Planificación Automática

PECL2

Curso 2021-2022

Isabel Blanco Martínez

Mario Pacheco Benito

Patricia Cuesta Ruiz

ÍNDICE

Parte 1: Planificación jerárquica con SHOP y Pyhop

Ejercicio 1.1. Logística de emergencias en SHOP2

El desarrollo en SHOP2 se lleva a cabo de la siguiente manera:

Desarrollamos un dominio nuevo para SHOP2 y un generador nuevo con los respectivos cambios nuevos en el problema. Para ello, creamos el dominio *dronedom* que está compuesto por los siguientes operadores:

- **move:** este operador se encarga de mover el drone de una localización a otra.

```
( defdomain dronedom (

    (:operator (!move ?d ?f ?t)

        ((DRONE ?d) (LOCATION ?f) (LOCATION ?t) (drone-at ?d ?f))
        ((drone-at ?d ?f)) ; delete
        ((drone-at ?d ?t)) ; add

    )

    (:operator (!pick-up ?d ?b ?c ?l)
```

- **pick-up:** este operador se encarga de coger una caja tras hacer algunas comprobaciones, como que el drone y la caja estén en el mismo sitio, que el drone y la caja estén vacías, etc.

```
(:operator (!pick-up ?d ?b ?c ?l)

    ( ; preconditions
        (DRONE ?d) (BOX ?b) (CONTENT ?c) (LOCATION ?l)
        (drone-at ?d ?l)
        (box-at ?b ?l)
        (drone-free ?d)
        (box-has ?b ?c)
        (box-free ?b)
    )
    ( ; delete
        (drone-free ?d)
        (box-at ?b ?l)
    )
    ( ; add
        (drone-carry ?d ?b)
    )
)
```

- **drop:** este operador se encarga de soltar una caja que lleva el drone, tras realizar ciertas comprobaciones, como que el humano y el drone están en la misma localización y que la caja tenga ese contenido.

```
(:operator (!drop ?d ?b ?c ?l ?h)

    ( ; preconditions
        (DRONE ?d) (BOX ?b) (CONTENT ?c) (LOCATION ?l) (HUMAN ?h)
        (drone-at ?d ?l)
        (drone-carry ?d ?b)
        (human-at ?h ?l)
        (box-has ?b ?c)
    )
    ( ; delete
        (drone-carry ?d ?b)
        (box-free ?b)
        (human-needs ?h ?c)
    )
    ( ; add
        (drone-free ?d)
        (box-at ?b ?l)
    )

)
```

También encontramos los dos siguientes métodos:

- **send-box:** encargado de mandar una a una cada una de las cajas, mediante los operadores pick-up, move y drop. Por último, siempre hacemos que el dron vuelva al almacén al terminar de enviar la caja.

```
(:method (send-box ?h ?c)

    ( ; preconditions
        (box-at ?b ?from)
        (human-at ?h ?to)
        (drone-free ?d)
        (box-free ?b)
        (drone-at ?d ?from)
        (box-has ?b ?c)
    )
    ( ; subtasks
        (!pick-up ?d ?b ?c ?from)
        (!move ?d ?from ?to)
        (!drop ?d ?b ?c ?to ?h)
        (!move ?d ?to ?from)
    )

)
```

- **send-all-boxes:** método recursivo que se encarga de comprobar si queda algún humano con alguna necesidad y llama al método send-box para mandar esa caja.

```
(:method (send-all-boxes) ; enviar-todo

    ( ; preconditions
       (human-needs ?h ?c)
    )
    ( ; subtasks
       (send-box ?h ?c) (send-all-boxes)
    )

    () ; preconditions
    () ; efect

)
```

Uno de los principales cambios que hemos hecho respecto a la primera práctica es que ahora tenemos *human-needs* para saber si un humano sigue teniendo alguna necesidad.

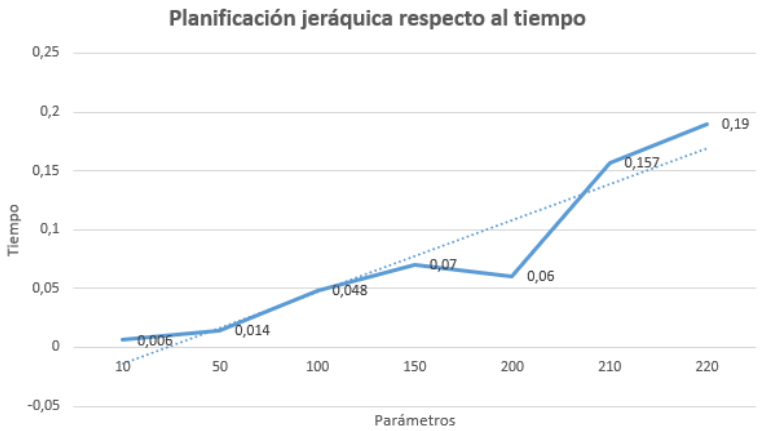
Creamos también el nuevo archivo de problema, que llamamos *problem.* El principal cambio es en los goals que ya no aparecen, sino que tenemos la siguiente estructura:

```
( ; TASKS
    (send-all-boxes) ; enviar-todo
)
```

