

PRACTICA 2

PLANIFICACIÓN AUTOMÁTICA

Blanca Calderón González
Franck Michael Fierro Chicaiza

Índice

Parte I: Planificación jerárquica con SHOP y PYHOP	3
Ejercicio 1.1 : Logísticas de emergencias en SHOP2	3
Explicación del código:	3
-Dominio de planificación:	3
-Generador del problema:	5
Preguntas:	6
1. Explica cómo crece el tiempo que tarda en generar una solución según el tamaño del problema, elaborando una gráfica.	6
2. Compáralo los resultados anteriores con los del planificador que mejor rendimiento mostró en el ejercicio 1.3 de la práctica 1.	8
3. Explica los resultados y reflexiona sobre las ventajas e inconvenientes de la planificación jerárquica respecto a la planificación clásica.	10
Ejercicio 1.2: Comparativa de SHOP2 Y PYHOP	12
Explicación del código:	12
-Dominio de planificación:	12
-Generador del problema:	13
Preguntas:	14
1. ¿Qué ventajas tiene Pyhop respecto a SHOP2? ¿Qué inconvenientes?	14
2. Explica los problemas existentes para aplicar planificación automática en videojuegos, los problemas que resuelve Pyhop y los retos pendientes.	18
3. Explica cómo afecta esto al backtracking que realiza el planificador respecto a SHOP2.	19
4. Reflexiona sobre algún dominio de planificación en el que esto pueda resultar especialmente ventajoso.	19
Parte 2: Planificación jerárquica avanzada	20
Ejercicio 2.1 : Logísticas de emergencias avanzada en SHOP2	20
Explicación del código:	20
-Dominio de planificación:	20
-Generador del problema:	24
Demostrar el funcionamiento al menos bajo las siguientes circunstancias:	25
Ejercicio 2.2: Optimización del planificador	36
Explicación del código:	36
-Dominio de planificación:	36
-Generador del problema:	37
Explicar cómo las modificaciones realizadas mejoran los resultados obtenidos por el planificador en los casos planteados en el ejercicio anterior.	37

Parte I: Planificación jerárquica con SHOP y PYHOP

Ejercicio 1.1 : Logísticas de emergencias en SHOP2

Explicación del código:

-Dominio de planificación:

Se ha hecho el mismo dominio que en ejercicio 1.1 de la primera práctica contando con un solo dron (pero adaptado a que pueda haber varios) con dos brazos que lleva la caja a cada persona que necesita el contenido de ésta.

Los cambios llevados a cabo para hacer esto en SHOP2 han sido cambiar la estructura para que sea SHOP2, lo más característico es que ya no se definen los predicados en el dominio.

Además, se ha modificado la definición del dominio de forma que se cumplan los requisitos del enunciado al contar con un método enviar-todo en el cual el dron entrega todas las cajas a las personas que lo necesitan y un método realizar-entrega que es usado por el anterior.

Los cambios en la estructura se pueden observar claramente en la acción mover-dron el cual en el ejercicio 1.1 de la primera práctica era de la siguiente forma:

```
(:action mover-dron
  :parameters (
    ?d - dron
    ?from ?to - localizacion
  )
  :precondition (and
    (dron-en ?d ?from)
  )
  :effect (and
    (not (dron-en ?d ?from))
    (dron-en ?d ?to)
  )
)
```

En cambio actualmente se definiría así:

```
(:operator (!mover-dron ?d ?from ?to)
  (
    (DRON ?d)
    (LOCALIZACION ?from)
```

```

(LOCALIZACION ?to)
(dron-en ?d ?from)
)
((dron-en ?d ?from))
((dron-en ?d ?to))
)

```

Las principales diferencias se pueden ver que ya no se definen como action sino como operators, que las precondiciones se juntan con los parámetros y que el efecto negado desaparece.

Los dos nuevos métodos añadidos son los siguientes:

-Enviar-todo → Se encarga de enviar todas las cajas necesarias a las personas que lo necesitan haciendo uso de la recursividad como se observa en su estructura en la que primero se comprueba que siga habiendo alguna persona que necesita algún contenido ya que si no terminamos el proceso al haber enviado todas las cajas necesarias (condición de parada de la recursividad) y si hay se llama al método realizar entrega que explicaremos a continuación.

Destacar que para mayor simplificación está programado de forma que el dron comience el envío en el depósito y vuelva a él al finalizar tal y como se observa en realizar-envio.

A continuación, se observa su estructura:

```

(:method (enviar-todo)
  ((persona-necesita ?p ?ct))
  (
    (realizar-entrega ?p ?ct)
    (enviar-todo)
  )
  ()
  ()
)

```

-Realizar-entrega → Este método es llamado por enviar-todo y en él se hacen las comprobaciones para realizar las acciones definidas para realizar una entrega (coger-caja, mover-dron, entregar-caja), estas comprobaciones también sirven para sacar el valor de los parámetros ya que SHOP2 tiene la característica de generar automáticamente los valores de los parámetros por lo que se obtendría el valor del contenido (caja-contiene ?c ?ct), la localización de la persona (persona-en ?p ?l) y el brazo del dron(dron-tiene ?d ?b), también se comprueba que el dron empiece en el depósito.

Tras estas comprobaciones se llaman a los respectivos operators para realizar la entrega de la caja a la persona cogiendo primero la caja con el contenido necesario del depósito (donde originalmente esta el dron), moviendo el dron del depósito a la

localización de la persona dónde realiza su entrega y finalmente volviendo al depósito para coger la siguiente caja.

```
(:method (realizar-entrega ?p ?ct)
  (
    (caja-contiene ?c ?ct)
    (persona-en ?p ?l)
    (dron-en ?d deposito)
    (dron-tiene ?d ?b)
  )
  (
    (!coger-caja ?d ?b ?c ?ct deposito)
    (!mover-dron ?d deposito ?l)
    (!entregar-caja ?d ?b ?c ?ct ?l ?p)
    (!mover-dron ?d ?l deposito)
  )
)
```

-Generador del problema:

Los principales cambios en el generador de problemas es que se han eliminado la definición de init y goal (la meta ahora es la llamada al método enviar-todo) ya que no son necesarios en el problema, se ha cambiado el nombre a defproblem problem drone y la forma de crear los objetos como se observa en el código mostrado más adelante.

El código nuevo realizado es el siguiente:

```
for x in drone:
    f.write("\t(DRON " + x + ")\n")

    for i in range(len(drone) * 2):
        f.write("\t(BRAZO brazo" + str(i + 1) + ")\n")

f.write("\t(LOCALIZACION deposito)\n")
for x in location:
    f.write("\t(LOCALIZACION " + x + ")\n")

for x in crate:
    f.write("\t(CAJA " + x + ")\n")

for x in content_types:
    f.write("\t(CONTENIDO " + x + ")\n")

for x in person:
    f.write("\t(PERSONA " + x + ")\n")
```

```
f.write("\n")
```

Preguntas:

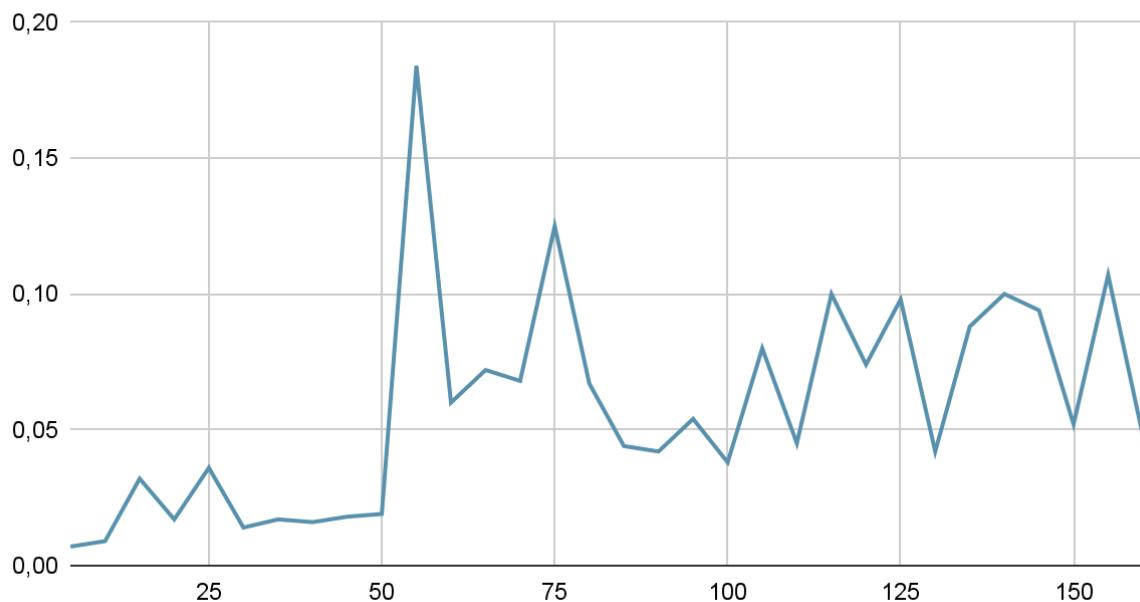
1. Explica cómo crece el tiempo que tarda en generar una solución según el tamaño del problema, elaborando una gráfica.

Tamaño problema	Tiempo	Nº acciones
5	0,007	20
10	0,009	40
15	0,032	60
20	0,017	80
25	0,036	100
30	0,014	120
35	0,017	140
40	0,016	160
45	0,018	180
50	0,019	200
55	0,184	220
60	0,06	240
65	0,072	260
70	0,068	280
75	0,125	300
80	0,067	320
85	0,044	340
90	0,042	360
95	0,054	380
100	0,038	400
105	0,08	420
110	0,045	440

115	0,1	460
120	0,074	480
125	0,098	500
130	0,042	520
135	0,088	540
140	0,1	560
145	0,094	580
150	0,052	600
155	0,107	620
160	0,048	640

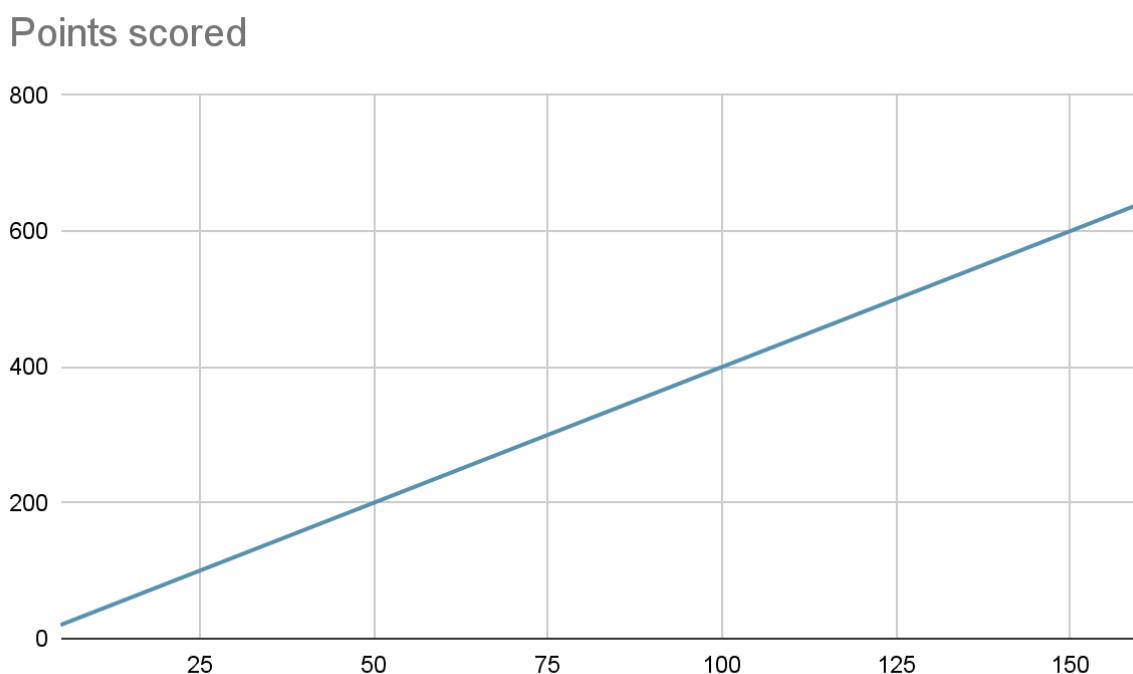
-Gráfica comparativa del tamaño del problema con el tiempo de ejecución (X → tamaño problema, y → tiempo)

Points scored



Observando la gráfica se ve que no hay una relación clara entre el tiempo de ejecución y el tamaño del problema ya que hay veces que con un tamaño mayor disminuye el tiempo de ejecución aunque en general si se observa que cuando aumenta el tamaño del problema aumenta también el tiempo de ejecución necesario para elaborar el plan (si observamos el tiempo cuando el tamaño del problema es 25 y cuando es 150 se observa este aumento aunque no sea constante)

**-Gráfica comparativa del tamaño del problema con el número de acciones realizadas
(x -> tamaño del problema, y → nº Acciones)**



En cambio la relación entre el número de acciones realizadas con el número de acciones si es constante y directamente proporcional como observamos en la gráfica.

