutos**.

La **gráfica creada para ver mejor cómo crece exponencialmente el tiempo de ejecución en segundos** del planificador FF modificando los parámetros es la siguiente. En nuestro caso, tenemos en cuenta el parámetro de humanos del problema como referencia del tamaño del problema.



Ejercicio 1.3. Comparativa rendimiento planificadores

A continuación vamos a llevar a cabo un estudio de los diferentes planificadores presentados en la IPC 2004 e IPC 2011: FF, LPG-TD, SGPLAN40, SATPLAN y FastDownward.

	Año presentación	Categoría	Posición	Versión PDDL
FF	AIPS 2002	Strips Track	1º	PDDL 2.1
LPG-TD	IPC 2004	"Timed Initial Literals"	1º	PDDL 2.1, 2.2
	IPC 2004	Plan Quality	1º	
	IPC 2004	Suboptimal Temporal Metric Track	2º	
SGPLAN4	IPC 2004	Suboptimal Temporal Metric Track	1º	PDDL 2.2
		Suboptimal Propositional Track	2º	
SATPLAN	ICAPS 2004	Optimal deterministic planning	1º	PDDL2.1 Level 1 to 3
FastDownward	IPC 2004	Suboptimal Propositional Track	1º	PDDL 2.2 Level 1

FF (Fast Forward)

```
./ff -o drone-domain.pddl -f drone-problem.pddl
```

FF calcula una solución (plan relajado) al problema simplificado R sobre el grafo de planificación relajado. La longitud del plan relajado se usa como estimación de la dificultad del problema. Dicha estimación se usa posteriormente para controlar un proceso de búsqueda local similar al ascenso de colinas (hill-climbing).

FF combina la búsqueda local con una búsqueda en anchura para evitar los mínimos locales. También usa el plan relajado como guía para seleccionar las acciones durante la búsqueda en anchura con el objetivo de reducir el factor de ramificación.

LPG-TD (Local search for Planning Graphs)

```
./lpg-td -o drone-domain.pddl -f drone-problem.pddl -n 1
```

LPG es un planificador basado en búsqueda local y gráficos de planificación que maneja dominios PDDL2.1 que involucran cantidades y duraciones numéricas. El sistema puede resolver tanto problemas de generación de planes como de adaptación de planes. LPG-td es

una extensión de LPG para manejar las nuevas características de la planificación estándar lenguajes de descripción de dominio PDDL2.2.

La función de evaluación utiliza algunas heurísticas para estimar el "costo de búsqueda" y el "costo de ejecución" de lograr una condición previa (posiblemente numérica). Las duraciones de las acciones y las cantidades numéricas (por ejemplo, el consumo de combustible) se representan en los gráficos de acciones y se modelan en la función de evaluación. En los dominios temporales, las acciones se ordenan utilizando un "gráfico de precedencia" que se mantiene durante la búsqueda y que tiene en cuenta las relaciones mutex del gráfico de planificación.

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	0	10	10	10	10	0.00
1	0	20	20	20	20	0.07
1	0	30	30	30	30	0.53
1	0	40	40	40	40	2.52
1	0	50	50	50	50	10.20
1	0	80	80	80	80	68.09
1	0	90	90	90	90	90.16
1	0	100	100	100	100	443.87

Tras hacer pruebas cambiando los parámetros e incrementando en un orden de 10, hemos podido observar que para que tenga un tiempo de ejecución aproximado de 10 segundos, los posibles parámetros son **-d 1 -r 0 -l 50-p 50 -c 50 -g 50**. Tarda un tiempo total de **10.20 segundos**. En el caso de disponer un tiempo máximo de 5 minutos, hemos podido comprobar que los parámetros ideales nunca se aproximan lo suficiente a dicho tiempo. El que más se aproxima aunque **sobrepasa el tiempo** es **-d 1 -r 0 -l 100 -p 100 -c 100 -g 100**. La duración total es de 443.87 segundos, es decir, **7.39 minutos**. El anterior a él tiene una duración de 90.16 segundos, es decir, **1.50 minutos.**

Como podemos comprobar, este planificador encuentra un plan de una manera mucho más rápida que el Fast Forward. Esto se debe a que el **LPG es un planificador rápido** que utiliza la búsqueda local para resolver gráficos de planificación y puede usar varias heurísticas basadas en una función objetivo parametrizada. Estos parámetros ponderan diferentes tipos de inconsistencias en el plan parcial representado por el estado de búsqueda actual y se evalúan dinámicamente durante la búsqueda utilizando multiplicadores de Lagrange.

SGPLAN40

```
./sgplan40 -o drone-domain.pddl -f drone-problem.pddl
```

Divide un gran problema de planificación en subproblemas, cada uno con su propio subobjetivo, y resuelve soluciones inconsistentes de subobjetivos usando nuestra condición de punto de silla extendida.

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	0	10	10	10	10	0.00
1	0	20	20	20	20	0.06
1	0	30	30	30	30	0.24
1	0	40	40	40	40	1.64
1	0	50	50	50	50	3.00
1	0	60	60	60	60	7.20
1	0	70	70	70	70	29.08
1	0	80	80	80	80	40.52
1	0	90	90	90	90	91.70
1	0	100	100	100	100	105.10
1	0	120	120	120	120	309.92

Tras hacer pruebas cambiando los parámetros e incrementando en un orden de 10, hemos podido observar que para que tenga un tiempo de ejecución aproximado de 10 segundos, los posibles parámetros son **-d 1 -r 0 -l 60-p 60 -c 60 -g 60**. Tarda un tiempo total de **7.20 segundos**. El siguiente caso más cercano se aleja aún más de los 10 segundos propuestos, con una duración de 29.08 segundos. En el caso de disponer un tiempo máximo de 5 minutos, hemos podido comprobar que los parámetros ideales para que no sobrepase el tiempo son **-d 1 -r 0 -l 120 -p 120 -c 120 -g 120**. La duración total es de 309.92 segundos, es decir, **5.16 minutos**.

Los tiempos de ejecución son similares al del Fast Forward ya que el SGPlan40 r**esuelve los subproblemas individualmente usando el planificador Metric-FF modificado**. Sin embargo, al dividirlo en subproblemas, consigue obtener un tiempo mucho menor.

SATPLAN

```
./satplan -solver siege -domain drone-domain.pddl -problem
drone-problem.pddl
```

Se denominan planificadores SAT aquellos que realizan una traducción a SAT para resolver el problema de planificación. Dadas las características, estos planificadores suelen tener algunas limitaciones respecto a los tipos de problemas que pueden resolver, pero este esquema genera planificadores muy rápidos.