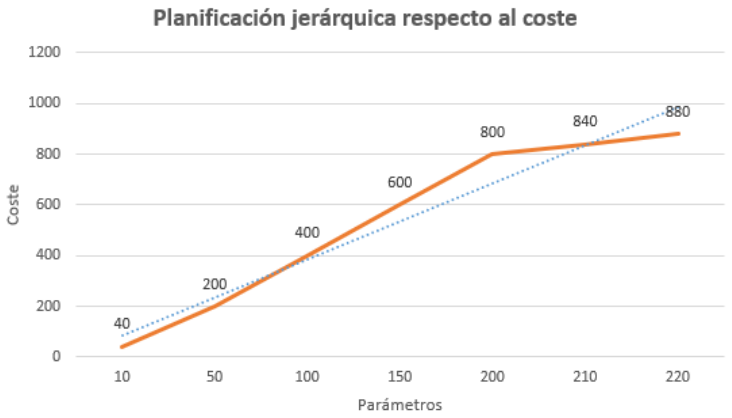
Se plantean las siguientes preguntas:

1. **Explica cómo crece el tiempo que tarda en generar una solución según el tamaño del problema, elaborando una gráfica.**

El tiempo va creciendo poco a poco según vamos aumentando el número de parámetros en nuestro problema, como podemos observar en la tabla del apartado número 2. En la siguiente gráfica, vamos a poder también observar que el aumento de tiempo es aproximadamente lineal:



También vamos a representar el coste respecto al número de parámetros en la siguiente gráfica:



2. **Compáralo los resultados anteriores con los del planificador que mejor rendimiento mostró en el ejercicio 1.3 de la práctica 1.**

El planificador que mejores rendimiento mostró en la práctica anterior fue el Fast-Downward con la opción alias --lamafirst. Los resultados obtenidos fueron los siguientes:

-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

Aunque el número de goals ya no es necesario para el problema, seguimos necesitando declarar dicha variable para "human-needs". Vamos a realizar una serie de pruebas ejecutando JSHOP2. Para ello, modificamos el generador de problemas para que nos genere los nuevos problemas con la tarea (send-all-boxes).

-d	-r	-l	-p	-c	-g	Tiempo de ejecución	Coste del Plan
1	0	10	10	10	10	0.006	40
1	0	50	50	50	50	0.014	200
1	0	100	100	100	100	0.048	400
1	0	150	150	150	150	0.07	600
1	0	200	200	200	200	0.06	800
1	0	210	210	210	210	0.157	840
1	0	220	220	220	220	0.19	880
1	0	230	230	230	230	-	-

Cuando queremos ejecutar un problema el cuál tiene los siguientes parámetros: **-d 1 -r 0 -l 230 -p 230 -c 230 -g 230**, genera un error de espacio en el problema porque el "código es demasiado largo". Esto es porque hay un límite de bytes en los ficheros, y en este caso es superado.

```
(base) patri@patri:~/Documentos/JSHOP2$ ./jshop2-console.sh dronedom
/home/patri/Documentos/JSHOP2/jshop2-console/JSHOP2.jar:/home/patri/Documentos/
JSHOP2/jshop2-console/antlr.jar:.
problem.java:917: error: code too large
        private static void createState0(State s)       {
                           ^
1 error
Error: no se ha encontrado o cargado la clase principal problem
Causado por: java.lang.ClassNotFoundException: problem
```

Sin embargo, si comparamos los resultados del tiempo tardado con el algoritmo de planificación clásica Fast-Downward, con los resultados del algoritmo de planificación jerárquica SHOP2, podemos ver una clara diferencia a favor de SHOP2 que es capaz de resolver todos los problemas que se le han puesto por debajo del medio segundo.

3. **Explica los resultados y reflexiona sobre las ventajas e inconvenientes de la planificación jerárquica respecto a la planificación clásica.**

Respecto a la planificación clásica, podemos ver que con la planificación jerárquica obtenemos una serie de ventajas:

- Podemos obtener el coste de cada plan, mientras que con la planificación clásica no todos los planificadores calculaban los costes.
- Funciona de manera recursiva e independiente del dominio.
- Descompone las acciones en otras más primitivas para que las acciones logren el objetivo.
- Disminución de la complejidad de los problemas y a su vez el tiempo que tarda en resolverse

En cuanto a las desventajas:

- La planificación se limita en planificación jerárquica a buscar la mejor forma de descomponer el problema.
- Los planes generados son totalmente ordenados, es decir, no busca planes en paralelo.

Ejercicio 1.2. Comparativa de SHOP2 y Pyhop

El desarrollo del dominio de Pyhop se lleva a cabo de la siguiente manera. Introducimos a mano 2 humanos para probar la ejecución del plan respecto a SHOP2. Primero definimos los operadores:

- **move:** operador el cuál asigna a cada estado de donde están los drones, una localización.

```
# Operadores
def move(state, d, f, t):
    if state.drone_at[d] == f:
        state.drone_at[d] = t
        return state
    else: return False
```

- **pick_up:** para coger una caja, el dron y la caja deben estar en el mismo sitio, el dron y la caja deben de estar libres, y la caja debe tener el contenido que se pide.

  Tras esto, la caja deja de estar libre, la caja estará en el dron y el dron portará la caja

```
def pick_up(state, d, b, c, l):
    if (state.drone_at[d] == l) and (state.box_at[b] == l) and (state.drone_free[d]) and (state.box_has[b] == c) and (state.box_free[b]):
        state.drone_free[d] = False
        state.box_at[b] = d # localizacion de la caja pasa a ser el drone
        state.drone_carry[d] = b
        return state
    else: return False
```

- **drop:** igual que el anterior solo que para soltar la caja, el dron ha de llevarla y debe ser el contenido que necesita el humano.