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

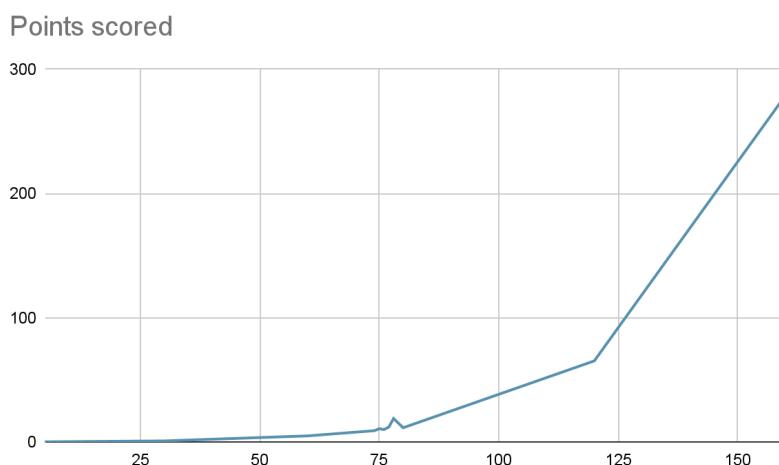
El planificador que mejor rendimiento mostró en el ejercicio 1.3 de la práctica 1 fue el fastDownward resolviendo un problema de tamaño 160 en 278,952 segundos.

Resultados fastDownward Ejercicio 1.3 Practica 1

Tamaño del problema	Tiempo
5	0,08
30	0,8
60	4,84
74	9,03
75	10,645
76	9,95

77	11,93
78	18,84
80	11,3856
120	65,199
160	278,952

-Gráfica fastDownward

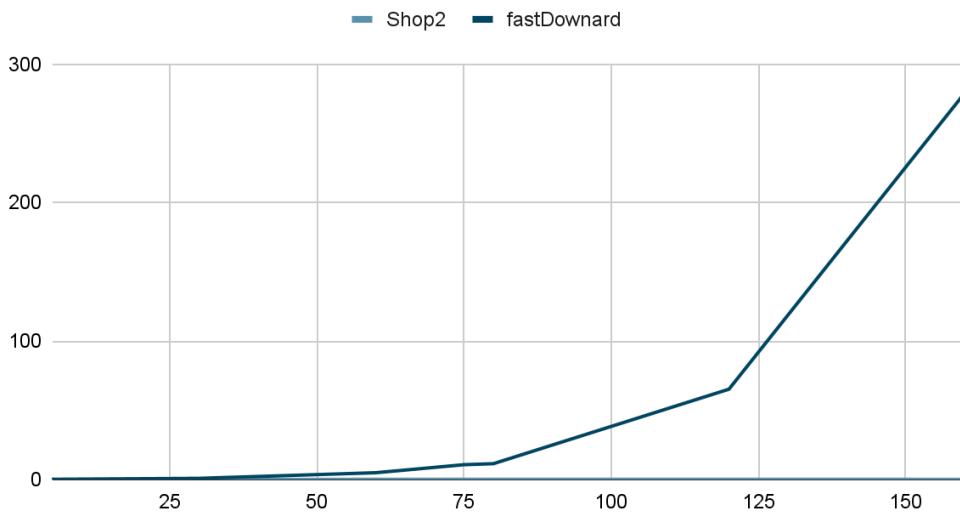


-Tabla comparativa con los datos comunes de fastDownward y SHOP2

Tamaño problema	Tiempos SHOP2	Tiempos FastDownward
5	0,07	0,08
30	0,014	0,8
60	0,06	4,84
75	0,125	10,645
80	0,067	11,3856
120	0,074	65,199
160	0,048	278,952

-Gráfica comparativa de fastDownward y shop2 (como los valores de shop2 son muy pequeños menores de 1 en la gráfica parece una línea constante en comparación con los grandes valores de fastDownward)

Points scored



Analizando los datos de las tablas obtenidos de ambos planificadores nos damos cuenta con diferencia que SHOP2 tiene un mayor rendimiento que fastDownward con bastante diferencia ya que si cogemos por ejemplo el mayor tamaño de problema que es 160 y comparamos los tiempos conseguidos en ambos planificador se observa que en SHOP2 el tiempo es muchísimo menor siendo este 0,048 ni siquiera llegando a tardar un minuto a diferencia del tiempo en fastDownward que es 278,952 por lo que hay una diferencia de casi de 5 minutos.

Esto también se observa en la gráfica comparativa de ambos planificadores en la cual a partir de un tamaño de problema 50 el tiempo de ambos empieza a ser muy diferente aumentando en gran medida los tiempos de fastDownward y quedándose los de SHOP2 muy cercanos a 0 (menores de 1 por lo que parece una línea recta en comparación con los grandes valores de fastDownward).

En conclusión, el planificador SHOP2 tiene mayor rendimiento que el fastDownward(planificador con mayor rendimiento de la práctica anterior) teniendo una gran diferencia de tiempos siendo mucho más rápido SHOP2.

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

Como se ha mencionado en el ejercicio anterior los resultados muestran claramente que el planificador SHOP2 tiene mayor rendimiento resolviendo problemas de mayor tamaño en un tiempo considerablemente menor que el planificador clásico fastDownward.

Las **ventajas** de la planificación jerárquica son:

- Estructura clara y coherente.

-Permite obtener el coste de cada plan a diferencia de la planificación clásica en la cual no todos los planificadores calculaban los costes.

-Tiene un enfoque basado en objetivos , es decir, establece objetivos y metas claras permitiendo desglosar los objetivos en otros más específicos mientras que la clásica se centra en realizar tareas específicas.

-Descompone las acciones en unas más primitivas de forma que las acciones logren el objetivo.

-Mayor flexibilidad al permitir ajustar los planes a los cambios en el entorno.

-Disminuye la complejidad de los problemas.

-Reduce el tiempo que tarda en resolver los problemas.

Por otra parte sus **inconvenientes** son:

-Puede ser demasiado rígida en su estructura dificultando la adaptación a cambios imprevistos.

-Solo busca la mejor forma de descomponer el problema.

-No busca planes en paralelo ya que los planes generados son totalmente ordenados.

En resumen la planificación jerárquica tiene numerosas ventajas frente a la clásica permitiendo la descomposición en acciones más primitivas y disminuyendo la complejidad y tiempo de resolución de los problemas como hemos apreciado en la anterior pregunta en la cual el planificador SHOP2 tardaba mucho menos en resolver el problema en comparación con el fastDownward.

Ejercicio 1.2: Comparativa de SHOP2 Y PYHOP

Explicación del código:

-Dominio de planificación:

En este caso se ha utilizado el mismo dominio que para el ejercicio 1.1 modificándolo para que sea Pyhop (programándolos en Python).

En este caso los operadores y métodos que encontrábamos en SHOP se definen como funciones en python aunque son pasados a métodos usando hop.declare_methods('enviar', enviar_todo) y a operadores con hop.declare_operators(mover_dron, coger_caja, dejar_caja, entregar_caja).

Por ejemplo el operador mover-dron quedaría de la siguiente forma:

```
def mover_dron(state, dron, locFrom, locTo):
    if state.dron_en[dron] == locFrom:
        state.dron_en[dron] = locTo
        return state
    else: return False
```

En este operador se puede observar cómo se comprueban las condiciones en el if y si se cumplen se realizar el movimiento del dron cambiando el estado el cual se devuelve (nuevo estado en el que dron está en locTo), si no se cumple condición no se realiza la acción y devuelve estado falso terminando la ejecución. De forma similar se hace en el resto de operadores y métodos.

Los métodos enviar-todo y realizar-entrega por su parte quedarían de la siguiente forma:

```
#Metodos
def enviar_todo(state, a):
    clave = list(state.persona_necesita.keys())
    if len(clave) > 0:
        persona = clave[0]
        return [('realizar', persona, state.persona_necesita[persona]), ('enviar', "")]
    elif len(clave) == 0:
        return []
    return False

def realizar_entrega(state, persona, contenido):
    loc = state.persona_en[persona]
    caja = buscar_clave(state.caja_contiene, contenido)
    dron = buscar_clave(state.dron_en, 'deposito')
    brazo = state.dron_tiene['dron1'][0]
    return [('coger_caja', dron, brazo, caja, contenido, 'deposito'),
            ('mover_dron', dron, 'deposito', loc),
            ('entregar_caja', dron, brazo, caja, contenido, loc, persona),
```

```
('mover_dron', dron, loc, 'deposito')]
```

En enviar todo lo primero que se hace es sacar el número de personas que necesitan alguna caja a través del diccionario persona_necesita, si la longitud de este diccionario es mayor que 0 (hay alguna persona que necesita una caja) se llama a realizar-entrega (llamada recursiva), si la longitud es 0(no quedan personas que necesiten una caja) se termina la ejecución devolviendo el plan encontrado y si no se cumple ninguna de los anteriores (longitud menor que 0 lo cual es erróneo) se devuelve false terminando ejecución errónea.

Además, cabe destacar que se han utilizado diccionarios para guardar los valores de los distintos parámetros los cuales creamos haciendo uso de funciones auxiliares como la siguiente la cual crea tantas cajas como valor n le indiquemos:

```
def crear_cajas(n):
    dic = {}
    for i in range(1, n + 1):
        dic['caja' + str(i)] = 'deposito'
    return dic
```

-Generador del problema:

En este caso no se ha llevado a cabo un generador de problemas como tal pero sí que se ha programado dentro del archivo de dominio de forma que se nos permita crear problemas de distinto tamaños cambiando el número de elementos.

Ésto se ha llevado a cabo el siguiente código:

```
#Init
caja = 5
persona = 5
localizacion = 5
contenido = random.randint(0, caja)

state1 = hop.State('state1')
state1.dron_en = {'dron1':'deposito'}
state1.caja_en = crear_cajas(caja)
state1.persona_en = crear_personas(persona, localizacion)

state1.caja_contiene = llenar_caja(caja, contenido)
state1.persona_necesita = crear_necesidad(persona, contenido)

state1.persona_tiene = {}

state1.dron_tiene = {'dron1':['brazo1', 'brazo2']}
state1.libre = {'brazo1':True, 'brazo2':True}
state1.brazo_tiene = {}
```

En este código indicamos primero el número que queremos de cada elemento haciendo uso de las funciones auxiliares como `crear_cajas(cajas)` mencionadas anteriormente.

Preguntas:

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

Las **ventajas** de Pyhop son:

- Más fácil de usar al tener una sintaxis más sencilla y ser una versión en python simplificada de SHOP
- Mayor flexibilidad al permitir definir los operadores de manera intuitiva
- Permite una mayor reutilización del código y escalabilidad al estar organizado en módulos.
- Iguala los parámetros del método y la tarea.
- Las subtareas se producen al ejecutar el código.
- Permite usar lógica sobre el estado actual para determinar la mejor secuencia de subtareas.
- Representa los estados del mundo a través de enlaces de variables ordinarias en vez de proposiciones lógicas.
- Un estado es un objeto de Python.

Los **inconvenientes** son:

- Tiene la limitación de que solo una instancia de método coincide con la tarea.
- Limitación ya que las subtareas no se conocen anticipadamente.
- No se unifican los parámetros de las tareas automáticamente por lo que es necesario pasar todos los parámetros a diferencia de en SHOP2 el cual era capaz de buscar automáticamente el valor de los parámetros según el plan generado.