Encuentra soluciones con una longitud paralela mínima, es decir, muchas acciones (que no interfieren) pueden ocurrir en paralelo en cada paso de tiempo, y se garantiza que el número total de pasos de tiempo sea lo más pequeño posible.

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	0	2	2	2	2	0.04
1	0	3	3	3	3	0.08
1	0	4	4	4	4	0.18
1	0	5	5	5	5	2.00
1	0	6	6	6	6	11.36
1	0	7	7	7	7	258.87

Debido a los tiempos de ejecución tan elevados que proporciona este planificador haremos pruebas hasta llegar a 10. Hemos podido observar que para que tenga un tiempo de ejecución aproximado de 10 segundos, los posibles parámetros son **-d 1 -r 0 -l 6-p 6 -c 6 -g 6**, con una duración total de **11.36 segundos**. En el caso de disponer un tiempo máximo de 5 minutos, hemos podido comprobar que los parámetros ideales para que no sobrepase el tiempo son **-d 1 -r 0 -l 7 -p 7 -c 7 -g 7**. La duración total es de 258.87 segundos, es decir, **4.31 minutos**.

Los tiempos de ejecución aumentan mucho en comparación con otros planificadores. Esto se debe a que el planificador SATPLAN tiene un rendimiento muy bajo en este dominio y hay que hacer pruebas con problemas más pequeños respecto a los planificadores anteriores.

FastDownward (con opción – alias lamafirst)

```
./downward.sif --alias lama-first ./drone-domain.pddl
drone-problem.pddl
```

Fast Downward es un sistema de planificación clásica basado en la búsqueda heurística. Puede tratar problemas generales de planificación determinista codificados en el fragmento proposicional de PDDL2.2, incluidas funciones avanzadas como condiciones y efectos ADL y predicados derivados (axiomas). Al igual que otros planificadores conocidos como HSP y

FF, Fast Downward es un planificador de progresión que busca en el espacio de los estados una tarea de planificación en la dirección de avance.

Sin embargo, a diferencia de otros sistemas de planificación PDDL, éste no utiliza directamente la representación PDDL proposicional de una tarea de planificación. En su lugar, la entrada se traduce primero en una representación alternativa denominada "tareas de planificación multivaluadas", lo que hace explícitas muchas de las restricciones implícitas de una tarea de planificación proposicional. Aprovechando esta alternativa, en su año de lanzamiento (2004), la configuración Fast Downward empleó la búsqueda mejor primero multiheurística usando transiciones útiles y acciones útiles como operadores preferentes.

Esto hizo que fuera rápido diagonalmente.

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	0	10	10	10	10	0.013
1	0	20	20	20	20	0.05
1	0	30	30	30	30	0.21
1	0	40	40	40	40	0.57
1	0	50	50	50	50	0.67
1	0	60	60	60	60	1.53
1	0	70	70	70	70	2.83
1	0	80	80	80	80	7.56
1	0	90	90	90	90	6.60
1	0	100	100	100	100	12.57
1	0	120	120	120	120	99.01
1	0	150	150	150	150	73.40
1	0	180	180	180	180	164.79
1	0	190	190	190	190	370.50

Como podemos comprobar en la anterior tabla, el máximo problema que puede ejecutar sin que superen los 10 segundos será introduciendo los parámetros: **-d 1 -r 0 -l 90 -p 90 -c 90 -g 90**

Sin embargo, si el máximo es de 5 minutos, podríamos ejecutar los parámetros: **-d 1 -r 0 -l 180 -p 180 -c 180 -g 180** porque de ahí en adelante aumentan los segundos considerablemente.

Para concluir con este punto, si nos ceñimos a la experiencia de la ejecución de cada uno de los planificadores anteriores con los distintos problemas, el mejor planificador desde nuestro punto de vista es FastDownward y el que peor con respecto a tiempo en generar un plan, SATPLAN.

FastDownward ha sido capaz de generar planes en bastante poco tiempo en problemas que para los demás planificadores ni siquiera serían alcanzables, lo que le hace bastante rápido.

Obviamente que la diferencia está en el año de lanzamiento, ya que como bien sabemos estos planificadores son del 2002 al 2004, mientras que para FastDownward, existen planificadores publicados en el IPC del 2011 que hacen uso de FastDownward en su interior. Estos planificadores son:

Satisficing track:

- Fast Downward Autotune 1, satisficing version (seq-sat-fd-autotune-1)
- Fast Downward Autotune 2, satisficing version (seq-sat-fd-autotune-2)
- *Fast Downward Stone Soup 1, satisficing version (seq-sat-fdss-1)
- *Fast Downward Stone Soup 2, satisficing version (seq-sat-fdss-2)
- LAMA 2011 (seq-sat-lama-2011)

Optimization track:

- BJOLP (seq-opt-bjolp)
- Fast Downward Autotune, optimizing version (seq-opt-fd-autotune)
- Fast Downward Stone Soup 1, optimizing version (seq-opt-fdss-1)
- Fast Downward Stone Soup 2, optimizing version (seq-opt-fdss-2)
- LM-Cut (seq-opt-lmcut)
- Merge and Shrink (seq-opt-merge-and-shrink)
- Selective Max (seq-opt-selmax)

Estos planificadores si que son más lentos como veremos más adelante, pero en este caso hemos usado la opción lama-first.

La opción lama-first busca un plan lo más rápido posible sin tener en cuenta el coste y es por ello que los planes generados son en tan poco tiempo. Como todavía no hemos introducido ningún coste a las acciones, por el momento nos quedamos con que FastDownwar con la opción lama-first es el mejor planificador.

Parte 2: Planificación con números y optimización

Ejercicio 2.1. Servicio de emergencias, transportadores

1. Modificación de dominio

Ahora los requisitos son los siguientes:

```
(:requirements
    :strips :typing
)
```

Añadimos 4 nuevos predicados: carrier-at, carrier-has-box, next y carrier-n-boxes.

```
(:predicates
    (drone-at ?d - drone ?l - location) 6👁 2☑ 2☒
    (drone-free ?d - drone) 3👁 2☑ 2☒
    (drone-carry-box ?d - drone ?b - box) 2👁 2☑ 2☒
    (human-at ?h - human ?l - location) 1👁
    (human-has ?h - human ?c - content) 1☑
    (box-at ?b - box ?l - location) 1👁 1☑ 1☒
    (box-has ?b - box ?c - content) 2👁
    (box-free ?b - box) 1👁 1☒
    (carrier-at ?r - carrier ?l - location) 3👁 1☑ 1☒
    (carrier-has-box ?r ?b) 1👁 1☑ 1☒
    (next ?n1 ?n2 - num) 2👁
    (carrier-n-boxes ?r - carrier ?n - num) 2👁 2☑ 2☒
)
```