  Una vez se suelte, el humano ya no necesitará esa caja y por tanto, eliminaremos esa instancia del humano en el diccionario de “human_needs”.

```python
def drop(state, d, b, c, l, h):
    if (state.drone_at[d] == l) and (state.drone_carry[d] == b) and (state.human_at[h] == l) and (state.box_has[b] == c):
        state.drone_carry[d] = False
        state.box_free[b] = False
        #state.human_needs[h].remove(c) # borrar contenido especifico de un humano
        #if (len(state.human_needs[h]) == 0):
        state.human_needs.pop(h)
        state.drone_free[d] = True
        state.box_at[b] = l
        return state
    else: return False
```

Una vez declarados, mediante la extensión hop los declaramos:

```python
hop.declare_operators(move, pick_up, drop)
```

Después definimos los métodos:

- **send_box:** Éste método lo que buscará es un contenido que corresponderá a una caja. Es posible que el contenido no exista en las cajas por lo que necesitamos un booleano que nos diga si el contenido ha sido encontrado en alguna caja.

  Si la caja ha sido encontrada y se encuentra en el almacén, debemos saber a dónde nos dirigimos (donde está el humano).

```python
# methods
def send_box(state, h, c):
    box_keys = state.box_has.keys()
    drone_keys = state.drone_at.keys()
    d = drone_keys[0] # tenemos el drone
    box_found = False

    for i in range (len(box_keys)):
        if state.box_has[box_keys[i]] == c:
            b = box_keys[i] # obtenemos la caja en la variable b
            box_found = True
            break
    if (box_found == True) and (state.box_at[b] == 'warehouse'):
        to = state.human_at[h] # localizacion a la que tiene que ir el drone
        f = 'warehouse' # from
        return [('pick_up', d, b, c, f),
                ('move', d, f, to),
                ('drop', d, b, c, to, h),
                ('move', d, to, f)]
    else:
        return False
```

- **send_all_boxes:** Llamada recursiva al método “sen_box”, el cuál recibe el humano y su necesidad. En caso de que no haya más humanos con más necesidades el método finalizará.

```
def send_all_boxes(state):
    if (len(state.human_needs) > 0):
        human_keys = state.human_needs.keys()
        return [('c', human_keys[0], state.human_needs[human_keys[0]]), ('h')]
    else:
        return []
```

Una vez definidos los métodos los declaramos en la clase hop. Debido a un problema con Pyhop debemos definir los métodos con un único carácter, es decir, *send_box* será **‘c’** y *send_all_boxes* será **‘h’**.

```
hop.declare_methods('c', send_box)
hop.declare_methods('h', send_all_boxes)
```

Después, inicializamos los estados de todas las variables utilizadas para sustituir la utilización de un “archivo problema”, de la siguiente manera:

```
state1 = hop.State('state1')

state1.drone_at = {'drone1' : 'warehouse'}

state1.box_at = {'box1': 'warehouse',
                 'box2': 'warehouse',
                 'box3': 'warehouse',
                 'box4': 'warehouse',
                 'box5': 'warehouse',
                 'box6': 'warehouse',
                 'box7': 'warehouse',
                 'box8': 'warehouse',
                 'box9': 'warehouse',
                 'box10': 'warehouse'}

state1.drone_free = {'drone1' : True}
```

```
state1.box_has = {'box1' :'meds',
                  'box2' : 'meds',
                  'box3' : 'meds',
                  'box4' : 'meds',
                  'box5' : 'meds',
                  'box6' : 'food',
                  'box7' : 'food',
                  'box8' : 'food',
                  'box9' : 'food',
                  'box10' : 'food'}

state1.box_free = {'box1' : True,
                   'box2' : True,
                   'box3' : True,
                   'box4' : True,
                   'box5' : True,
                   'box6' : True,
                   'box7' : True,
                   'box8' : True,
                   'box9' : True,
                   'box10' : True }

state1.drone_carry = {'drone1' : False}
```

```
state1.human_at = {'human1' : 'loc1',
                   'human2' : 'loc2',
                   'human3' : 'loc3',
                   'human4' : 'loc4',
                   'human5' : 'loc5',
                   'human6' : 'loc6',
                   'human7' : 'loc7',
                   'human8' : 'loc8',
                   'human9' : 'loc9',
                   'human10' : 'loc10'}

state1.human_needs = {'human1' : 'food',
                      'human2' : 'food',
                      'human3' : 'food',
                      'human4' : 'food',
                      'human5' : 'food',
                      'human6' : 'meds',
                      'human7' : 'meds',
                      'human8' : 'meds',
                      'human9' : 'meds',
                      'human10':  'meds'}
```

En nuestro caso, inicializamos *drone_at*, *box_at*, *drone_free*, *box_has*, *box_free*, *drone_carry*, *human_at* y *human_needs*.

Inicializamos mediante la siguiente línea de código:

```
hop.plan(state1, [('h')], hop.get_operators(), hop.get_methods(), verbose = 1)
```

Podríamos cambiar la opción verbose según lo que queramos ver. Con verbose 1 veremos la solución y el plan final. Con verbose 3, por ejemplo, veríamos todos los pasos intermedios que toma hasta llegar a la solución.

Mediante los siguientes parámetros vamos a comparar el funcionamiento de SHOP2 y Pyhop.