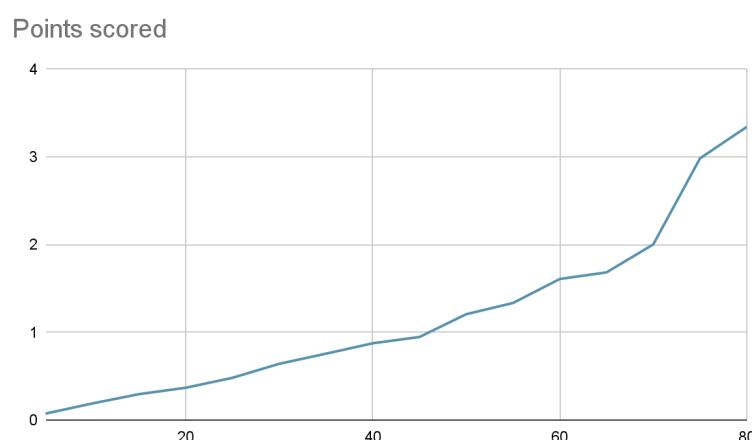
A continuación se observan los tiempos obtenidos tras intentar resolver el mismo problema que en el ejercicio 1.1 pero esta vez usando Pyhop:

-Tabla de tiempos de Pyhop

Tamaño problema	Tiempo	Nº acciones
5	0,07319	20
10	0,18872	40
15	0,29503	60
20	0,36804	80
25	0,48142	100
30	0,64126	120
35	0,75654	140
40	0,87518	160
45	0,94699	180
50	1,20707	200
55	1,334412	220
60	1,60825	240
65	1,68368	260
70	2,00271	280
75	2,9845	300
80	3,34325	320
85	ERROR MAXIMA RECURSION	ERROR MAXIMA RECURSION

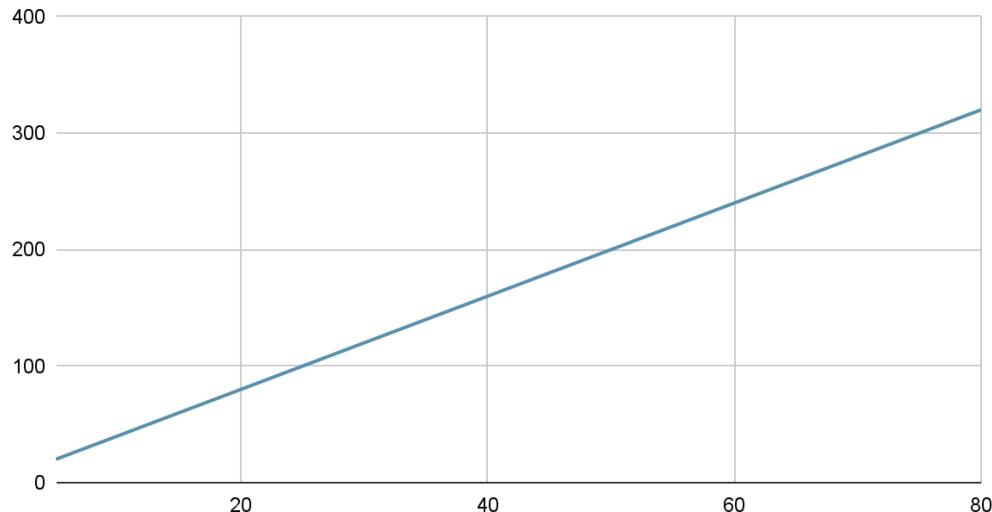
Este error se debe a que python no permite un número tan grande de recursividad pero el planificador podría seguir funcionando si cambiamos el límite de python.

-Gráfica de los tiempos de Pyhop (x → tamaño problema, y→ tiempo)



-Gráfica del número de acciones según el tamaño del problema

Points scored



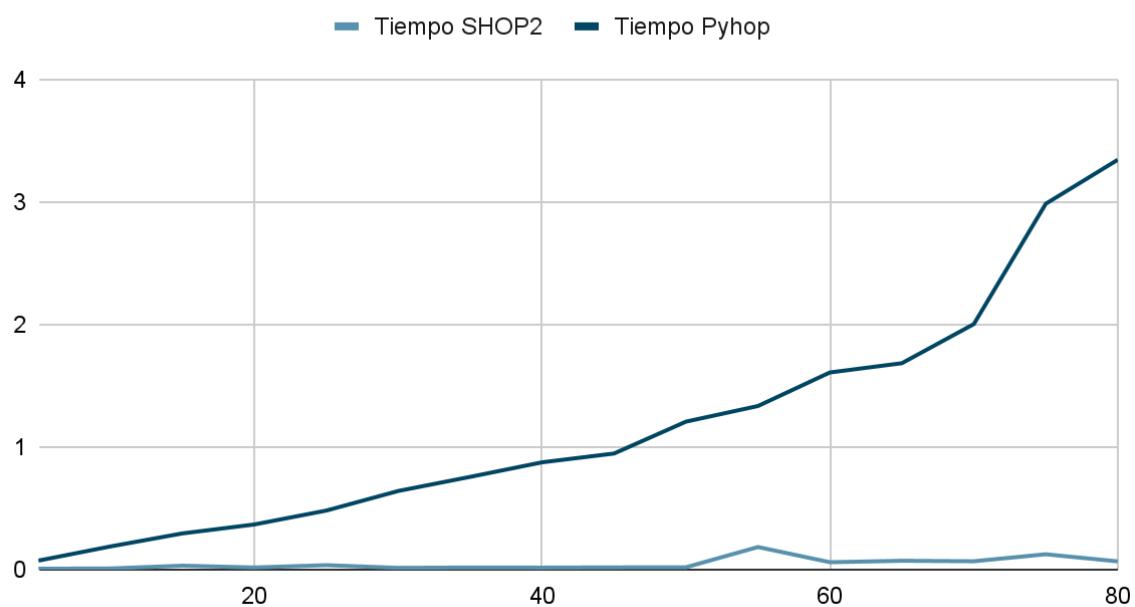
-Tabla comparativa de los tiempos obtenidos al resolver con Pyhop con los obtenidos cuando se usó SHOP2

Tamaño problema	Tiempo SHOP2	Tiempo Pyhop
5	0,007	0,07319
10	0,009	0,18872
15	0,032	0,29503
20	0,017	0,36804
25	0,036	0,48142
30	0,014	0,64126
35	0,017	0,75654
40	0,016	0,87518
45	0,018	0,94699
50	0,019	1,20707
55	0,184	1,334412
60	0,06	1,60825

65	0,072	1,68368
70	0,068	2,00271
75	0,125	2,9845
80	0,067	3,34325

-Gráfica comparativa de los tiempos conseguidos al resolver problemas de distinto tamaño con SHOP2 y Pyhop

Points scored



Tras analizar los datos producidos se observa que Pyhop tarda algo más que SHOP2 al resolver los planes viendo esta diferencia desde el principio cuando el tamaño del problema es 5, cuanto mayor es el tamaño del plan mayor es la diferencia de tiempo aunque éste no es muy grande diferenciándose solo en 3 segundos cuando el tamaño del problema es 80.

Debido a ésto podemos afirmar que el rendimiento de Pyhop es peor al necesitar más tiempo para resolver los problemas esto se puede deber a que en Pyhop no se unifican los parámetros de las tareas automáticamente lo cual puede afectar al backtracking haciendo el proceso más lento tal y como se menciona más adelante en la pregunta 3.

2. Explica los problemas existentes para aplicar planificación automática en videojuegos, los problemas que resuelve Pyhop y los retos pendientes.

Los **problemas** para aplicar la planificación automática en videojuegos son las incompatibilidades producidas entre la planificación automática y los videojuegos en los siguientes aspectos:

-Estado → En este caso un videojuego necesita estructuras de datos lo cual resulta un problema ya que la planificación automática proporciona una serie de proposiciones lo que resulta en una incompatibilidad.

-Acciones → De forma similar al anterior los videojuegos necesitan poder modificar las estructuras de datos y la planificación automática solo permite añadir o borrar proposiciones.

-Agentes → Otro problema es que los videojuegos necesitan poder tener varios agentes y en la planificación automática solo puede haber uno.

-Mundo → El mundo en los videojuegos es dinámico pudiendo tener cambios inesperados, en cambio en la planificación automática es estático manteniéndose igual.

-Tiempo disponible → En los videojuegos se dispone de muy poco tiempo necesitando realizar los cálculos rápidamente mientras que la planificación automática tiene tanto tiempo como el plan necesita.

-Objetivo → El objetivo en los videojuegos es encontrar una solución parcial, en cambio en la planificación automática se quiere encontrar la solución completa al problema.

-Ejecución → En los videojuegos la ejecución se realiza a la vez que la planificación al contrario que en la planificación automática en la que la ejecución empieza tras finalizar la planificación.

Los **problemas que resuelve Pyhop** son la incompatibilidad del estado y de las acciones fácilmente usando en vez de proposiciones lógicas variables de estado para adecuarse a la necesidad de los videojuegos de usar estructuras de datos.

Cómo Pyhop solo resuelve la incompatibilidad del estado y de las acciones el resto de problemas (agentes, mundo, tiempo disponible, objetivo y ejecución) se quedan como **retos pendientes** de resolver no sabiendo si hay una solución general o si tiene que ser específica del juego.

3. Explica cómo afecta esto al backtracking que realiza el planificador respecto a SHOP2.

Esta desventaja afecta al backtracking ya que al no unificar los parámetros de las tareas el planificador debe realizar más comprobaciones en la búsqueda de soluciones para encontrar una asignación válida a estos parámetros lo cual puede provocar que el backtracking sea un proceso más lento requiriendo más tiempo que en el caso de SHOP2.

Desde la perspectiva de la programación el introducir los valores de los parámetros manualmente hace que se requiera más esfuerzo y mayores conocimientos técnicos que para usar SHOP2.

En conclusión, el no poder unificar los parámetros de las tareas automáticamente provoca un peor rendimiento del backtracking lo que puede aumentar el tiempo de cómputo por lo que en este aspecto SHOP2 es más eficiente que Pyhop al unificar automáticamente los parámetros tal y como se puede observar en los resultados de ejecución del ejercicio 1 en el que se aprecia que los tiempos de Pyhop son mayores que con SHOP2.

4. Reflexiona sobre algún dominio de planificación en el que esto pueda resultar especialmente ventajoso.

Esta posibilidad puede resultar bastante ventajosa en cualquier dominio de planificación en el que alguna de sus tareas necesiten ejecutar procesos o programas externos como en el caso del control de las máquinas de una fábrica que necesitan usar un programa de control externo llamado por el planificador.

También puede ser bastante útil si se necesita acceder a datos externos, esto es una gran ventaja porque si se utiliza una gran cantidad de datos éstos pueden estar almacenados en otro sitio y no obligatoriamente en el dominio de planificación lo que beneficia la claridad del dominio y la estructura general. Un ejemplo sería si quieras obtener la mejor ruta en coche a un lugar poder acceder a los datos en tiempo real del tráfico y planificar según éstos, otro ejemplo sería para un sistema de ambulancias una vez se recibe una alarma de emergencias determinar qué ambulancia mandar según su posición actual y el tráfico del camino que seguirían.

Parte 2: Planificación jerárquica avanzada

Ejercicio 2.1 : Logísticas de emergencias avanzada en SHOP2

Explicación del código:

-Dominio de planificación:

Los cambios más importantes realizados en el dominio para cumplir los requisitos del enunciado han sido añadir el uso de los fluents para representar las necesidades de las personas y el número de cajas de cada tipo en una localización de forma que desaparecen las personas y las cajas cómo se puede apreciar en el código de la acción coger-contenido (en ejercicios anteriores se llamaba coger-caja), en este trozo de código se puede observar cómo loc-tiene usa fluents que indique cuántas cajas (indicado por n) con un contenido específico (ct) hay en cierta localización(l), también se puede ver que tras realizar esta acción el número de cajas con ese contenido en la localización disminuye al cogerlo el dron restando uno al fluent n (call - ?n 1).

```
(:operator (!coger-contenido ?d ?ct ?l)
  (
    (DRON ?d)
    (CONTENIDO ?ct)
    (LOCALIZACION ?l)
    (dron-en ?d ?l)
    (loc-tiene ?l ?ct ?n)
    (dron-libre ?d)
  )
  (
    (dron-libre ?d)
    (loc-tiene ?l ?ct ?n)
  )
  (
    (dron-tiene ?d ?ct)
    (loc-tiene ?l ?ct (call - ?n 1))
  )
  0
)
```

Otro cambio que cabe destacar es que se vuelve hacer uso de los transportadores los cuales tienen un límite de capacidad pudiendo haber varios y almacenar distintos tipos de contenido, este límite también se ha llevado a cabo usando fluents.

Al incluir el transportador también se han incluido las acciones mover-transportador, coger-de-transportador, dejar-en-transportador, cargar-transportador. A continuación, se ve un ejemplo de una de estas acciones:

```
(:operator (!coger-de-transportador ?d ?t ?ct ?l)
  (
    (DRON ?d)
    (TRANSPORTADOR ?t)
    (CONTENIDO ?ct)
    (LOCALIZACION ?l)
    (dron-en ?d ?l)
    (transportador-en ?t ?l)
    (transportador-tiene ?t ?ct ?n)
    (capacidad-transportador ?t ?m)
    (dron-libre ?d)
  )
  (
    (dron-libre ?d)
    (transportador-tiene ?t ?ct ?n)
    (capacidad-transportador ?t ?m)
  )
  (
    (dron-tiene ?d ?ct)
    (transportador-tiene ?t ?ct (call - ?n 1))
    (capacidad-transportador ?t (call + ?m 1))
  )
  0
)
```

Como se puede observar en el código se tiene en cuenta el número de cajas con contenido que lleva el transportador en ese momento (transportador-tiene) y la capacidad con la que cuenta(capacidad-transportador) de forma que cuando se saca una caja del transportador aumenta su capacidad pero disminuye la cantidad de contenido que tiene.