Añadimos al dominio 3 nuevas acciones: move-carrier, put-box-on-carrier, take-off-box-carrier.

```
(:action move-carrier
    :parameters (?d - drone ?to ?from - location ?r - carrier)
    :precondition (and
        (drone-free ?d)
        (drone-at ?d ?from)
        (carrier-at ?r ?from)
    )
    :effect (and
        (not(carrier-at ?r ?from)); el carrier deja de estar en la localización
        (not(drone-at ?d ?from))
        (carrier-at ?r ?to)
        (drone-at ?d ?to)
    )
)
```

```
(:action put-box-on-carrier
    :parameters (?d - drone ?b - box ?l - location ?r - carrier ?n1 ?n2 - num)
    :precondition (and
        (drone-at ?d ?l)
        (carrier-at ?r ?l) ;debe estar bajado el carrier
        (drone-carry-box ?d ?b) ;el dron tiene que tener la caja (haber hecho pick-up)
        (next ?n1 ?n2) ;no ha llegado al limite de capacidad
        (carrier-n-boxes ?r ?n1)
    )
    :effect (and
        (drone-free ?d)
        (not (drone-carry-box ?d ?b)) ;el dron deja de tener la caja
        (carrier-has-box ?r ?b) ;para tenerla el carrier
        (not (carrier-n-boxes ?r ?n1))
        (carrier-n-boxes ?r ?n2)
    )
)
```

```
(:action take-off-box-carrier
    :parameters (?d - drone ?b - box ?l - location ?r - carrier ?n1 ?n2 - num)
    :precondition (and
        (drone-at ?d ?l)
        (carrier-at ?r ?l) ;el carrier y el dron deben estar en el mismo sitio
        (carrier-has-box ?r ?b) ;la caja a coger la tiene que tener el carrier
        (drone-free ?d) ;el dron tiene que estar vacio
        (carrier-n-boxes ?r ?n2)
        (next ?n1 ?n2)
    )
    :effect (and
        (not(drone-free ?d))
        (drone-carry-box ?d ?b) ;el dron coge la caja, y
        (not(carrier-has-box ?r ?b)) ;el carrier ya no la tendrá
        (carrier-n-boxes ?r ?n1)
        (not(carrier-n-boxes ?r ?n2))
    )
)
```

2. **Adaptar el generador de problemas acorde al nuevo dominio.**

Para adaptar el generador al dominio nuevo, incluimos los siguientes cambios por cada apartado:

- **Define**: incluimos el contador para cada posición de las cajas y la definición de los carriers solicitados por el usuario.

```
for x in carrier:
    f.write("\t" + x + " - carrier\n")
```

```
f.write("\t" + 'n0 n1 n2 n3 n4' + " - num\n")
```

- **Init**: incluimos los carriers e iniciamos los valores para el contador n.

```
for x in carrier:
    f.write("\t(carrier-at " + x + " " + location[0] + ")\n")
    f.write("\t(carrier-n-boxes " + x + " " + 'n0' + ")\n")
```

```
    f.write('\t(next n0 n1)\n')
    f.write('\t(next n1 n2)\n')
    f.write('\t(next n2 n3)\n')
    f.write('\t(next n3 n4)\n')
```

- **Goals**: incluimos que los carriers

```
for x in carrier:
    # TODO: Write a goal that the drone x is at the warehouse
    f.write("\t(carrier-at " + x + " " + location[0] + ")\n")
```

Realizamos las diferentes pruebas generando nuevos problemas que incluyen 1 carrier que será transportado por el dron. Además, buscaremos un problema con el tamaño necesario cuya estimación de tiempo sea 2 minutos. Estas pruebas son mostradas en el apartado 4 de este ejercicio, en el apartado de FF.

3. **Verificar que los dominios y problemas generados se resuelven correctamente utilizando las nuevas acciones del controlador.**

Si comentamos el método move utilizado en la parte 1, podemos ver que forzamos al planificador a utilizar los métodos move-carrier, put-box-on-carrier, take-off-box-carrier, lo que quiere decir que el dron carga el transportador con una caja, después moverá el transportador a la localización necesitada, y después el dron bajará la caja y se la entregará al humano que la solicita.

Si no comentamos el move de la parte 1, el planificador no resolverá el problema de manera óptima (no cargará 4 cajas en el transportador), y ni siquiera hará uso de éste usando la acción move que únicamente mueve el dron de una ubicación a otra.

Ejecutando un problema de ejemplo podemos ver el siguiente fragmento teniendo la acción move comentada:

```
0.001: (PICK-UP DRONE1 BOX3 CLOTHES WAREHOUSE) [1]
1.002: (PUT-BOX-ON-CARRIER DRONE1 BOX3 WAREHOUSE CARRIER1 N0 N1) [1]
2.003: (MOVE-CARRIER DRONE1 LOC1 WAREHOUSE CARRIER1) [1]
3.004: (TAKE-OFF-BOX-CARRIER DRONE1 BOX3 LOC1 CARRIER1 N0 N1) [1]
4.005: (DROP DRONE1 BOX3 CLOTHES LOC1 HUMAN2) [1]
5.006: (MOVE-CARRIER DRONE1 WAREHOUSE LOC1 CARRIER1) [1]
```

Mientras que si descomentamos la acción el planificador realiza los siguientes pasos:

```
0.001: (PICK-UP DRONE1 BOX3 CLOTHES WAREHOUSE) [1]
1.002: (MOVE DRONE1 WAREHOUSE LOC1) [1]
2.003: (DROP DRONE1 BOX3 CLOTHES LOC1 HUMAN2) [1]
3.004: (MOVE DRONE1 LOC1 WAREHOUSE) [1]
```

Tras analizar estos movimientos, podemos decir que el planificador FF de normal va a optar por utilizar el método move, ya que no es un planificador óptimo y hará menos movimientos por no tener que hacer hacer el move-carrier.

4. **Ejecutar los problemas generados de la parte 1 de la práctica.**

FF (Fast Forward)

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	10	10	10	10	0.15
1	1	20	20	20	20	5.87
1	1	30	30	30	30	46.18
1	1	34	34	34	34	133.21
1	1	40	40	40	40	243.85

Los planes generados funcionan de forma que el planificador opta por seguir utilizando el dron para mover individualmente cada caja. Esto se debe a que el Fast Forward no es un planificador óptimo y lo ejecuta de esta manera para ahorrarse hacer el move-carrier. Los **tiempos de ejecución aumentan** respecto a los tiempos de ejecución obtenidos en la parte 1. Debido al aumento exponencial del tiempo, en esta ocasión solo vamos a realizar los 4 primeros, ya que el **-d 1 -r 1 -l 40-p 40 -c 40 -g 40** el tiempo es de 243.85 segundos, es decir, **4.06 minutos**, mientras que en la parte 1 el tiempo era de 64.12 segundos.