	-d	**-r**	**-l**	**-p**	**-c**	**-g**
JSHOP2	1	0	10	10	10	10
Pyhop	1	0	10	10	10	10

Plan SHOP2:

```
1 plan(s) were found:

Plan #1:
Plan cost: 40.0

(!pick-up drone1 box1 food warehouse)
(!move drone1 warehouse loc4)
(!drop drone1 box1 food loc4 human1)
(!move drone1 loc4 warehouse)
(!pick-up drone1 box2 food warehouse)
(!move drone1 warehouse loc1)
(!drop drone1 box2 food loc1 human2)
(!move drone1 loc1 warehouse)
(!pick-up drone1 box3 food warehouse)
(!move drone1 warehouse loc2)
(!drop drone1 box3 food loc2 human3)
(!move drone1 loc2 warehouse)
(!pick-up drone1 box4 food warehouse)
(!move drone1 warehouse loc10)
(!drop drone1 box4 food loc10 human4)
(!move drone1 loc10 warehouse)
(!pick-up drone1 box5 food warehouse)
(!move drone1 warehouse loc3)
(!drop drone1 box5 food loc3 human5)
(!move drone1 loc3 warehouse)
(!pick-up drone1 box10 meds warehouse)
(!move drone1 warehouse loc3)
(!drop drone1 box10 meds loc3 human5)
(!move drone1 loc3 warehouse)
(!pick-up drone1 box6 food warehouse)
(!move drone1 warehouse loc7)
(!drop drone1 box6 food loc7 human6)
(!move drone1 loc7 warehouse)
(!pick-up drone1 box7 food warehouse)
(!move drone1 warehouse loc10)
(!drop drone1 box7 food loc10 human8)
(!move drone1 loc10 warehouse)
(!pick-up drone1 box8 food warehouse)
(!move drone1 warehouse loc6)
(!drop drone1 box8 food loc6 human9)
(!move drone1 loc6 warehouse)
(!pick-up drone1 box9 food warehouse)
(!move drone1 warehouse loc7)
(!drop drone1 box9 food loc7 human10)
(!move drone1 loc7 warehouse)
--------------------

Time Used = 0.005
```

Plan Pyhop:

```
** hop, verbose=1: **
   state = state1
   tasks = ['h']
** result = [('pick_up', 'drone1', 'box5', 'meds', 'warehouse'), ('move', 'drone1', 'warehouse', 'l
oc9'), ('drop', 'drone1', 'box5', 'meds', 'loc9', 'human9'), ('move', 'drone1', 'loc9', 'warehouse'
), ('pick_up', 'drone1', 'box4', 'meds', 'warehouse'), ('move', 'drone1', 'warehouse', 'loc8'), ('d
rop', 'drone1', 'box4', 'meds', 'loc8', 'human8'), ('move', 'drone1', 'loc8', 'warehouse'), ('pick_
up', 'drone1', 'box1', 'meds', 'warehouse'), ('move', 'drone1', 'warehouse', 'loc7'), ('drop', 'dro
ne1', 'box1', 'meds', 'loc7', 'human7'), ('move', 'drone1', 'loc7', 'warehouse'), ('pick_up', 'dron
e1', 'box3', 'meds', 'warehouse'), ('move', 'drone1', 'warehouse', 'loc6'), ('drop', 'drone1', 'box
3', 'meds', 'loc6', 'human6'), ('move', 'drone1', 'loc6', 'warehouse'), ('pick_up', 'drone1', 'box9
', 'food', 'warehouse'), ('move', 'drone1', 'warehouse', 'loc5'), ('drop', 'drone1', 'box9', 'food'
, 'loc5', 'human5'), ('move', 'drone1', 'loc5', 'warehouse'), ('pick_up', 'drone1', 'box8', 'food',
 'warehouse'), ('move', 'drone1', 'warehouse', 'loc4'), ('drop', 'drone1', 'box8', 'food', 'loc4',
'human4'), ('move', 'drone1', 'loc4', 'warehouse'), ('pick_up', 'drone1', 'box7', 'food', 'warehous
e'), ('move', 'drone1', 'warehouse', 'loc3'), ('drop', 'drone1', 'box7', 'food', 'loc3', 'human3'),
 ('move', 'drone1', 'loc3', 'warehouse'), ('pick_up', 'drone1', 'box6', 'food', 'warehouse'), ('mov
e', 'drone1', 'warehouse', 'loc2'), ('drop', 'drone1', 'box6', 'food', 'loc2', 'human2'), ('move',
'drone1', 'loc2', 'warehouse'), ('pick_up', 'drone1', 'box10', 'food', 'warehouse'), ('move', 'dron
e1', 'warehouse', 'loc1'), ('drop', 'drone1', 'box10', 'food', 'loc1', 'human1'), ('move', 'drone1'
, 'loc1', 'warehouse'), ('pick_up', 'drone1', 'box2', 'meds', 'warehouse'), ('move', 'drone1', 'war
ehouse', 'loc10'), ('drop', 'drone1', 'box2', 'meds', 'loc10', 'human10'), ('move', 'drone1', 'loc1
0', 'warehouse')]
```

Se plantean las siguientes preguntas:

1. **¿Qué ventajas tiene Pyhop respecto a SHOP2? ¿Qué inconvenientes?**

En Pyhop tratamos tareas totalmente ordenadas, mientras que con SHOP2 son tareas parcialmente ordenadas.

Además, Pyhop representa estados del mundo utilizando enlaces de variables ordinarias, no proposiciones lógicas. Un estado es solo un objeto de Python que contiene los enlaces de variables (operadores y métodos se refieren a estados explícitamente). Por ejemplo, se puede escribir s.loc['v'] = 'd' para decir que el vehículo v está en la ubicación d en el estado s.