También se ha incluido el coste de las acciones tal y como indicaba el enunciado poniendo coste 50 más la capacidad del transportador en la acción mover-transportador como se observa a continuación:

```
(:operator (!mover-transportador ?d ?t ?from ?to)
  (
    (DRON ?d)
    (TRANSPORTADOR ?t)
    (LOCALIZACION ?from)
    (LOCALIZACION ?to)
    (dron-en ?d ?from)
    (transportador-en ?t ?from)
    (limite-transportador ?t ?n)
    (dron-libre ?d)
  )
  (

```

```

        (dron-en ?d ?from)
        (transportador-en ?t ?from)
    )
    (
        (dron-en ?d ?to)
        (transportador-en ?t ?to)
    )
    (call + 50 (call / ?n 2))
)

```

Además, a la acción mover-dron también se le ha puesto coste 50 y al resto de acciones coste 0.

Por último, en el método realizar-entrega se ha tenido en cuenta si el dron solo tiene que entregar una caja o varias, ya que si solo es un contenido lo entrega directamente sin hacer uso del transportador y si hay varias si que lo usa.

```

(:method (realizar-entrega ?l ?ct ?n)
    ;Solo
    (
        (call >= 1 ?n)
        (dron-en ?d ?loc)
    )
    (
        (!coger-contenido ?d ?ct ?loc)
        (!mover-dron ?d ?loc ?l)
        (!entregar-contenido ?d ?ct ?l)
        (!mover-dron ?d ?l ?loc)
    )
    ;No solo
    (
        (dron-en ?d ?loc)
        (transportador-en ?t ?loc)
    )
    (
        (cargar-transportador ?ct ?n ?t)
        (!mover-transportador ?d ?t ?loc ?l)
        (realizar-entrega ?d ?t ?ct ?n)
        (!mover-transportador ?d ?t ?l ?loc)
    )
)
)
```

Además, se han llevado a cabo tres nuevos métodos que son cargar-transportador en el cual se rellena el transportador con un contenido siempre y cuando no se llegue al límite de éste, este método es llamado por cargar en el cual se va cambiando el contenido a cargar y entregar como su nombre indica saca contenido que se necesita en una localización del transportador siempre y cuando quede contenido que sacar y la localización siga necesitandolo. Su código es el siguiente:

```

(:method (cargar-transportador ?ct ?n ?t)
  (
    (loc-tiene ?loc ?ct ?restantes)
    (call > ?restantes 0)
    (capacidad-transportador ?t ?capa)
    (call >= ?capa ?n)
    (call > ?n 0)
    (dron-en ?d ?loc)
  )
  (
    (!coger-contenido ?d ?ct ?loc)
    (!dejar-en-transportador ?d ?t ?ct ?loc)
    (cargar-transportador ?ct (call - ?n 1) ?t)
  )
  (call >= 0 ?n)
  ()
)
)

(:method (cargar ?t ?l)
  (
    (dron-en ?d ?loc)
    (loc-necesita ?l ?ct ?n)
    (call > ?n 0)
    (capacidad-transportador ?t ?capa)
    (call > ?capa 0)
    (transportador-tiene ?t ?ct ?m)
    (call > ?n ?m)
  )
  (
    (cargar-transportador ?ct ?n ?t)
    (cargar ?t ?l)
  )
  ()
  ()
)
)

(:method (entregar ?t)
  (
    (dron-en ?d ?l)
    (loc-necesita ?l ?ct ?n)
    (call > ?n 0)
    (transportador-tiene ?t ?ct ?m)
    (call > ?m 0)
  )
  (

```

```

        (realizar-entrega ?d ?t ?ct ?m)
        (entregar ?t)
    )
()

()
)
```

-Generador del problema:

En el caso del generador del problema se ha eliminado la generación de las personas y de las cajas y se ha añadido que en el comando se puedan introducir distintos valores para el límite de cada transportador y el número de cajas que se encuentran en cada localización teniendo tantos transportadores y localizaciones como números separados por comas se introduzcan.

Un ejemplo del comando es: python3 generate-problem -d 1 -r "20,50,100" -l "10,20,50" -p 5 -c 5 -g 5 siendo -r los transportadores, -l las localizaciones y el resto de parámetros no serían usados (su valor es irrelevante en este caso).

Para conseguir esto se ha cambiado el código al principio del main en el que se indica qué tipos de datos se recogen del comando, por ejemplo en el caso de los transportadores se ha modificado el valor de type de int a string(str) de forma que se puedan introducir varios valores separador por comas como se observa a continuación:

```
parser.add_option('-r', '--carriers', metavar='NUM', type=str, dest='carriers',
                  help='the number of carriers, for later labs; use 0 for no carriers')
```

Tras ésto se ha creado una nueva variable que contenga los valores introducidos (límite de cada transportador) separando la cadena introducida por las comas:

```
lim_transportador = options.carriers.split(', ')
```

Esta lista se ha utilizado para crear los distintos transportadores del problema (habrá tantos como valores halla en la lista):

```
for x in range(len(lim_transportador)):
    carrier.append("transportador" + str(x + 1))
```

Se definen los predicados de la capacidad del transportador, su límite, contenido que tiene etc

```
for i in range(len(lim_transportador)):
    f.write("\t(límite-transportador transportador" + str(i + 1) + " " + lim_transportador[i] +
")\n")
```

```
for i in range(len(lim_transportador)):
```

```
f.write("\t(capacidad-transportador transportador" + str(i + 1) + " " +
lim_transportador[i] + ")\n")

for i in range(len(lim_transportador)):
    for j in range(2):
        f.write("\t(transportador-tiene transportador" + str(i + 1) + " " + content_types[j] + "
0)\n")
    f.write("\n")
```

Lo explicado anteriormente se ha realizado de forma similar para la cantidad de contenido que se encuentra en cada localización aunque como el tipo de contenido debe ser aleatorio se ha realizado el siguiente código para asegurarnos que se crea el contenido aleatorio indicado para cada localización:

```
sum_contenido = [0, 0]
```

```
for i in range(len(cajas_loc)):  
    comida = random.randrange(int(lim_transportador[i]))  
    f.write("\t(loc-necesita loc" + str(i + 1) + " comida " + str(comida) + ")\n")  
    medicina = int(lim_transportador[i]) - comida  
    f.write("\t(loc-necesita loc" + str(i + 1) + " medicina " + str(medicina) + ")\n")  
    sum_contenido[0] += comida  
    sum_contenido[1] += medicina  
f.write("\n")  
  
for i in range(2):  
    f.write("\t(loc-tiene deposito " + content_types[i] + " " + str(sum_contenido[i]) + ")\n")  
f.write("\n")
```

Demostrar el funcionamiento al menos bajo las siguientes circunstancias:

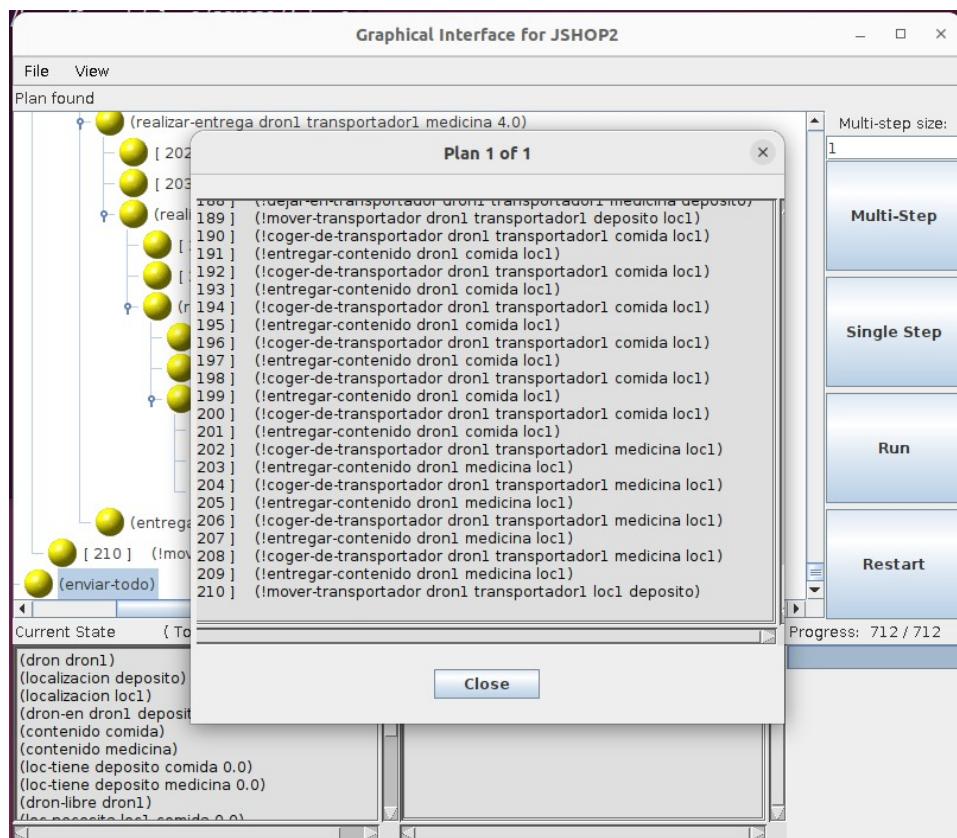
- Un lugar necesita 50 cajas y solo hay un transportador para 10 cajas.

-Salida por pantalla:

```

(!dejar-en-transportador droni transportador1 comida deposito)
(!coger-contenido droni comida deposito)
(!dejar-en-transportador droni transportador1 comida deposito)
(!coger-contenido droni comida deposito)
(!dejar-en-transportador droni transportador1 comida deposito)
(!coger-contenido droni comida deposito)
(!dejar-en-transportador droni transportador1 comida deposito)
(!coger-contenido droni comida deposito)
(!dejar-en-transportador droni transportador1 comida deposito)
(!coger-contenido droni medicina deposito)
(!dejar-en-transportador droni transportador1 medicina deposito)
(!coger-contenido droni medicina deposito)
(!dejar-en-transportador droni transportador1 medicina deposito)
(!coger-contenido droni medicina deposito)
(!dejar-en-transportador droni transportador1 medicina deposito)
(!coger-contenido droni medicina deposito)
(!dejar-en-transportador droni transportador1 medicina deposito)
(!coger-contenido droni medicina deposito)
(!mover-transportador droni transportador1 deposito loc1)
(!coger-de-transportador droni transportador1 comida loc1)
(!entregar-contenido droni comida loc1)
(!coger-de-transportador droni transportador1 comida loc1)
(!entregar-contenido droni comida loc1)
(!coger-de-transportador droni transportador1 comida loc1)
(!entregar-contenido droni comida loc1)
(!coger-de-transportador droni transportador1 comida loc1)
(!entregar-contenido droni comida loc1)
(!coger-de-transportador droni transportador1 comida loc1)
(!entregar-contenido droni comida loc1)
(!coger-de-transportador droni transportador1 medicina loc1)
(!entregar-contenido droni medicina loc1)
(!coger-de-transportador droni transportador1 medicina loc1)
(!entregar-contenido droni medicina loc1)
(!coger-de-transportador droni transportador1 medicina loc1)
(!entregar-contenido droni medicina loc1)
(!coger-de-transportador droni transportador1 medicina loc1)
(!entregar-contenido droni medicina loc1)
(!mover-transportador droni transportador1 loc1 deposito)
.....

```



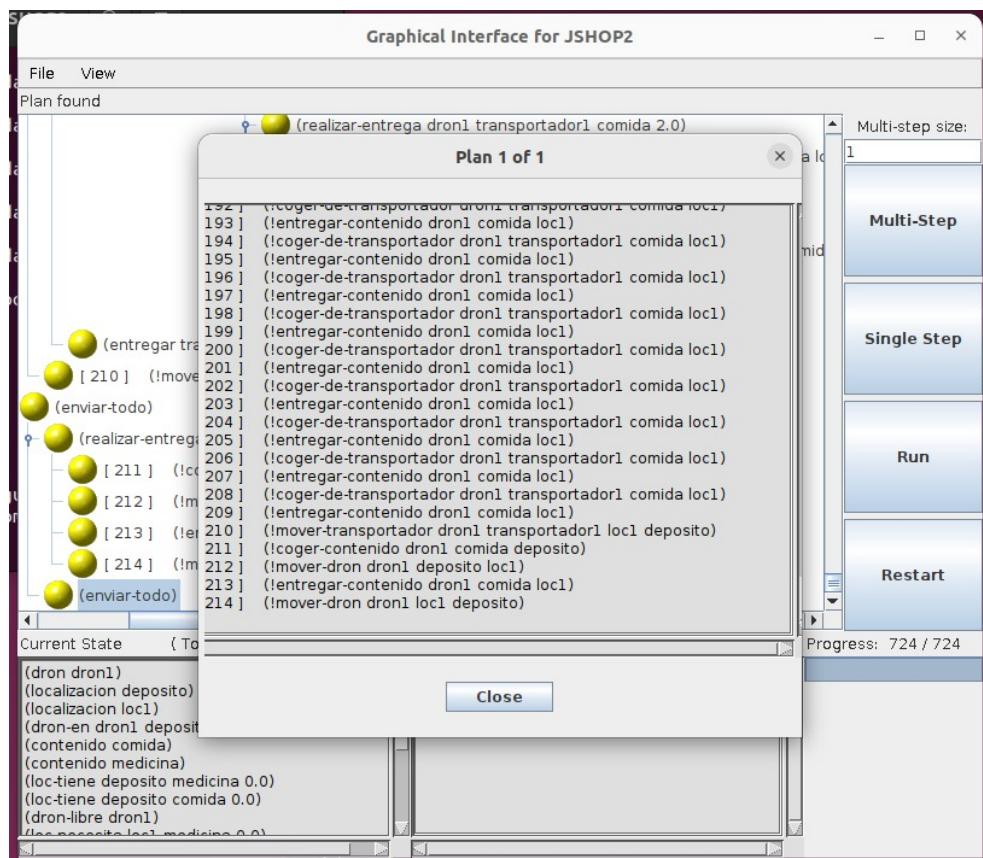
-Nº acciones: 210

-Tiempo usado: 0,064

-Coste del plan: 550

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

-Salida por pantalla:



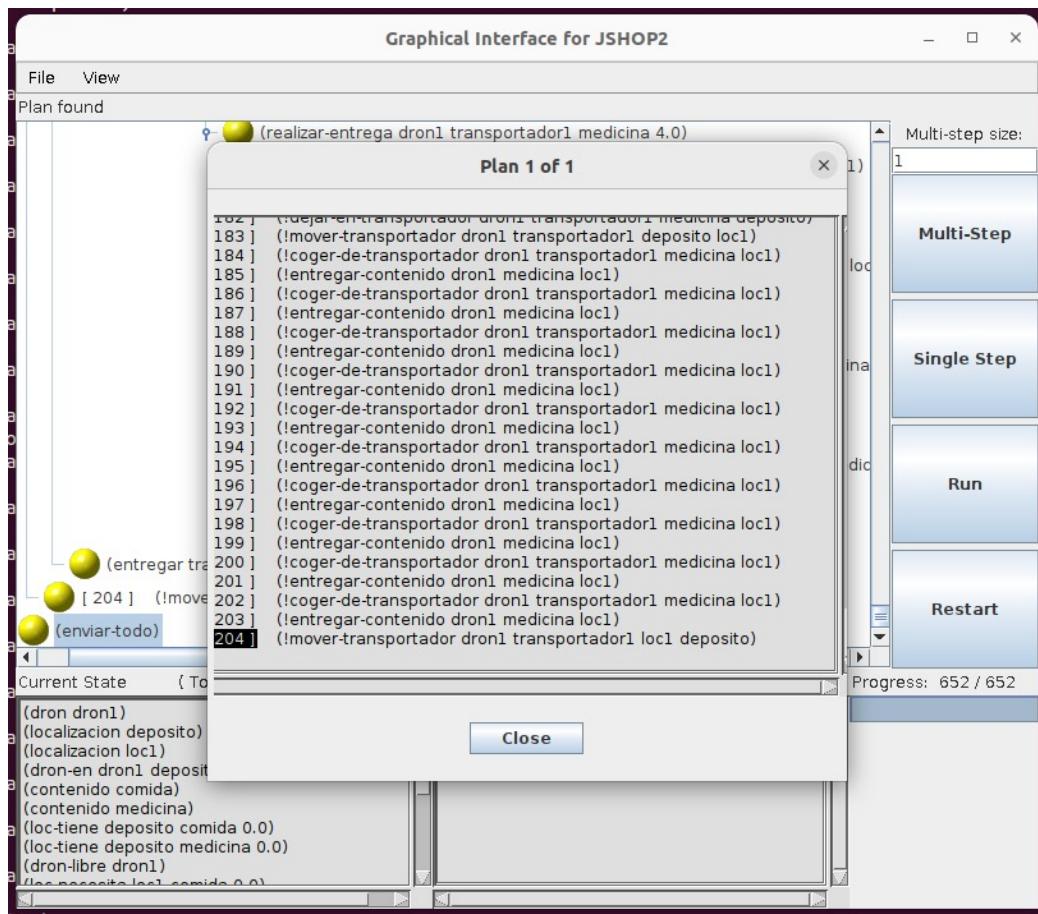
-N^a Acciones: 214

-Tiempo usado: 0,024

-Coste del plan: 650

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

-Salida por pantalla:



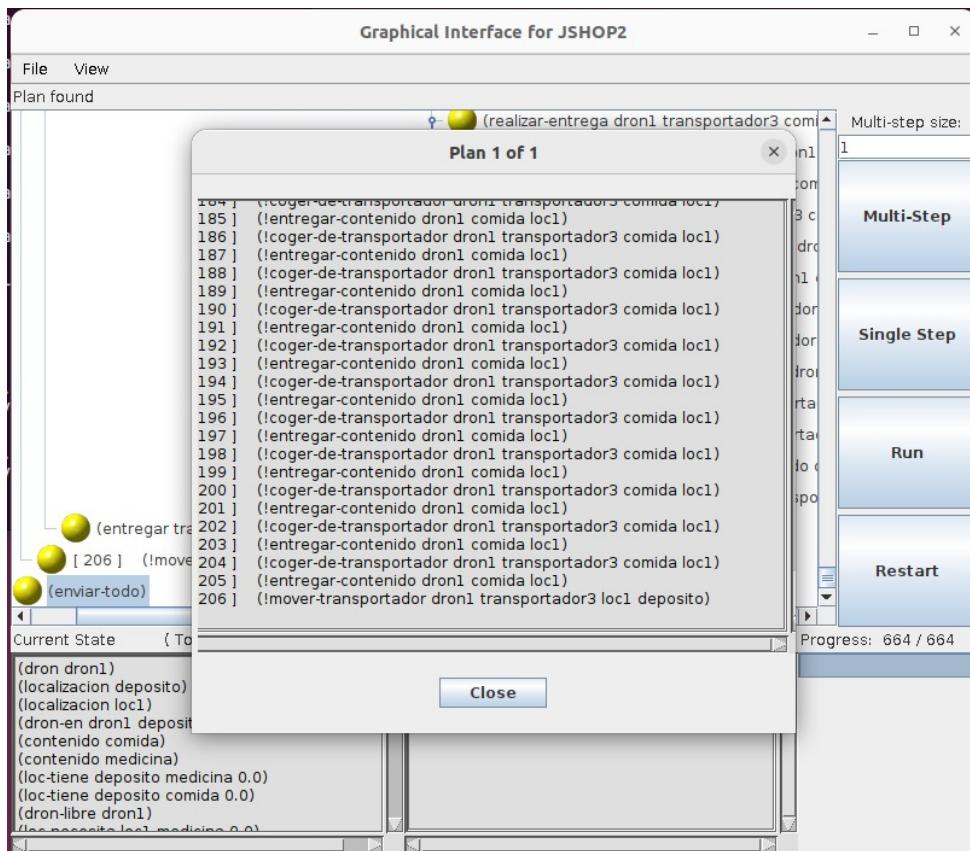
-Nº Acciones: 204

-Tiempo usado: 0,063

-Coste del plan: 280

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

-Salida por pantalla:



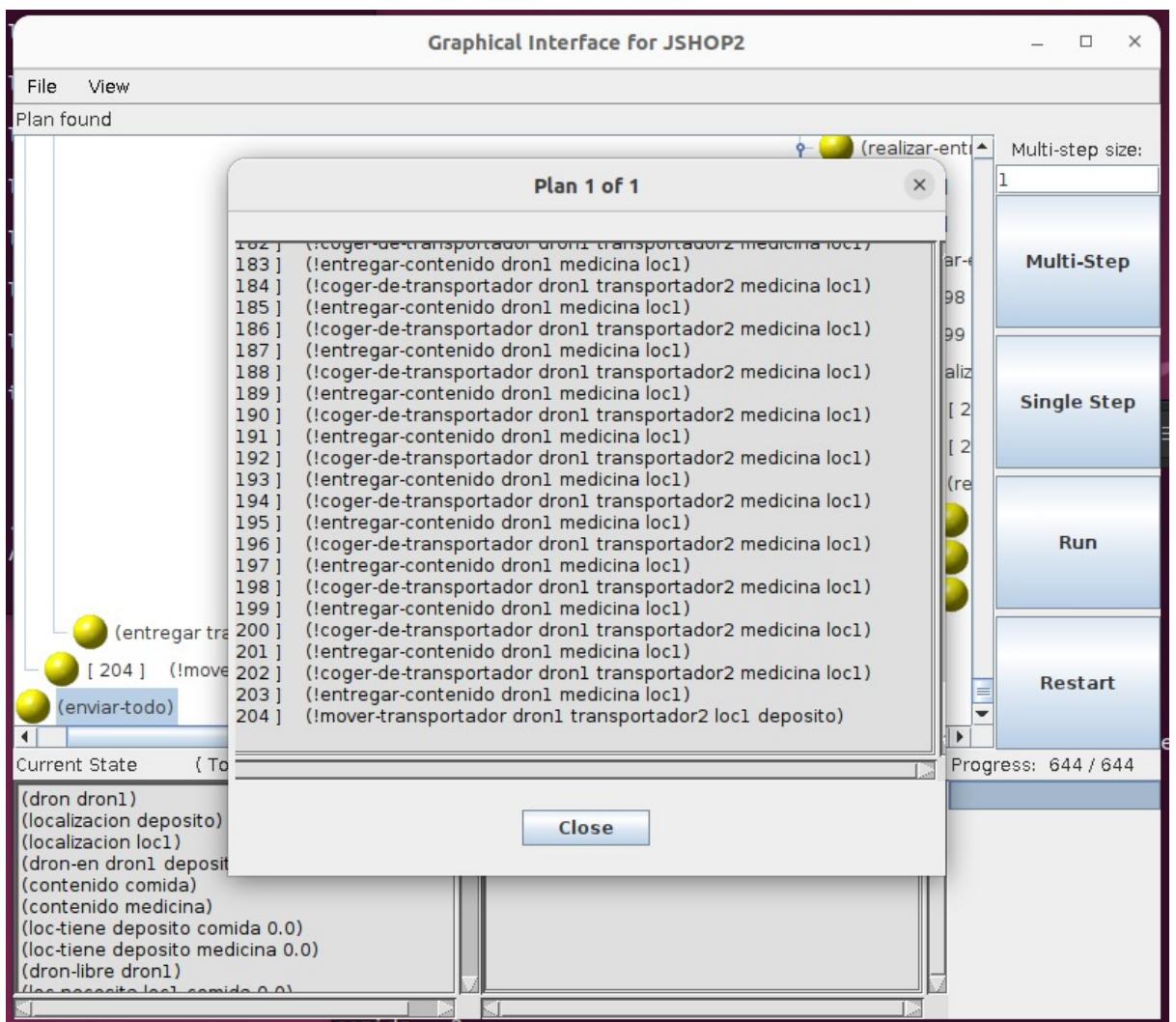
-Nº Acciones: 206

-Tiempo usado: 0,063

-Coste del plan: 360

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

-Salida por pantalla:



-Nº Acciones: 204

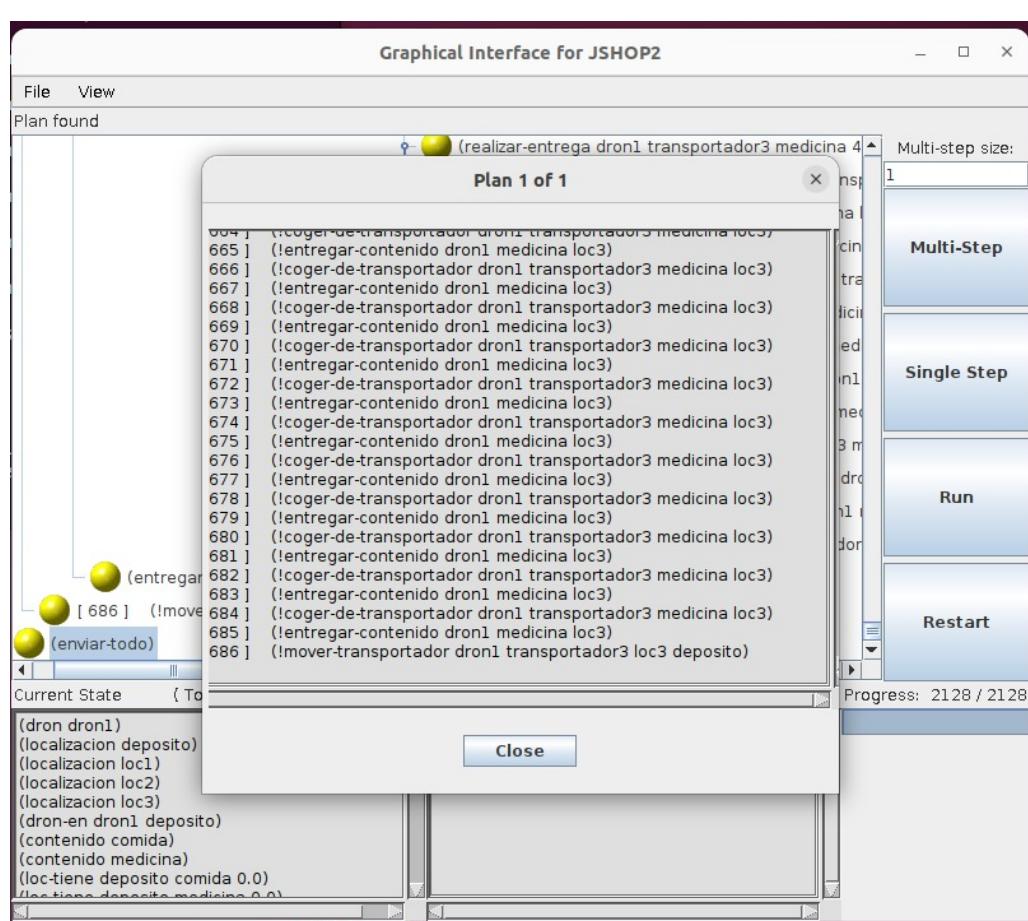
-Tiempo usado: 0,064

-Coste del plan: 270

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

-Salida por pantalla:

Time Used = 0.054



-NºAcciones: 686

-Tiempo usado: 0,054

-Coste del plan: 470

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

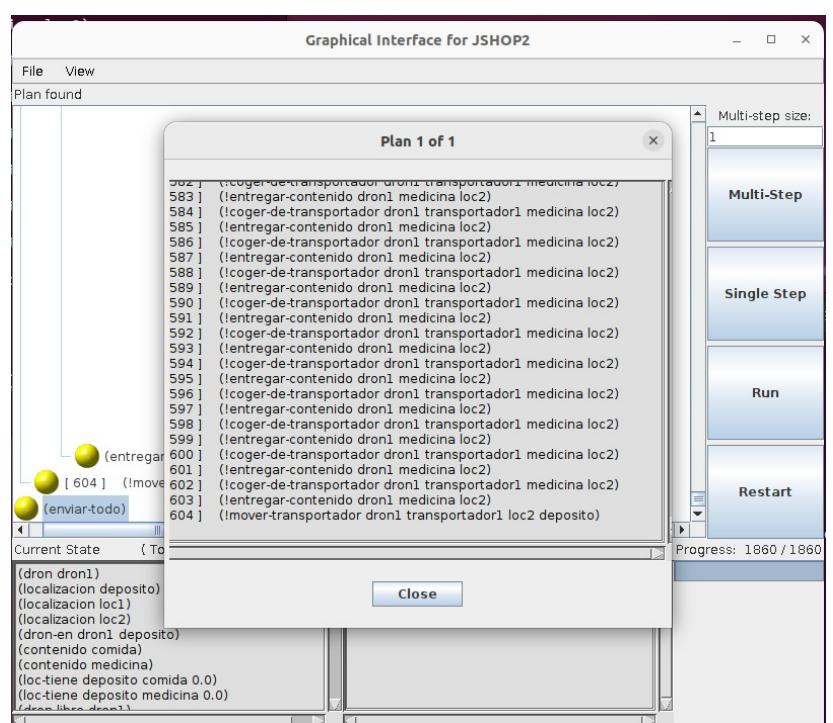
-Salida por pantalla:

Time Used = 0.045

-Nº Acciones: 604

-Tiempo usado: 0,045

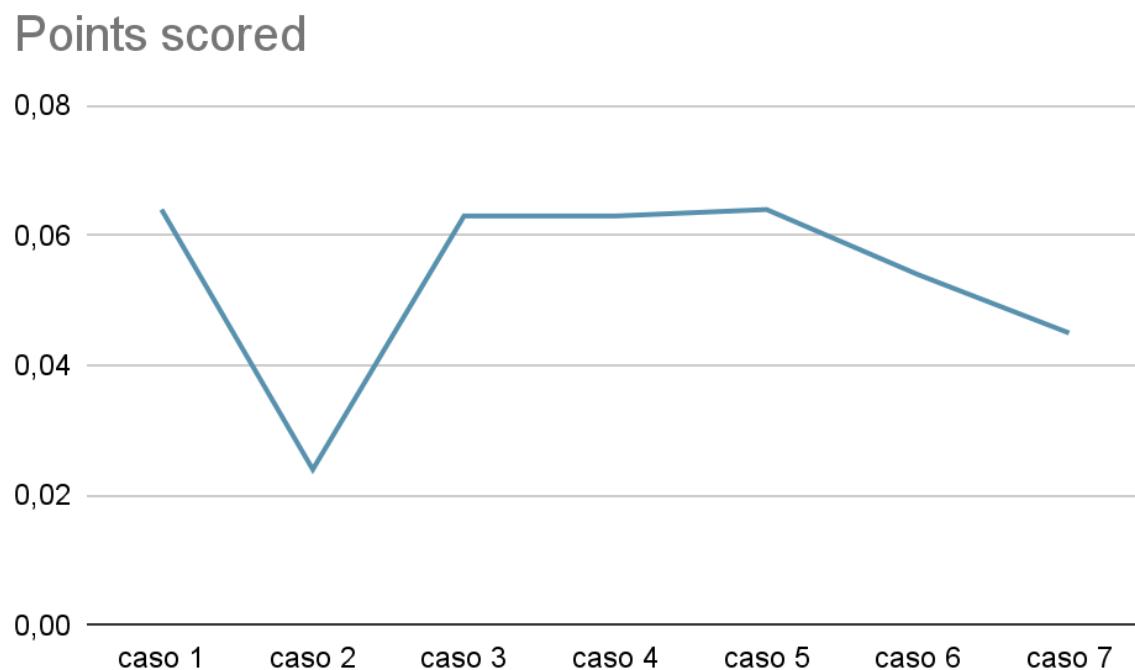
-Coste del plan: 500



-Tabla comparativa de los resultados obtenidos para cada combinación de valores anterior:

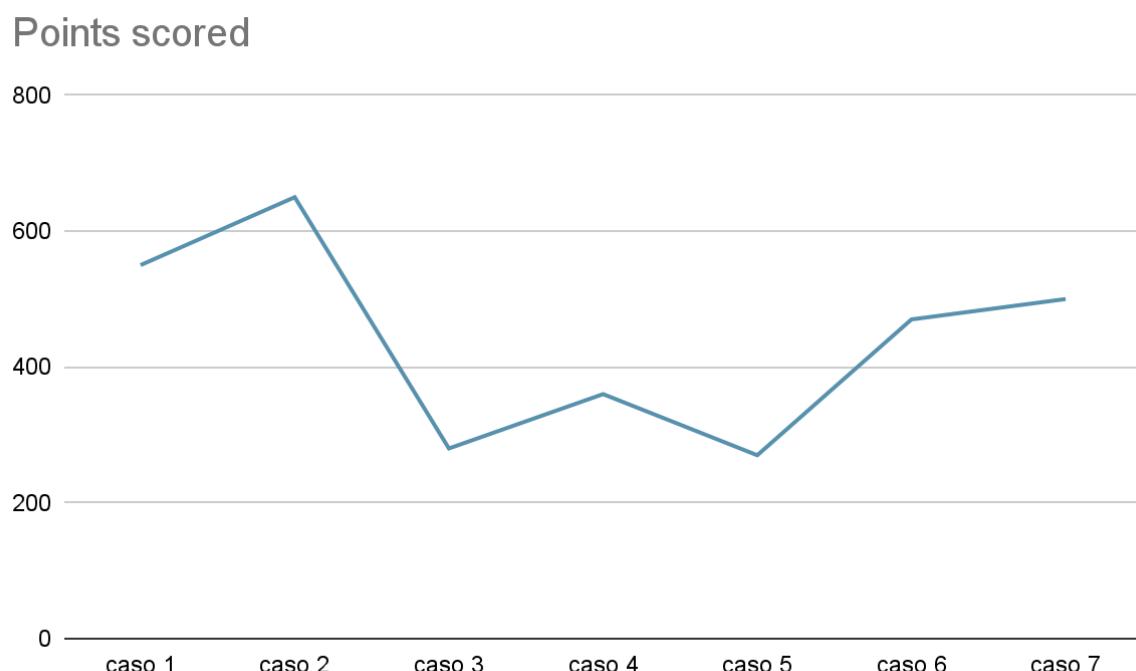
NºTransportadores	Límite transportadores (en orden)	Nº lugares que necesitan caja	Nº cajas necesita cada lugar (en orden)	Coste del plan	Tiempo	Nº Acciones
1	10	1	50	550	0,064	210
1	10	1	51	650	0,024	214
1	40	1	50	280	0,063	204
3	10, 20, 30	1	50	360	0,063	206
3	20, 50, 100	1	50	270	0,064	204
3	20, 50, 100	3	20, 50, 100	470	0,054	686
1	150	2	50, 100	500	0,045	604

-Gráfica en la que se muestra el tiempo para cada caso anterior (x son los casos numerados e y los tiempos):



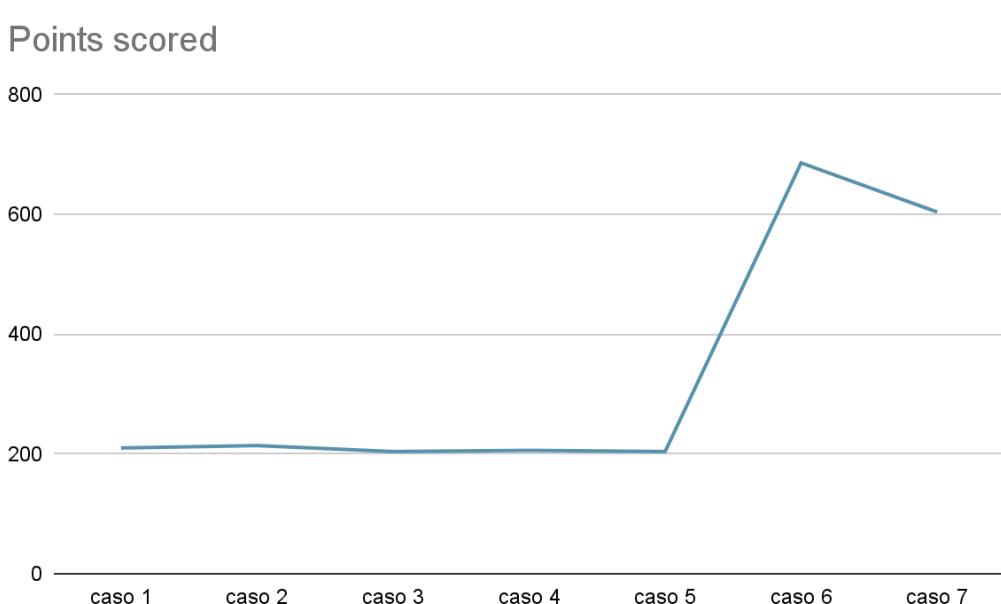
Se observa que el tiempo de ejecución es muy pequeño no pasando de los 0.08 segundos y que no hay una relación clara entre el tamaño del problema y el tiempo necesario ya que como vemos en el caso 2 (localización necesita 51 cajas y un transportador de límite 10) el tiempo disminuye respecto al caso 1 (localización necesita 50 cajas y límite de transportador es 10) a pesar de tener un tamaño mayor de problema aunque como se ve en los siguientes gráficos el coste y el número de acciones si que aumenta al aumentar el tamaño del problema en este caso.

-Gráfica en la que se muestra el coste para cada caso anterior (x son los caso numerados e y los costes):



Se observa que cuando se aumenta el límite del transportador disminuye el coste al tener que hacer menos movimientos para entregar el contenido.

-Gráfica en la que se muestra el número de acciones para cada caso anterior (x son los caso numerados e y el número de acciones):



En este caso el número de acciones es más o menos lineal hasta que se aumenta el número de transportadores y las necesidades de las localizaciones.

Ejercicio 2.2: Optimización del planificador

Explicación del código:

-Dominio de planificación:

Para llevar a cabo las optimizaciones pedidas en el enunciado se ha cambiado el dominio del ejercicio anterior modificando el método enviar-todo de forma se atienda primero a las localizaciones que necesite más contenidos, esto se determina con la línea:

```
:sort-by ?n > (and (loc-total-necesita ?l ?n) (call > ?n 0))
```

Siendo el predicado loc-total-necesita un nuevo predicado añadido que indica el contenido total que una localización necesita.