Como se nos pide en uno de los apartados anteriores (2), realizamos además la búsqueda de un problema que tenga una **duración aproximada de 2 minutos**. El que más se aproxima es **-d 1 -r 1 -l 34 -p 34 -c 34 -g 34,** con una duración de 133.21, es decir, 2.20 minutos.

LPG-TD

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	10	10	10	10	0.04
1	1	20	20	20	20	0.31
1	1	30	30	30	30	46.22
1	1	40	40	40	40	587.35

Los planes generados funcionan de forma que el planificador opta por seguir utilizando el dron para mover individualmente cada caja, sin embargo en algunas ocasiones mueve el transportador. Esto se debe a que el LPG-TD no es un planificador óptimo y lo ejecuta de esta manera para ahorrarse hacer el move-carrier. Los tiempos de ejecución aumentan respecto a los tiempos de ejecución obtenidos en la parte 1. Por ello, solo hemos realizado las 4 primeras pruebas, ya que el tiempo crece exponencialmente y con -d 1 -r 1 -l 40-p 40 -c 40 -g 40 el tiempo es de 587.35 segundos, es decir, 9.5 minutos, mientras que en la parte 1 el tiempo era de 2.52 segundos.

SGPLAN40

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	10	10	10	10	0.00
1	1	20	20	20	20	0.22
1	1	30	30	30	30	0.98
1	1	40	40	40	40	7.50
1	1	50	50	50	50	31.46
1	1	60	60	60	60	132.52
1	1	70	70	70	70	364.44

De nuevo el planificador opta por seguir utilizando el dron para mover individualmente cada caja. Los **tiempos de ejecución aumentan** respecto a los tiempos de ejecución obtenidos en la parte 1.

Por ello, hemos realizado las primeras 7 pruebas, ya que con **-d 1 -r 1 -l 70-p 70 -c 70 -g 70** el tiempo es de 364.44 segundos, es decir, **6.07 minutos**, mientras que en la parte 1 el tiempo era de 29.08 segundos.

SATPLAN

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	2	2	2	2	0.06
1	1	3	3	3	3	0.19
1	1	4	4	4	4	19.90
1	1	5	5	5	5	180.93

De nuevo el planificador opta por seguir utilizando el dron para mover individualmente cada caja. Los **tiempos de ejecución aumentan** respecto a los tiempos de ejecución obtenidos en la parte 1.

Por ello, hemos realizado las primeras 4 pruebas, ya que con **-d 1 -r 1 -l 5-p 5 -c 5 -g 5** el tiempo es de 180.93 segundos, es decir, **3.01 minutos**, mientras que en la parte 1 el tiempo era de 29.08 segundos.

FastDownward

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	0	10	10	10	10	0.03
1	0	20	20	20	20	0.19
1	0	30	30	30	30	0.68
1	0	40	40	40	40	2.29
1	0	50	50	50	50	10.54
1	0	60	60	60	60	18.49
1	0	70	70	70	70	29.93

Para que el FastDownward funcione, hemos **eliminado temporalmente** del dominio el requerimiento ‘:fluents’ debido a que no lo soporta. A continuación podemos observar el error que nos sale por la terminal:

```
pachben1@ubuntu:~/Descargas/singularity-ce-3.9.5$ ./downward.sif --alias lama-first
./drone-domain.pddl drone_problem_d1_r1_l10_p10_c10_g10_ct2.pddl
INFO     Running translator.
INFO     translator stdin: None
INFO     translator time limit: None
INFO     translator memory limit: None
INFO     translator command line string: /usr/bin/python3 /workspace/downward/builds
/release/bin/translate/translate.py ./drone-domain.pddl drone_problem_d1_r1_l10_p10_
c10_g10_ct2.pddl --sas-file output.sas
Parsing...
b'Traceback (most recent call last):\n  File "/workspace/downward/builds/release/bin
/translate/translate.py", line 727, in <module>\n    main()\n  File "/workspace/down
ward/builds/release/bin/translate/translate.py", line 686, in main\n    domain_filen
ame=options.domain, task_filename=options.task)\n  File "/workspace/downward/builds/
release/bin/translate/pddl_parser/pddl_file.py", line 33, in open\n    return parsin
g_functions.parse_task(domain_pddl, task_pddl)\n  File "/workspace/downward/builds/r
elease/bin/translate/pddl_parser/parsing_functions.py", line 297, in parse_task\n
 = parse_domain_pddl(domain_pddl)\n  File "/workspace/downward/builds/release/bin/tr
anslate/pddl_parser/parsing_functions.py", line 349, in parse_domain_pddl\n    requi
rements = pddl.Requirements(opt[1:])\n  File "/workspace/downward/builds/release/bin
/translate/pddl/tasks.py", line 68, in __init__\n    ":derived-predicates", ":action
-costs"), req\nAssertionError: :fluents\n'
translate exit code: 30
```

Tras modificarlo, ya hemos podido ejecutar correctamente el dominio y el problema.

Los tiempos no son considerablemente altos con FastDownward, pero si es cierto que aumentan con respecto al anterior dominio y problema con los mismos parámetros. Por otro lado, es destacable que no use la acción move-carrier:

```
move drone1 loc69 loc6 (1)
move drone1 loc6 warehouse (1)
pick-up drone1 box8 food warehouse (1)
move drone1 warehouse loc6 (1)
drop drone1 box8 food loc6 human29 (1)
move drone1 loc6 loc60 (1)
move drone1 loc60 warehouse (1)
pick-up drone1 box9 food warehouse (1)
move drone1 warehouse loc60 (1)
drop drone1 box9 food loc60 human50 (1)
move drone1 loc60 warehouse (1)
```

Hablando en términos generales para este punto, **todos los planificadores** tardan más con los mismos parámetros con respecto al anterior punto (1.3) y **no hacen uso de move-carrier** en ningún caso (excepto LPG-TD en algún caso). Esto último se debe a que al no haber costes en las acciones los planificadores no buscan la solución más óptima si no alguna que pueda ejecutarse.

El planificador más rápido es FastDownward con bastante diferencia como en el punto 1.3, donde también era claramente superior a los demás planes.