Además, los métodos y operadores HTN son funciones ordinarias de Python y es muy fácil de implementar y entender (<150 líneas de código).

Pero tendría un inconveniente y es que en SHOP2 no era necesario pasar por parámetros los argumentos estrictamente necesarios (la localización exacta, el humano exacto, etc), si no que SHOP2 buscaba automáticamente los valores de los parámetros según el plan generado (no se unifican las variables automáticamente).

2. **Leyendo la siguiente presentación (http://www.cs.umd.edu/~nau/papers/nau2013game.pdf) explica los problemas existentes para aplicar planificación automática en videojuegos, los problemas que resuelve Pyhop y los retos pendientes.**

Los problemas existentes al aplicar planificación automática en videojuegos son:

- **Estado**: en un videojuego se requieren estructuras de datos, y una planificación automática solo puede proporcionar una serie de proposiciones.
- **Acciones**: en un videojuego se requiere poder modificar las estructuras de datos, pero la planificación automática, al solo permitirnos el uso de preposiciones, solo nos deja añadir o eliminar nuevas proposiciones.
- **Agentes**: en un videojuego se necesitan muchos agentes, mientras que con la planificación automática solo podemos tener uno.
- **Mundo**: en un videojuego el mundo donde se desarrolla el juego debe ser dinámico, y la planificación automática solo nos permite tener un mundo estático.
- **Tiempo disponible**: en un videojuego el tiempo disponible es muy pequeño, mientras que con planificación automática es aquel que el planificador necesite.
- **Objetivo**: en un videojuego lo que queremos es obtener una solución parcial al problema actual, y con la planificación automática solo podemos encontrar la solución completa.
- **Ejecución**: en un videojuego normalmente la ejecución se lleva a cabo durante la planificación, mientras que con planificación automática solo podemos comenzar con la ejecución una vez haya terminado y obtenido el plan.

De todos estos problemas, sólo algunos podrían arreglarse de manera sencilla, ya que otros son problemas demasiado complejos.

Pyhop resuelve la incompatibilidad de estado, que en los juegos es referida a las estructuras de datos. También resuelve la incompatibilidad de las acciones que en los juegos se refiere a la modificación de la estructura de datos.

Como bien sabemos, no resuelve todos los problemas que puede tener un juego, como son los agentes, el mundo, el tiempo disponible, el objetivo y la ejecución.

3. **Una de las mayores desventajas de Pyhop respecto a SHOP es que no se unifican los parámetros de las tareas automáticamente, sino que hay que asignar las variables por código, pudiendo usar para ello cualquier código que deseemos. Explica cómo afecta esto al backtracking que realiza el planificador respecto a SHOP2.**

En SHOP2 si algún parámetro lleva a error, se debe hacer backtracking y probar otro método. Lo bueno es que varias instancias podrían coincidir con esta tarea y ser válidas para el problema.

En Pyhop, los parámetros que se pasan al método deben de ser iguales que los de la tarea, por lo que únicamente una instancia es la buena para ese método.

El problema de que sea así en Pyhop es que las subtareas se van a conocer en tiempo de ejecución, y sin embargo, en SHOP2 se conocen de antes. Lo bueno de Pyhop es que tiene la capacidad de saber la mejor secuencia de subtareas en tiempo de ejecución.

4. **Los planificadores jerárquicos suelen disponen de la posibilidad de llamar a funciones externas (ej.: funciones LISP con SHOP2 o funciones Python en Pyhop) que se encarguen de asignar variables o evaluar precondiciones, traduciéndose en la práctica en que parte de la planificación se realiza de forma separada. Reflexiona sobre algún dominio de planificación en el que esto pueda resultar especialmente ventajoso.**

Realmente será ventajoso para todos los dominios, aunque será más ventajoso si tenemos más datos. Esto es porque por ejemplo el lenguaje LISP dispone de liberación automática de memoria (lo cual para problemas grandes sería ventajoso), pero es menos eficiente que otros lenguajes, por lo cual para un problema pequeño podría ser perjudicial ya que retrasaría la ejecución.

Parte 2: Planificación jerárquica avanzada

Ejercicio 2.1. Logística de emergencias avanzada en SHOP2

Los primeros pasos a realizar para modificar este dominio es eliminar “los humanos” y “las cajas”. Ahora únicamente tenemos localizaciones que necesitan ciertos contenidos.

El coste que tenemos en cuenta en el proceso es el siguiente: cada vez que el drone se mueve tiene un coste de 50. En cada trayecto tiene que añadir el coste de la capacidad del carrier.

Para que el coste de un viaje sea ida + vuelta + la carga del transportador, la última es sumada a la hora de dropear una caja ya sea desde el dron o desde el carrier.