También se ha modificado el método realizar-entrega añadiendo las condiciones necesarias para que se escoja el transportador más adecuado según su capacidad y contenido a entregar comprobando si el límite del transportador es igual al contenido a entregar o si es mayor o menor (que sea menor es la última opción ya que no es la óptima, lo idóneo sería que el límite fuese igual a la cantidad a entregar).

```
(:method (realizar-entrega ?l ?n)
  ;Solo
  (
    (dron-en ?d ?loc)
    (call = 1 ?n)
    (loc-necesita ?l ?ct ?m)
    (call > ?m 0)
  )
  (
    (!coger-contenido ?d ?ct ?loc)
    (!mover-dron ?d ?loc ?l)
    (!entregar-contenido ?d ?ct ?l)
    (!mover-dron ?d ?l ?loc)
  )

  ;No solo
  ;Igual
  (
    :sort-by ?lim > (and (dron-en ?d ?loc) (transportador-en ?t ?loc)
    (limite-transportador ?t ?lim) (call = ?lim ?n))
  )
  (
    (cargar ?t ?l)
    (!mover-transportador ?d ?t ?loc ?l)
    (entregar ?t)
    (!mover-transportador ?d ?t ?l ?loc)
```

```

)
;Mayor
(
  :sort-by ?lim > (and (dron-en ?d ?loc) (transportador-en ?t ?loc)
(limite-transportador ?t ?lim) (call > ?lim ?n))
)
(
  (cargar ?t ?l)
  (!mover-transportador ?d ?t ?loc ?l)
  (entregar ?t)
  (!mover-transportador ?d ?t ?l ?loc)
)
;Menor
(
  :sort-by ?lim > (and (dron-en ?d ?loc) (transportador-en ?t ?loc)
(limite-transportador ?t ?lim))
)
(
  (cargar ?t ?l)
  (!mover-transportador ?d ?t ?loc ?l)
  (entregar ?t)
  (!mover-transportador ?d ?t ?l ?loc)
)
)

```

-Generador del problema:

En el caso del generador solo se ha cambiado de forma que introduzca el nuevo predicado explicado anteriormente (loc-total-necesita).

```

for i in range(len(cajas_loc)):
    f.write("\t(loc-total-necesita loc" + str(i + 1) + " " + cajas_loc[i] + ")\n")

```

Explicar cómo las modificaciones realizadas mejoran los resultados obtenidos por el planificador en los casos planteados en el ejercicio anterior.

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

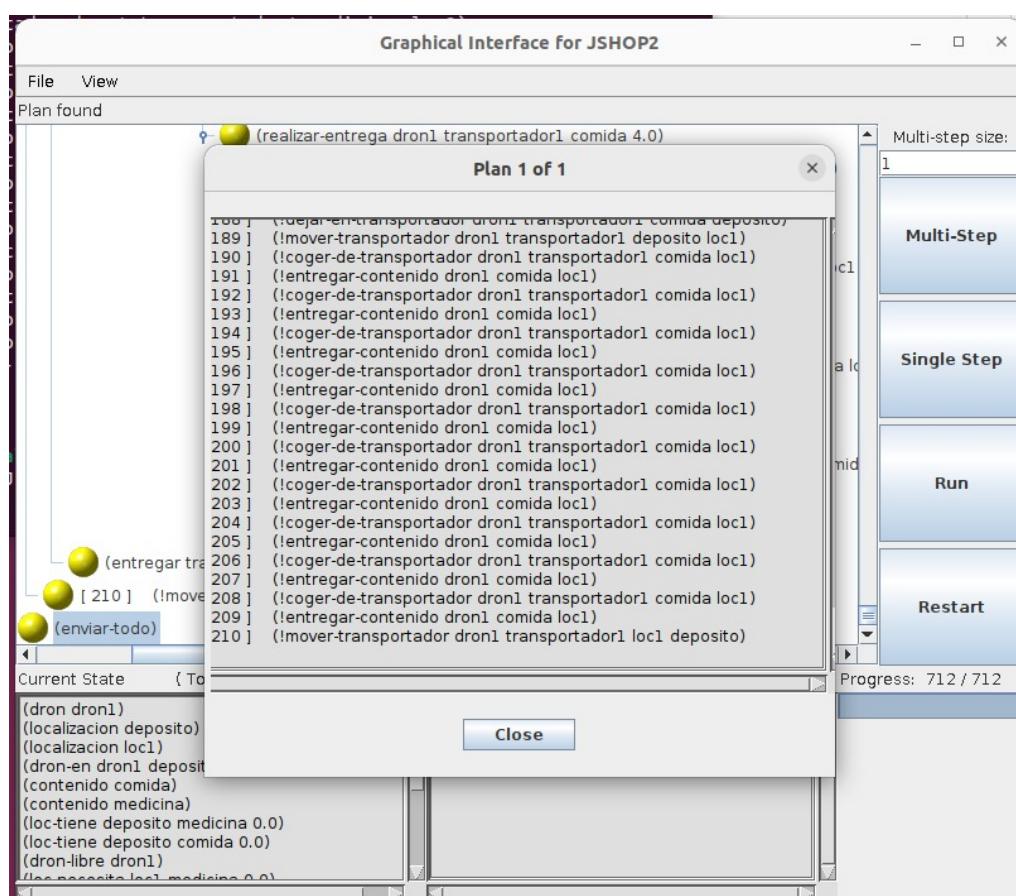
-Salida por pantalla:

```

(!dejar-en-transportador dron1 transportador1 comida deposito)
(!coger-contenido dron1 comida deposito)
(!dejar-en-transportador dron1 transportador1 comida deposito)
(!coger-contenido dron1 comida deposito)
(!dejar-en-transportador dron1 transportador1 comida deposito)
(!coger-contenido dron1 comida deposito)
(!dejar-en-transportador dron1 transportador1 comida deposito)
(!mover-transportador dron1 transportador1 deposito loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!mover-transportador dron1 transportador1 loc1 deposito)
-----

```

Time Used = 0.034



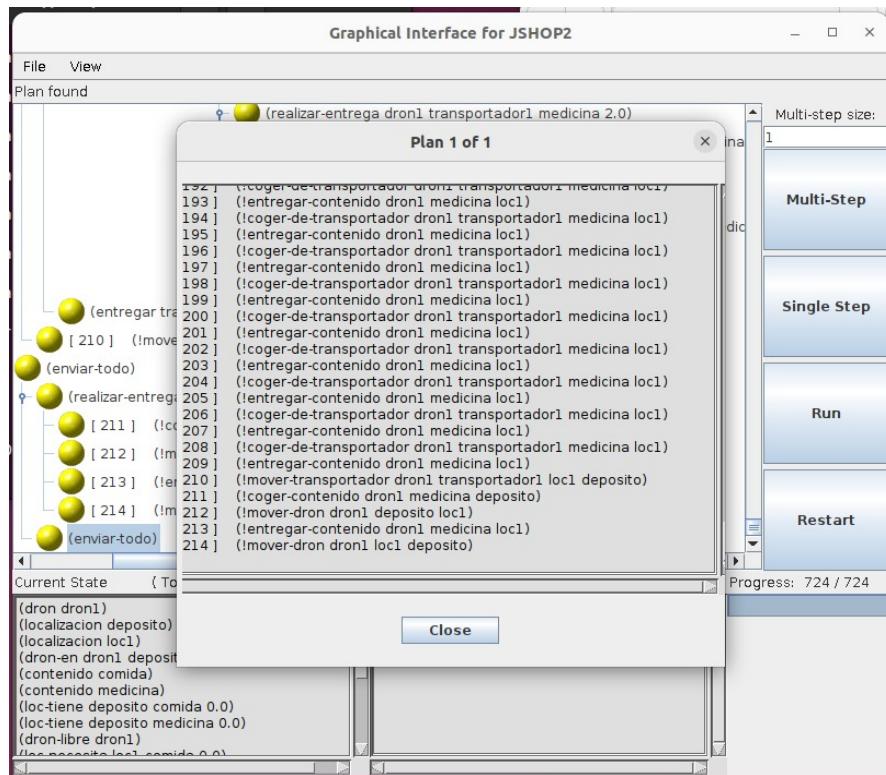
-Nº Acciones: 210

-Tiempo usado: 0,034

-Coste del plan: 550

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

-Salida por pantalla:



-NºAcciones: 214

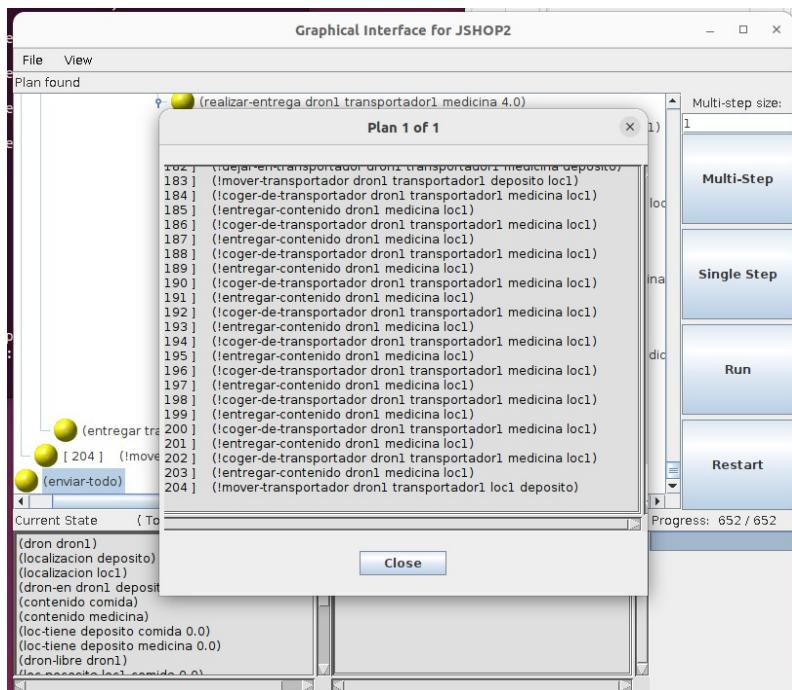
-Tiempo usado: 0,04

-Coste del plan: 650

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

-Salida por pantalla:

```
(!coger-contenido dron1 medicina deposito)
(!dejar-en-transportador dron1 transportador1 medicina deposito)
(!coger-contenido dron1 medicina deposito)
(!dejar-en-transportador dron1 transportador1 medicina deposito)
(!coger-contenido dron1 medicina deposito)
(!dejar-en-transportador dron1 transportador1 medicina deposito)
(!mover-transportador dron1 transportador1 deposito loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!mover-transportador dron1 transportador1 loc1 deposito)
-----
Time Used = 0.027
```



-Nº acciones: 204

-Tiempo usado: 0,027

-Coste del plan: 280

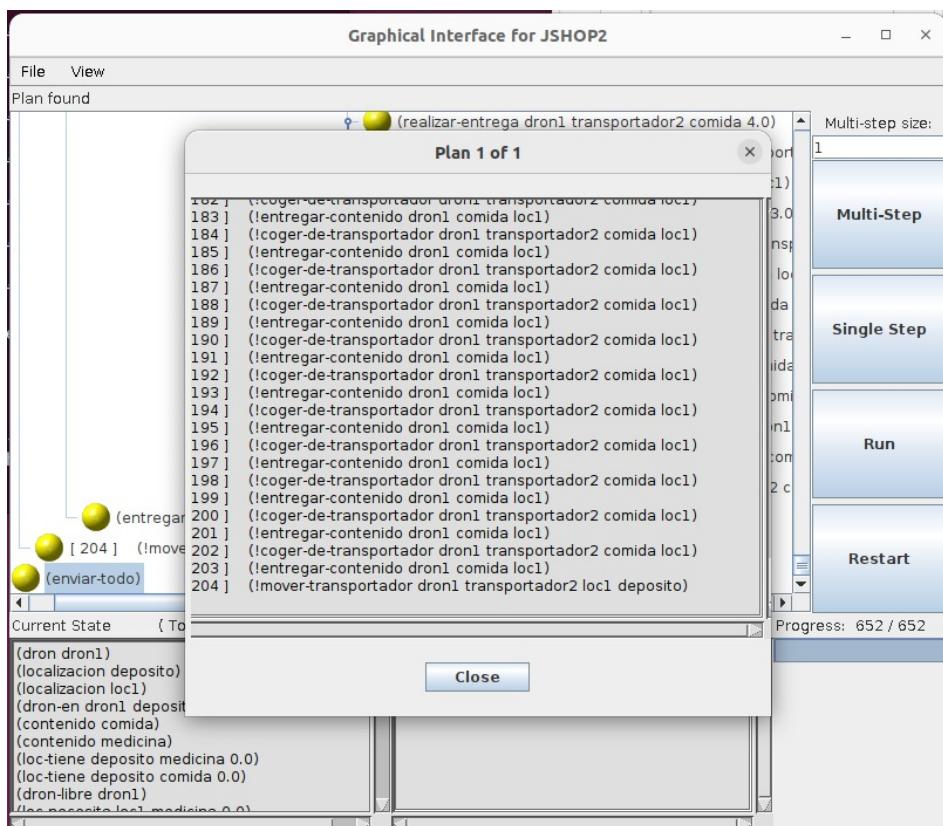
- Un lugar necesita 50 cajas y hay transportadores de 10, 20 y 30 cajas.