Ejercicio 2.2. Servicio de emergencias, costes de acción

1. Modificación del dominio

En el apartado de requisitos, añadimos :action-costs:

```
(:requirements
    :strips :fluents :typing :action-costs
)
```

Creamos un nuevo apartado de functions, donde añadimos total-cost y fly-cost:

```
(:functions
    (total-cost) 6↗
    (fly-cost ?origen ?destino - location) 2👁
)
```

Modificamos cada una de las acciones del dominio. En las acciones que incluyen movimiento, el total-cost es el coste del vuelo de una localización a otra que se declara en el dominio. El coste de estos es un aleatorio que sacamos con la función random en el generador de problemas. El resto de acciones tendrán un incremento de 1 al coste total:

```
(:action move-carrier
    :parameters (?d - drone ?to ?from - location ?r - carrier)
    :precondition (and
        (drone-free ?d)
        (drone-at ?d ?from)
        (carrier-at ?r ?from)
    )
    :effect (and
        (not(carrier-at ?r ?from)); el carrier deja de estar en la localización
        (not(drone-at ?d ?from))
        (carrier-at ?r ?to)
        (drone-at ?d ?to)
        (increase (total-cost) (fly-cost ?from ?to))
    )
)
```

```
(:action move
    :parameters (?d - drone ?from ?to - location)
    :precondition (and
        (drone-at ?d ?from)
    )
    :effect (and
        (drone-at ?d ?to)
        (not (drone-at ?d ?from))
        (increase (total-cost) (fly-cost ?from ?to))
    )
)
```

```
(:action put-box-on-carrier
    :parameters (?d - drone ?b - box ?l - location ?r - carrier ?n1 ?n2 - num)
    :precondition (and
        (drone-at ?d ?l)
        (carrier-at ?r ?l) ;debe estar bajado el carrier
        (drone-carry-box ?d ?b) ;el dron tiene que tener la caja (haber hecho pick-up)
        (next ?n1 ?n2) ;no ha llegado al limite de capacidad
        (carrier-n-boxes ?r ?n1)
    )
    :effect (and
        (drone-free ?d)
        (not (drone-carry-box ?d ?b)) ;el dron deja de tener la caja
        (carrier-has-box ?r ?b) ;para tenerla el carrier
        (not (carrier-n-boxes ?r ?n1))
        (carrier-n-boxes ?r ?n2)
        (increase (total-cost) 1)
    )
)
```

```
(:action pick-up
    :parameters (?d - drone ?b - box ?c - content ?l - location)
    :precondition (and
        (drone-at ?d ?l)
        (box-at ?b ?l)
        (drone-free ?d)
        (box-has ?b ?c)
        (box-free ?b)
    )
    :effect (and
        (drone-carry-box ?d ?b)
        (not (drone-free ?d))
        (not (box-at ?b ?l))
        (increase (total-cost) 1)
    )
)
```

```
(:action take-off-box-carrier
    :parameters (?d - drone ?b - box ?l - location ?r - carrier ?n1 ?n2 - num)
    :precondition (and
        (drone-at ?d ?l)
        (carrier-at ?r ?l) ;el carrier y el dron deben estar en el mismo sitio
        (carrier-has-box ?r ?b) ;la caja a coger la tiene que tener el carrier
        (drone-free ?d) ;el dron tiene que estar vacio
        (carrier-n-boxes ?r ?n2)
        (next ?n1 ?n2)
    )
    :effect (and
        (not(drone-free ?d))
        (drone-carry-box ?d ?b) ;el dron coge la caja, y
        (not(carrier-has-box ?r ?b)) ;el carrier ya no la tendrá
        (carrier-n-boxes ?r ?n1)
        (not(carrier-n-boxes ?r ?n2))
        (increase (total-cost) 1)
    )
)
```

```
(:action drop
    :parameters (?d - drone ?b - box ?c - content ?l - location ?h - human)
    :precondition (and
        (drone-carry-box ?d ?b)
        (drone-at ?d ?l)
        (human-at ?h ?l)
        (box-has ?b ?c)
    )
    :effect (and
        (drone-free ?d)
        (box-at ?b ?l)
        (human-has ?h ?c)
        (not (drone-carry-box ?d ?b))
        (not (box-free ?b))
        (increase (total-cost) 1)
    )
)
```

Si realizamos pruebas con el planificador Optic-CLP de la página https://pddl-editor.herokuapp.com/editor con un TimeOut de 10 segundos, podemos ver que, si eliminamos la acción move del dominio, lo hace de manera correcta, moviendo varias cajas al carrier:

```
; Plan found with metric 62.000
; Theoretical reachable cost 62.000
; States evaluated so far: 9332
; States pruned based on pre-heuristic cost lower bound: 0
; Time 7.14
0.000: (pick-up drone1 box1 meds warehouse) [0.001]
0.001: (put-box-on-carrier drone1 box1 warehouse carrier1 n0 n1) [0.001]
0.002: (pick-up drone1 box2 water warehouse) [0.001]
0.003: (put-box-on-carrier drone1 box2 warehouse carrier1 n1 n2) [0.001]
0.004: (pick-up drone1 box3 clothes warehouse) [0.001]
0.005: (put-box-on-carrier drone1 box3 warehouse carrier1 n2 n3) [0.001]
0.006: (move-carrier drone1 loc1 warehouse carrier1) [0.001]
0.007: (take-off-box-carrier drone1 box2 loc1 carrier1 n2 n3) [0.001]
0.008: (drop drone1 box2 water loc1 human1) [0.001]
0.009: (take-off-box-carrier drone1 box1 loc1 carrier1 n1 n2) [0.001]
0.010: (drop drone1 box1 meds loc1 human1) [0.001]
0.011: (take-off-box-carrier drone1 box3 loc1 carrier1 n0 n1) [0.001]
0.012: (drop drone1 box3 clothes loc1 human2) [0.001]
0.013: (move-carrier drone1 loc2 loc1 carrier1) [0.001]
0.014: (move-carrier drone1 warehouse loc2 carrier1) [0.001]
"
```

2. Modificación del generador de problemas

Para el nuevo problema hay que añadir 2 cosas nuevas.

En el init los valores de fly-cost de volar de una ubicación a otra, teniendo en cuenta de dónde a dónde estamos viajando (más adelante lo veremos) e inicializar el total-cost a 0 que, es el valor del coste total del plan generado.

Debemos añadir al final del problema siempre la siguiente línea de código:

```
(:metric minimize (total-cost))
```

Esta línea buscará el coste mínimo de entre todos los planes.