Los parámetros que utiliza cada operador son los siguientes:

- **put-box-on-carrier** drone carrier loc

```
(:operator (!put-box-on-carrier ?d ?r ?l)

    ( ; preconditions
        (DRONE ?d) (CARRIER ?r) (LOCATION ?l)
        (drone-at ?d ?l)
        (carrier-at ?r ?l)
        (drone-free ?d)
    )
    ( ; delete
    )
    ( ; add
    )
    0

)
```

- **move-carrier** drone carrier loc_from loc_to

```
(:operator (!move-carrier ?d ?r ?f ?t)

    (; preconditions
        (DRONE ?d) (CARRIER ?r) (LOCATION ?f) (LOCATION ?t)
        (drone-at ?d ?f)
        (carrier-at ?r ?f)
    )
    ( ; delete
        (drone-at ?d ?f)
        (carrier-at ?r ?f)
    )
    ( ;add
        (drone-at ?d ?t)
        (carrier-at ?r ?t)
    )
    50;cost

)
```

- **take-off-box-carrier** drone carrier loc loc_needs carrier-capacity

```
(:operator (!take-off-box-carrier ?d ?r ?l ?n ?c)

    ( ; preconditions
        (DRONE ?d) (CARRIER ?r) (LOCATION ?l) (CAPACITY ?r ?c)
        (drone-at ?d ?l)
        (carrier-at ?r ?l)
        (loc-needs ?l ?n)
    )
    ( ; delete
        (loc-needs ?l ?n)
    )
    ( ; add
        (loc-needs ?l (call - ?n ?c))
    )
    ?c

)
```

- **pick-up** drone loc

```
(:operator (!pick-up ?d ?l)

    ( ; preconditions
        (DRONE ?d) (LOCATION ?l)
        (drone-at ?d ?l)
        (drone-free ?d)
    )
    ( ; delete
        (drone-free ?d)
    )
    ( ; add

    )
    0

)
```

- **move** drone loc_from loc_to

```
(:operator (!move ?d ?f ?t)

    ( ; preconditions
        (DRONE ?d) (LOCATION ?f) (LOCATION ?t)
        (drone-at ?d ?f)
    )
    ( ; delete
        (drone-at ?d ?f)
    )
    ( ; add
        (drone-at ?d ?t)
    )
    50

)
```

- **drop** drone loc loc_needs

```
(:operator (!drop ?d ?l ?n)

    ( ; preconditions
       (DRONE ?d) (LOCATION ?l)
       (loc-needs ?l ?n)
       (drone-at ?d ?l)
    )
    ( ; delete
       (loc-needs ?l ?n)
    )
    ( ; add
       (drone-free ?d)
       (loc-needs ?l (call - ?n 1))
    )
    1

)
```

Un lugar necesita 50 cajas y solo hay un transportador para 10 cajas.

El drone 1 mueve el carrier, que coge las cajas de 10 en 10 ya que esa es su capacidad. Realiza 10 moves, 5 llevando 10 cajas y 5 volviendo al almacén. En cada movimiento suma 50 de coste, y por cada vez que mueve el carrier añade 10 por ser la capacidad del carrier.

```
1 plan(s) were found:

Plan #1:
Plan cost: 550.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 50.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 40.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 30.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 20.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 10.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
--------------------

Time Used = 0.004
```

Un lugar necesita 51 cajas y solo hay un transportador para 10 cajas.

Como podemos ver, mueve el carrier para entregar las cajas de 10 en 10, y cuando queda una única caja por repartir, la coge, la mueve y la suelta directamente con el drone.

```
1 plan(s) were found:

Plan #1:
Plan cost: 651.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 51.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 41.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 31.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 21.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 11.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!pick-up drone1 warehouse)
(!move drone1 warehouse loc1)
(!drop drone1 loc1 1.0)
(!move drone1 loc1 warehouse)
--------------------

Time Used = 0.007
```

Un lugar necesita 50 cajas y solo hay un transportador para 40 cajas.

Como podemos ver, el transportador al ser únicamente de tamaño 40, nos obliga a realizar dos viajes. En el primer viaje cargamos 40 cajas al transportador, y tras volver al warehouse, únicamente cargará 10 cajas, que son las que se siguen necesitando esa localización.

```
1 plan(s) were found:

Plan #1:
Plan cost: 280.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 50.0 40.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 10.0 40.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
--------------------

Time Used = 0.005
```

Un lugar necesita 50 cajas y hay transportadores de 10, 20 y 30 cajas.

En este caso, a pesar de que se necesitan 50 cajas y hay tres transportadores de distintos tamaños, el planificador escoge siempre el transportador de 10 cajas, lo cual no es óptimo. Lo más óptimo sería realizar dos viajes, uno con el transportador de 30 cajas y luego otro con el de 20 cajas. Realmente lo hace así porque el planificador realiza el primer plan encontrado aunque no sea óptimo.

```
1 plan(s) were found:

Plan #1:
Plan cost: 550.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 50.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 40.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 30.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 20.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 10.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
--------------------

Time Used = 0.008
```

Por diferentes pruebas realizadas hemos podido ver que depende del órden en el cuál se pongan la capacidad de los transportadores cogerá una u otra. En este caso, en el problema hemos definido primero la capacidad del carrier 3 (30) y el plan generado es el siguiente:

```
1 plan(s) were found:

Plan #1:
Plan cost: 260.0

(!put-box-on-carrier drone1 carrier3 warehouse)
(!move-carrier drone1 carrier3 warehouse loc1)
(!take-off-box-carrier drone1 carrier3 loc1 50.0 30.0)
(!move-carrier drone1 carrier3 loc1 warehouse)
(!put-box-on-carrier drone1 carrier3 warehouse)
(!move-carrier drone1 carrier3 warehouse loc1)
(!take-off-box-carrier drone1 carrier3 loc1 20.0 30.0)
(!move-carrier drone1 carrier3 loc1 warehouse)
--------------------

Time Used = 0.007
```

Un lugar necesita 50 cajas y hay transportadores de 20, 50 y 100 cajas.