En el generador de problemas (*.py*) hemos declarado en el init los valores de coste de vuelo de una ubicación a otra que, como hemos dicho antes, lo generamos con un random en caso de que sea de un lugar a otro, y un 0 en caso de ser de la ubicación a la misma ubicación:

```
for i in range(len(location)):
    for j in range(len(location)):
        if (i == j):
            f.write("\t(= (fly-cost " + location[i] + " " + location[j] + ") 0)\n")
        else:
            rand = random.randint(1, 50)
            f.write("\t(= (fly-cost " + location[i] + " " + location[j] + ") " + str(rand) + ")\n")

f.write('\t(= (total-cost) 0)\n')
```

También se ha inicializado el coste total del plan a 0.

Por último, hemos añadido la última línea perteneciente al metric:

```
f.write("(:metric minimize (total-cost))")
```

3. Planificadores Metric-FF, LAMA, BJLOP, LM-Cut y FDSS2.

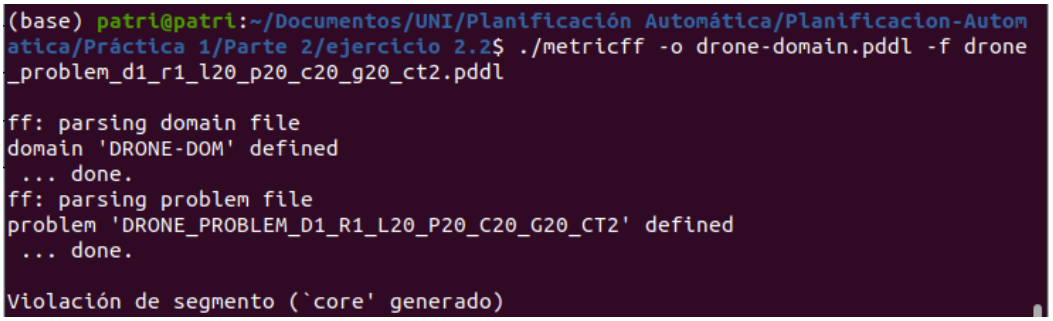
De nuevo, para los planificadores de FastDownward, hemos tenido que quitar ':fluents' de los requerimientos del dominio.

Metric-FF

```
./metricff -o drone-domain.pddl -f drone-problem.pddl
```

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	1	1	1	1	0.00
1	1	2	2	2	2	0.00
1	1	3	3	3	3	0.00
1	1	4	4	4	4	0.00
1	1	5	5	5	5	0.00
1	1	6	6	6	6	0.02
1	1	7	7	7	7	0.08
1	1	10	10	10	10	0.11

Utilizando este planificador, cuando se intentan añadir más de 10 localizaciones, 10 personas, 10 cajas y 10 goals, sale el siguiente error:



Para los siguientes, vamos a tener que tener bastante paciencia porque estos intentan buscar una solución óptima, y no dejan de buscar planes por lo cuál hemos probado desde valores pequeños para ir viendo como evolucionan.

LAMA

- Opción LAMA:

```
./downward.sif --alias lama ./drone-domain.pddl
drone-problem.pddl
```

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	2	2	2	2	0.02
1	1	3	3	3	3	0.07
1	1	4	4	4	4	0.56
1	1	5	5	5	5	5.91
1	1	6	6	6	6	128.69
1	1	7	7	7	7	1259.09

- Opción LAMA-first:

```
./downward.sif --alias lama-first ./drone-domain.pddl
drone-problem.pddl
```

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	2	2	2	2	0.01
1	1	3	3	3	3	0.01
1	1	4	4	4	4	0.01
1	1	5	5	5	5	0.02
1	1	10	10	10	10	0.05
1	1	20	20	20	20	0.21
1	1	50	50	50	50	6.13

BJOLP

```
./downward.sif --alias seq-opt.bjolp ./drone-domain.pddl
drone-problem.pddl
```

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	2	2	2	2	0.02
1	1	3	3	3	3	0.02
1	1	4	4	4	4	0.09
1	1	5	5	5	5	1.67
1	1	6	6	6	6	30.56
1	1	7	7	7	7	>2000.00

En la última ejecución no ha sido capaz de encontrar plan, sacando por pantalla:

“search exit code: -9”

LM-Cut

```
./downward.sif --alias seq-opt-lmcut ./drone-domain.pddl
drone-problem.pddl
```

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	2	2	2	2	0.01
1	1	3	3	3	3	0.03
1	1	4	4	4	4	0.21
1	1	5	5	5	5	6.86
1	1	6	6	6	6	150.88
1	1	7	7	7	7	>7200

FDSS2

```
./downward.sif --overall-time-limit 100 (poner el que se
quiera) --alias seq-opt-fdss-2 ./drone-domain.pddl
drone_problem2.pddl
```

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
1	1	5	5	5	5	0.41
1	1	6	6	6	6	5.95
1	1	7	7	7	7	5.27
1	1	8	8	8	8	8.73
1	1	10	10	10	10	??

En la última ejecución, excede la memoria en el plan y no es capaz de resolver el problema. Sacando por pantalla el siguiente error:

"search exit code: 23"

a. **¿Qué opciones puedo usar en Metric-FF para buscar soluciones óptimas? ¿Cómo afecta el parámetro -w a la optimalidad de la solución? ¿Qué representa dicho parámetro en el algoritmo Weigthed-A*?**

Las soluciones óptimas son las que consideramos que usen la minimización de costes, como son la opción de Weighted A*, A*epsilon o EHC+H y luego A*epsilon (se introducen con la opción -s)

Con el parámetro -w lo que estamos haciendo es introducir un peso para la búsqueda de las configuraciones anteriormente dichas.

WEIGTHED A* en función del parámetro -w:

```
./metricff -o drone-domain.pddl -f drone-problem.pddl
```

-d	-r	-l	-p	-c	-g	-w	Tiempo de ejecución
1	1	2	2	2	2	3	0.00
1	1	2	2	2	2	4	0.00
1	1	2	2	2	2	5	0.00
1	1	3	3	3	3	3	0.13
1	1	3	3	3	3	4	0.18
1	1	3	3	3	3	5	0.18
1	1	4	4	4	4	3	3.66
1	1	4	4	4	4	4	3.81
1	1	4	4	4	4	5	5.09
1	1	5	5	5	5	3	0.17
1	1	5	5	5	5	4	0.18
1	1	5	5	5	5	5	0.18

De forma general, cuanto mayor es el valor de la w, más tiempo tarda, ya que este valor representa el coste de minimización, es decir, la importancia que se le da a encontrar un plan de coste mínimo.

b. ¿Qué tipo de planificador es LAMA? ¿Qué ocurre cuando llamamos a downward.sif con la opción --alias lama-first? ¿Y con la opción –alias lama?

El proceso de búsqueda en LAMA está implementado por dos algoritmos.