En este caso, aunque tenemos 3 transportadores, el planificador mueve siempre el primer carrier definido, aunque lo más óptimo sería que usase el transportador de 50 para realizar un único viaje. Sin embargo, realiza dos viajes de 20 y el tercero transportando 10 (usando el de 20).

```
1 plan(s) were found:

Plan #1:
Plan cost: 360.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 50.0 20.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 30.0 20.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 10.0 20.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
--------------------

Time Used = 0.013
```

Un lugar necesita 20, otro 50 y otro 100 cajas y hay transportadores para esos mismos números.

Como en los casos anteriores, coge el carrier1 siempre, de manera que siempre realiza viajes de 20 en 20. Por lo tanto, no es óptimo el plan que encuentra pero lo hace de manera correcta.

```
Plan #1:
Plan cost: 1080.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 20.0 20.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc2)
(!take-off-box-carrier drone1 carrier1 loc2 50.0 20.0)
(!move-carrier drone1 carrier1 loc2 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc3)
(!take-off-box-carrier drone1 carrier1 loc3 100.0 20.0)
(!move-carrier drone1 carrier1 loc3 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc2)
(!take-off-box-carrier drone1 carrier1 loc2 30.0 20.0)
(!move-carrier drone1 carrier1 loc2 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc3)
(!take-off-box-carrier drone1 carrier1 loc3 80.0 20.0)
(!move-carrier drone1 carrier1 loc3 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc2)
(!take-off-box-carrier drone1 carrier1 loc2 10.0 20.0)
(!move-carrier drone1 carrier1 loc2 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc3)
(!take-off-box-carrier drone1 carrier1 loc3 60.0 20.0)
(!move-carrier drone1 carrier1 loc3 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc3)
(!take-off-box-carrier drone1 carrier1 loc3 40.0 20.0)
(!move-carrier drone1 carrier1 loc3 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc3)
(!take-off-box-carrier drone1 carrier1 loc3 20.0 20.0)
(!move-carrier drone1 carrier1 loc3 warehouse)
--------------------

Time Used = 0.004
```

Un lugar necesita 50 cajas y otro 100 cajas y hay un transportador para 150 cajas.

En este caso, aunque el transportador tiene capacidad de 150 cajas, nosotros en el refinamiento le hemos indicado que siempre que vuelve al warehouse debe cargar cajas, moverse y soltarlas. Por lo tanto, en vez de rellenar una única vez el transportador, primero carga 50 cajas para dejarlo en la primera localización y después 100 en la segunda.

```
1 plan(s) were found:

Plan #1:
Plan cost: 500.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 50.0 150.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc2)
(!take-off-box-carrier drone1 carrier1 loc2 100.0 150.0)
(!move-carrier drone1 carrier1 loc2 warehouse)
--------------------

Time Used = 0.006
```

Ejercicio 2.2. Optimización del planificador

Como hemos visto en los casos del ejercicio 2.1, el planificador siempre coge el primer transportador que se define sin comprobar cuál se ajusta más a las necesidades de la localización. Además, en algún caso en vez de rellenar el transportador entero para realizar varias acciones no aprovecha el tamaño del carrier. Para ello, en este apartado, añadimos ciertas precondiciones nuevas para optimizar el plan final.

Se piden dos optimizaciones:

a) **Generar planes en los que se atienda primero a los lugares con mayor necesidad total de recursos**

Para ello, modificamos send-all-boxes de la siguiente manera:

```
(:method (send-all-boxes) ; enviar-todo

    ( ; preconditions
        :sort-by ?n > (and (loc-needs ?l ?n) (call > ?n 0))
    )
    ( ; subtasks
        (send-box ?l ?n) (send-all-boxes)
    )

    () ; preconditions
    () ; subtasks

)
```

Con esto conseguimos que las primeras localizaciones que se atiendan sean las de mayor necesidad.

b) **Generar planes que hagan un uso eficiente de los transportadores eligiendo el transportador a enviar según su capacidad de cargas y el número de cajas que se necesitan en el destino**

Añadiendo un nuevo refinamiento para el caso en el cual el carrier sea del mismo tamaño que las necesidades, seleccionará dicho carrier.

En caso de que no haya un carrier con el mismo tamaño que la necesidad, seleccionará mediante sort-by el carrier de máximo tamaño para realizar el mínimo número de viajes, aunque el coste sea a veces mayor y no óptimo.

```
(:method (send-box ?to ?n)

    ( ; preconditions para mover drone
        (DRONE ?d)
        (drone-at ?d ?from)
        (call = ?n 1)
    )
    ( ; subtasks
        (!pick-up ?d ?from)
        (!move ?d ?from ?to)
        (!drop ?d ?to ?n)
        (!move ?d ?to ?from)
    )

    ( ; preconditions para mover carrier
        (DRONE ?d) (CAPACITY ?r ?c) (call = ?c ?n) (drone-at ?d ?from) (carrier-at ?r ?from) (call > ?n 1)
    )
    ( ; subtasks del carrier
        (!put-box-on-carrier ?d ?r ?from)
        (!move-carrier ?d ?r ?from ?to)
        (!take-off-box-carrier ?d ?r ?to ?n ?c)
        (!move-carrier ?d ?r ?to ?from)
    )

    ( ; preconditions para mover carrier
        :sort-by ?c > (and (DRONE ?d) (CAPACITY ?r ?c) (drone-at ?d ?from) (carrier-at ?r ?from) (call > ?n 1))
    )
    ( ; subtasks del carrier
        (!put-box-on-carrier ?d ?r ?from)
        (!move-carrier ?d ?r ?from ?to)
        (!take-off-box-carrier ?d ?r ?to ?n ?c)
        (!move-carrier ?d ?r ?to ?from)
    )

)
```

Un lugar necesita 50 cajas y solo hay un transportador para 10 cajas.