En el primer algoritmo, la búsqueda “primero el mejor” expande un solo estado en cada iteración. El estado en cuestión es el que tiene el menor valor heurístico ‘h’ de todos los nodos abiertos.

El segundo algoritmo implementa la búsqueda A* ponderada, la cual asocia los costes con los estados y expande el estado con menor valor f.

LAMA usa dos funciones heurísticas: Landmarks y una variación de FF.

Como podemos observar en las tablas comparativas que hemos hecho anteriormente en el documento, la opción lama-first es mucho más rápida que la opción lama.

Si en ambos ejecutamos con los parámetros en el problema: -d 1 -r 1 -l 5 -p 5 -c 5 -g 5 LAMA ni siquiera es capaz de encontrar plan:

```
[t=0.622803s, 36296 KB] Completely explored state space -- no solution!
```

Sin embargo, lama-first sí que es capaz de encontrar un plan. Este plan no usa el carrier (es decir, no hace uso de move-carrier) y genera un plan de coste 253, con 21 pasos:

```
[t=0.0214701s, 34512 KB] Plan length: 21 step(s).
[t=0.0214701s, 34512 KB] Plan cost: 253
```

c. **¿Qué tipo de planificador es BJOLP? ¿Qué tamaño de problema es capaz de resolver en menos de 5 minutos? ¿Cómo se comporta respecto a LAMA en este problema? ¿Qué planificador sería más adecuado utilizar en una situación como la planteada en esta práctica?**

Hace uso de puntos de referencia (Landmarks) para derivar una heurística admisible, que luego se utiliza para guiar la búsqueda de un plan rentable.

El tamaño de problema que puede resolver en menos de 5 minutos es con los siguientes parámetros: **-d 1 -r 1 -l 6 -p 6 -c 6 -g 6.** Ya que a partir de ahí los resultados de tiempo son descabellados. Este plan generado tiene 34 pasos y un coste de 72.

Por la experiencia tomada con ambos, es preferible usar BJOLP, primeramente porque es capaz de resolver el problema. Aparte, el coste para el problema no es excesivamente alto comparado con otros. La última razón y la más importante es que este problema está diseñado para que haga uso de move-carrier, y este planificador (BJOLP) hace uso de esta acción a causa de los costes de las acciones.

d. **¿Qué planificador óptimo es mejor: BJOLP o LM-Cut? Proporciona datos y resultados de pruebas que justifiquen tu respuesta.**

Según las pruebas que hemos podido realizar es difícil quedarse con uno de estos a causa de los tiempos de ejecución cuando los problemas van siendo más grandes.

El número de pasos y el coste de los planes para los parámetros ejecutados son exactamente iguales, además que hacen uso de la acción move-carrier (que es uno de los objetivos).

Por lo cuál, si tenemos que decidirnos entre uno de estos, diríamos que BJOLP es el óptimo por los tiempos de ejecución de los problemas.

e. **¿Qué tipo de planificador es FDSS? ¿En qué consiste un planificador de tipo "portfolio"? ¿Cuánto tarda en resolver el problema generado en el apartado c, en el que BJOLP tardaba 5 minutos aproximadamente?**

Es un planificador de tipo portfolio secuencial que utiliza varias heurísticas y algoritmos de búsqueda que se han implementado en el sistema de planificación Fast Downward. Fast Downward Stone Soup (FDSS) muestra una prueba de concepto de cómo mezclar planificadores para crear portfolios.

Los planificadores portfolio son aquellos que hacen uso de varios planificadores en paralelo para sacar el máximo rendimiento de cada uno. Como bien ya sabemos, cada planificador tiene sus pros y sus contras, por lo cual, si cogemos varios, aprovecharemos lo mejor de cada uno.

Para los parámetros seleccionados en el planificador BJOLP: **-d 1 -r 1 -l 6 -p 6 -c 6 -g 6,** mientras que éste tarda 30.56 segundos, FDSS-2 es capaz de resolverlo en tan solo 5.95 segundos. El plan generado tiene 34 pasos y un coste de 72 (igual que con BJOLP).

Parte 3: Planificación con concurrencia

Ejercicio 3.1. Acciones concurrentes en gestión de emergencias

A continuación, describimos las acciones que se pueden realizar en paralelo:

- Dos drones pueden moverse, coger cajas y soltarlas simultáneamente.
- Dos drones pueden cargar cajas a un mismo transportador.
- Dos drones pueden moverse a una localización al mismo tiempo.
- Dos drones pueden mover cada uno un transportador a una misma localización simultáneamente.

A continuación, describimos las acciones que no se pueden realizar en paralelo:

- Dos drones no pueden coger una caja al mismo tiempo.
- No se pueden dar dos cajas a un humano al mismo tiempo.
- Un dron no puede mover un transportador si el otro dron está cargando o descargando cajas a ese transportador.
- Un dron no puede mover varios transportadores simultáneamente.

Ejercicio 3.2. Implementación y pruebas de concurrencia

- **Modificación del dominio.**

Modificamos los requisitos, eliminamos :action-costs y añadimos :durative-actions.

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

Añadimos dos nuevos predicados:

- **(human free ?h - human):** para asegurarse de que el humano solo puede recibir una caja a la vez.
- **(carrier-drone-free ?r - carrier):** para asegurarse de que un dron no mueva el carrier si el otro lo está cargando.

Las funciones vuelven a cambiar y únicamente se mantiene fly-cost, ya que eliminaremos los costes y total-cost ya no nos hace falta.

Todas las acciones pasan a ser :durative-actions, tendrán una duración fija en los casos de coger y soltar una caja y en las acciones de mover dependerá del fly-cost. Ya no hay precondiciones, y las condiciones ahora se dividen en at start, over all y at end.

- **Modificación del generador de problemas**

Ya que hemos quitado los costes, debemos modificar en los problemas el metric_minimize y eliminar la declaración de total-cost en el init.

```
(:metric minimize (total-time)))
```

Pruebas del dominio con OPTIC de la página https://pddl-editor.herokuapp.com/editor