El plan encontrado es igual al de la primera parte. El coste sigue siendo **550**.

```
1 plan(s) were found:

Plan #1:
Plan cost: 550.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 50.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 40.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 30.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 20.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 10.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
--------------------

Time Used = 0.007
```

Un lugar necesita 51 cajas y solo hay un transportador para 10 cajas.

El plan encontrado es igual al de la primera parte. El coste sigue siendo **651**.

```
1 plan(s) were found:

Plan #1:
Plan cost: 651.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 51.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 41.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 31.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 21.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 11.0 10.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!pick-up drone1 warehouse)
(!move drone1 warehouse loc1)
(!drop drone1 loc1 1.0)
(!move drone1 loc1 warehouse)
--------------------

Time Used = 0.008
```

Un lugar necesita 50 cajas y solo hay un transportador para 40 cajas.

El plan encontrado es igual al de la primera parte. El coste sigue siendo **280**.

```
1 plan(s) were found:

Plan #1:
Plan cost: 280.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 50.0 40.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 10.0 40.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
--------------------

Time Used = 0.004
```

Un lugar necesita 50 cajas y hay transportadores de 10, 20 y 30 cajas.

El plan y el coste cambian respecto a la primera parte. En esta ocasión, se selecciona el carrier de mayor tamaño, es decir, el de 30. Y al terminar de mover esas 30 cajas quedarán 20, para las cuales moverá el carrier de 20, minimizando así el coste de las acciones y **optimizando al máximo**. El coste es **250**.

```
1 plan(s) were found:

Plan #1:
Plan cost: 250.0

(!put-box-on-carrier drone1 carrier3 warehouse)
(!move-carrier drone1 carrier3 warehouse loc1)
(!take-off-box-carrier drone1 carrier3 loc1 50.0 30.0)
(!move-carrier drone1 carrier3 loc1 warehouse)
(!put-box-on-carrier drone1 carrier2 warehouse)
(!move-carrier drone1 carrier2 warehouse loc1)
(!take-off-box-carrier drone1 carrier2 loc1 20.0 20.0)
(!move-carrier drone1 carrier2 loc1 warehouse)
--------------------

Time Used = 0.004
```

Un lugar necesita 50 cajas y hay transportadores de 20, 50 y 100 cajas.

El plan y el coste cambian respecto a la primera parte. En este caso, mueve directamente el carrier de tamaño 50 para transportar todas las cajas en un único movimiento, minimizando así el coste de las acciones y **optimizando al máximo**. El coste es **150**.

```
1 plan(s) were found:

Plan #1:
Plan cost: 150.0

(!put-box-on-carrier drone1 carrier2 warehouse)
(!move-carrier drone1 carrier2 warehouse loc1)
(!take-off-box-carrier drone1 carrier2 loc1 50.0 50.0)
(!move-carrier drone1 carrier2 loc1 warehouse)
--------------------

Time Used = 0.003
```

Un lugar necesita 20, otro 50 y otro 100 cajas y hay transportadores para esos mismos números.

El plan y el coste cambian respecto a la primera parte. Ahora los viajes se hacen empezando por el sitio que tiene más necesidades, y en todos los viajes coge los transportadores que corresponden al tamaño de las necesidades, minimizando así el coste de las acciones y **optimizando al máximo**. El coste es **470**.

```
1 plan(s) were found:

Plan #1:
Plan cost: 470.0

(!put-box-on-carrier drone1 carrier3 warehouse)
(!move-carrier drone1 carrier3 warehouse loc3)
(!take-off-box-carrier drone1 carrier3 loc3 100.0 100.0)
(!move-carrier drone1 carrier3 loc3 warehouse)
(!put-box-on-carrier drone1 carrier2 warehouse)
(!move-carrier drone1 carrier2 warehouse loc2)
(!take-off-box-carrier drone1 carrier2 loc2 50.0 50.0)
(!move-carrier drone1 carrier2 loc2 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 20.0 20.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
--------------------

Time Used = 0.004
```

a

Un lugar necesita 50 cajas y otro 100 cajas y hay un transportador para 150 cajas.

En este caso, cambia el plan pero no el coste. Primero entrega las cajas a la localización mayor de las dos utilizando el único carrier que hay disponible.

```
1 plan(s) were found:

Plan #1:
Plan cost: 500.0

(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc2)
(!take-off-box-carrier drone1 carrier1 loc2 100.0 150.0)
(!move-carrier drone1 carrier1 loc2 warehouse)
(!put-box-on-carrier drone1 carrier1 warehouse)
(!move-carrier drone1 carrier1 warehouse loc1)
(!take-off-box-carrier drone1 carrier1 loc1 50.0 150.0)
(!move-carrier drone1 carrier1 loc1 warehouse)
--------------------

Time Used = 0.005
```

COMPARATIVA DE COSTES CON Y SIN OPTIMIZACIÓN

Para ver de una forma mucho más sencilla el resultado de la optimización realizada, llevamos a cabo una tabla para comprobar resultados del ejercicio 2.1 y 2.2.

Problema	1	2	3	4	5	6	7
Coste normal	550	651	280	550	360	1080	500
Coste optimización	550	651	280	250	150	470	500