Haciendo uso de dos drones, dos carriers, y el goals son 3 humanos que se encuentran en dos localizaciones diferentes. Podemos ver que funciona correctamente, pero en ocasiones se hace un move-carrier sin cargarlo, que se convierten en movimientos innecesarios.

```
; Plan found with metric 70.002
; Theoretical reachable cost 70.003
; States evaluated so far: 42
; States pruned based on pre-heuristic cost lower bound: 0
; Time 0.10
0.000: (move drone2 loc1 warehouse) [5.000]
0.000: (pick-up drone1 box1 water warehouse) [1.000]
1.000: (move drone1 warehouse loc1) [5.000]
5.001: (pick-up drone2 box2 water warehouse) [1.000]
6.000: (drop drone1 box1 water loc1 human2) [1.000]
6.001: (move drone2 warehouse loc1) [5.000]
7.000: (move drone1 loc1 warehouse) [5.000]
11.001: (drop drone2 box2 water loc1 human1) [1.000]
12.000: (pick-up drone1 box3 water warehouse) [1.000]
12.001: (move drone2 loc1 warehouse) [5.000]
13.000: (move drone1 warehouse loc1) [5.000]
17.002: (move-carrier drone2 loc1 warehouse carrier1) [5.000]
18.001: (move drone1 loc1 loc2) [23.000]
22.003: (move drone2 loc1 warehouse) [5.000]
41.001: (drop drone1 box3 water loc2 human3) [1.000]
42.001: (move drone1 loc2 loc1) [23.000]
65.002: (move-carrier drone1 warehouse loc1 carrier1) [5.000]
```

Si probamos otro código que tiene 2 drones, un carrier y dos humanos en una misma localización (ya que de otra manera supera el timeot en la página), comentando la acción move, realiza el siguiente plan:

```
; Plan found with metric 24.009
; Theoretical reachable cost 24.010
; States evaluated so far: 23
; States pruned based on pre-heuristic cost lower bound: 0
; Time 0.04
0.000: (pick-up drone1 box1 water warehouse) [1.000]
1.001: (put-box-on-carrier drone1 box1 warehouse carrier1 n0 n1) [1.000]
2.002: (move-carrier drone1 loc1 warehouse carrier1) [5.000]
7.003: (take-off-box-carrier drone2 box1 loc1 carrier1 n0 n1) [1.000]
7.004: (move-carrier drone1 warehouse loc1 carrier1) [5.000]
8.004: (drop drone2 box1 water loc1 human2) [1.000]
12.005: (pick-up drone1 box2 water warehouse) [1.000]
13.006: (put-box-on-carrier drone1 box2 warehouse carrier1 n0 n1) [1.000]
14.007: (move-carrier drone1 loc1 warehouse carrier1) [5.000]
19.008: (take-off-box-carrier drone2 box2 loc1 carrier1 n0 n1) [1.000]
19.009: (move-carrier drone1 warehouse loc1 carrier1) [5.000]
20.009: (drop drone2 box2 water loc1 human1) [1.000]

* All goal deadlines now no later than 24.009
```

- **Investiga qué tamaño de problema es capaz de resolver el planificador OPTIC en un minuto de tiempo aproximadamente, y cómo afecta al rendimiento del planificador la variación de distintos parámetros como el número de drones o el número de cajas.**

Realizaremos pruebas para dos dominios diferentes. El primero de ellos con el **dominio completo** y el segundo con la **acción move comentada.**

Dominio completo

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
2	2	2	2	2	2	0.02
4	4	2	2	2	2	19.00
2	2	4	4	4	4	1.86
4	4	4	4	4	4	-
4	2	4	4	4	4	91.00
2	4	4	4	4	4	8.34
2	2	6	6	6	6	3.22
4	2	6	6	6	6	92.44
4	2	6	6	20	6	293.48
2	4	6	6	6	6	90.06
2	2	8	8	8	8	24.30
2	2	10	10	10	10	47.26
2	2	12	12	12	12	254.88

A partir del uso de 4 drones y 4 carriers junto con 4 localizaciones, 4 personas y 4 cajas excede el uso de memoria y no es capaz de generar ningún plan.

Al hacer pruebas, podemos comprobar que si **aumentamos el número de cajas**, el tiempo que tardará en encontrar un plan aumenta considerablemente, ya que el planificador tiene que considerar todos los costes de moverse entre localizaciones para minimizar la duración. Como se puede ver en la siguiente imagen, los drones se mueven entre localizaciones para tardar menos.

```
0.000: (move drone1 warehouse loc5)  [18.000]
0.000: (move drone2 warehouse loc5)  [18.000]
0.000: (move drone3 warehouse loc5)  [18.000]
0.000: (move-carrier drone4 loc1 warehouse carrier1)  [22.000]
18.001: (move drone3 loc5 loc1)  [2.000]
18.001: (move drone2 loc5 loc1)  [2.000]
18.001: (move drone1 loc5 loc1)  [2.000]
20.002: (move drone3 loc1 warehouse)  [22.000]
20.002: (move drone2 loc1 warehouse)  [22.000]
20.002: (move drone1 loc1 loc6)  [11.000]
22.001: (move-carrier drone4 loc5 loc1 carrier1)  [2.000]
24.002: (move drone4 loc5 loc1)  [2.000]
26.003: (move drone4 loc1 loc6)  [11.000]
31.003: (move drone1 loc6 loc5)  [4.000]
35.004: (move-carrier drone1 warehouse loc5 carrier1)  [18.000]
37.004: (move drone4 loc6 loc5)  [4.000]
41.005: (move drone4 loc5 loc1)  [2.000]
42.002: (pick-up drone3 box1 food warehouse)  [1.000]
```

Al **aumentar el número de drones**, el planificador tarda más en encontrar un plan. Pero en el caso de introducir menos carriers que drones, será capaz de encontrar un plan antes de que la memoria exceda.

Como se puede ver marcado en la tabla, el tamaño de problema que puede resolver en menos de un minuto es con los siguientes parámetros: **-d 2 -r 2 -l 10 -p 10 -c 10 -g 10.**

Move comentado

Al comentar la acción move, conseguiremos forzar que se produzcan las acciones *move-carrier, take-off-box-carrier, put-box-on-carrier.* Sin embargo, en la mayoría de los casos, no se cargará más de una caja en el carrier y realizará viajes unitarios con el carrier, algo que es normal en el planificador OPTIC, ya que no es un planificador óptimo.

-d	-r	-l	-p	-c	-g	Tiempo de ejecución
2	2	2	2	2	2	0.08
4	4	2	2	2	2	1.36
4	2	4	4	4	4	0.50
2	2	4	4	4	4	unsolvable
4	4	4	4	4	4	281.35
2	2	6	6	6	6	3.18
4	2	6	6	6	6	66.04
4	2	6	6	20	6	unsolvable
2	2	8	8	8	8	unsolvable
2	2	10	10	10	10	unsolvable
2	2	12	12	12	12	unsolvable

Como se puede ver marcado en la tabla, en el caso de tener el move comentado, el tamaño de problema que puede resolver en aproximadamente un minuto es con los siguientes parámetros: **-d 4 -r 2 -l 6 -p 6 -c 6 -g 6.**

En este caso, muchas de las pruebas no han devuelto plan y se quedan como *problem unsolvable,* como se puede ver en la siguiente imagen.