-Salida por pantalla:

```

(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador2 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!mover-transportador dron1 transportador2 loc1 deposito)
-----
```

Time Used = 0.068



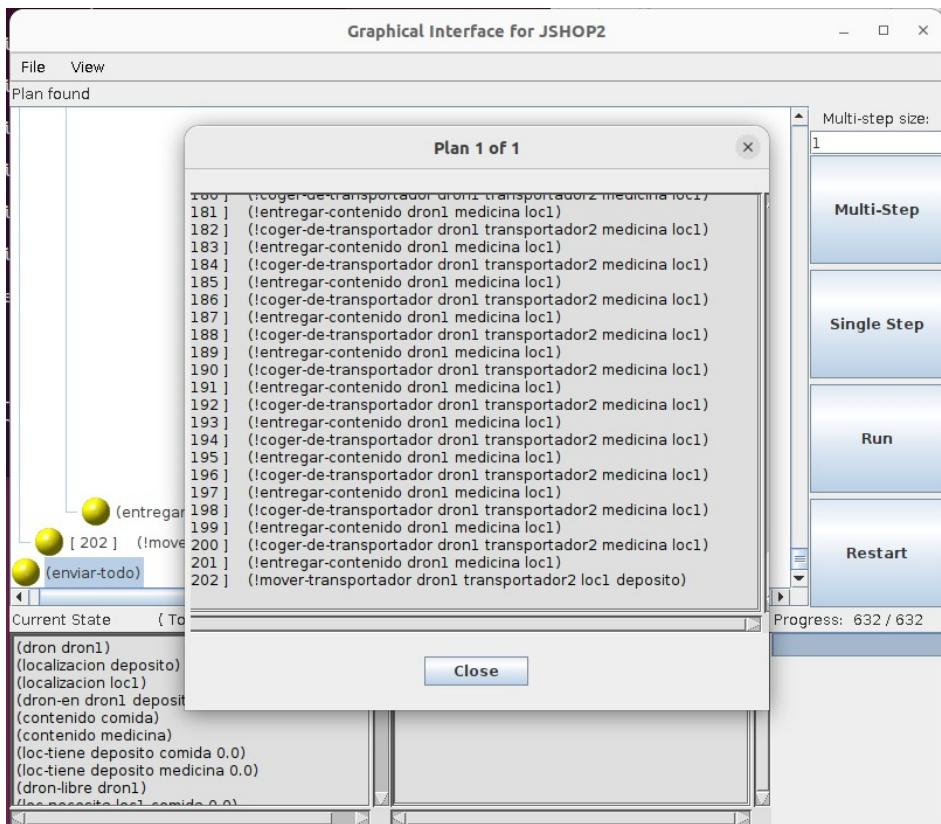
-Nº acciones: 204

-Tiempo usado: 0,068

-Coste del plan: 250

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

-Salida por pantalla:



-Nº Acciones: 202

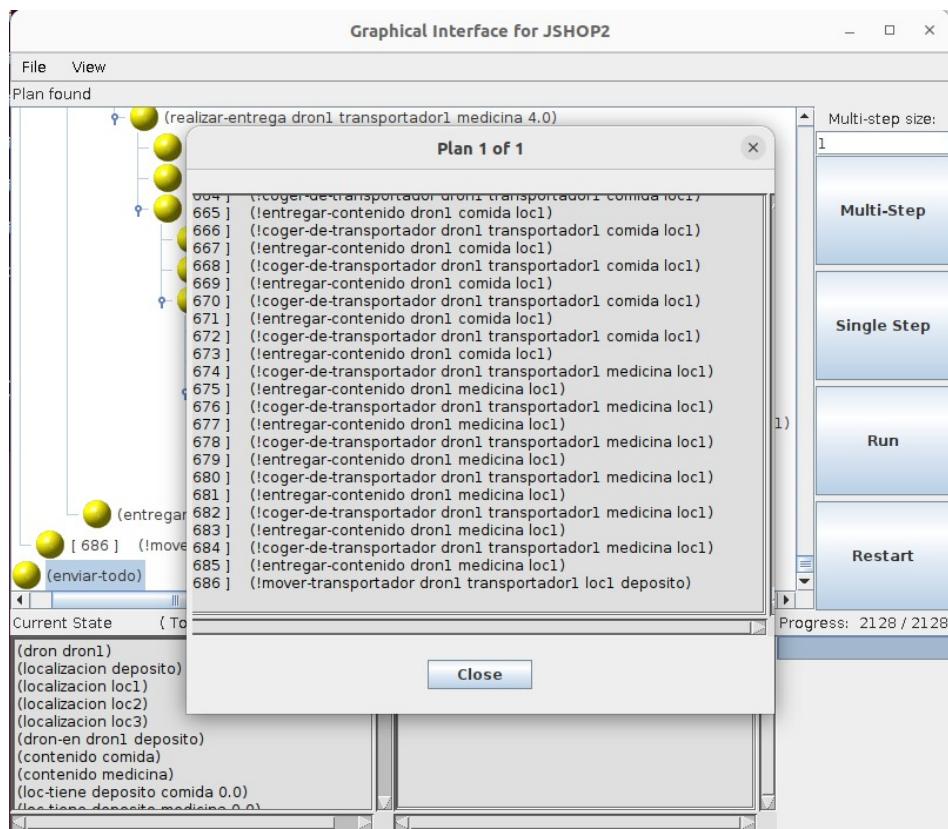
-Tiempo usado: 0,028

-Coste del plan: 150

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

-Salida por pantalla:

```
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 comida loc1)
(!entregar-contenido dron1 comida loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!coger-de-transportador dron1 transportador1 medicina loc1)
(!entregar-contenido dron1 medicina loc1)
(!mover-transportador dron1 transportador1 loc1 deposito)
```



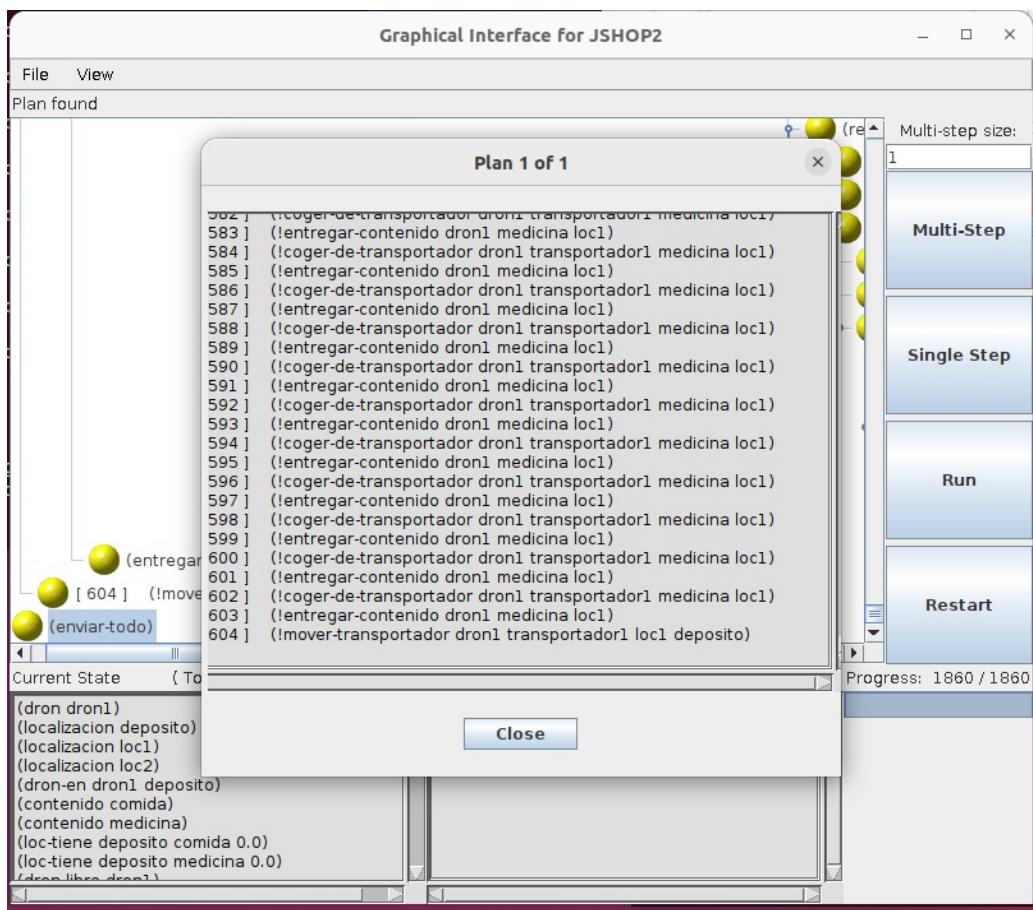
-Nº Acciones: 686

-Tiempo usado: 0,107

-Coste del plan: 470

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

-Salida por pantalla:



-Nº acciones: 604

-Tiempo usado: 0,068

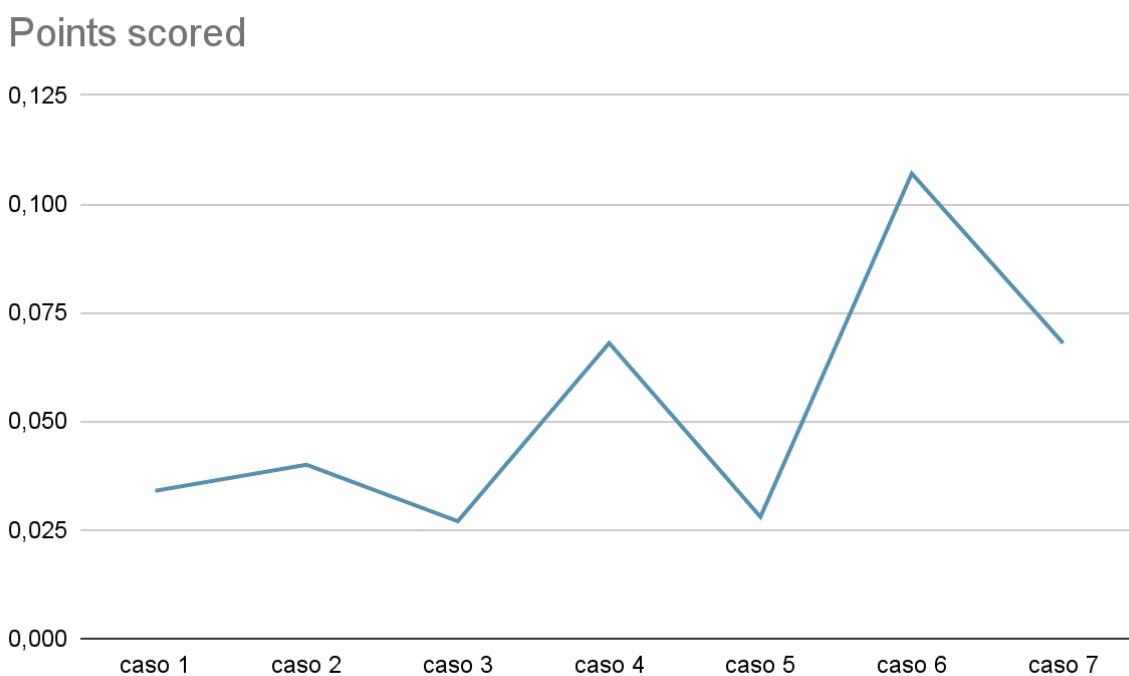
-Coste del plan: 500

-Tabla comparativa de los resultados obtenidos para cada combinación de valores anterior tras aplicar las mejoras:

NºTransportadores	Límite transportadores (en orden)	Nº lugares que necesitan caja	Nº cajas necesita cada lugar (en orden)	Coste del plan	Tiempo	Nº acciones
1	10	1	50	550	0,034	210
1	10	1	51	650	0,04	214
1	40	1	50	280	0,027	204
3	10, 20, 30	1	50	250	0,068	204

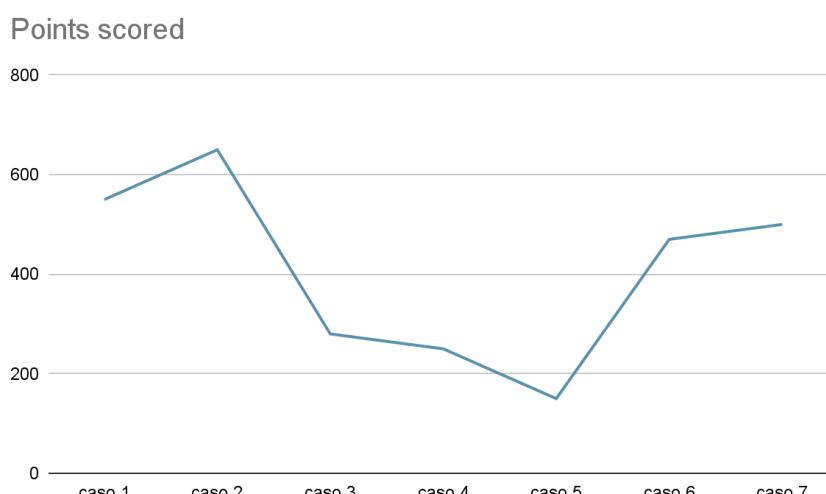
3	20, 50, 100	1	50	150	0,028	202
3	20, 50, 100	3	20, 50, 100	470	0,107	686
1	150	2	50, 100	500	0,068	604

-Gráfica en la que se muestra el tiempo para cada caso anterior(x son el número del caso e y el tiempo):



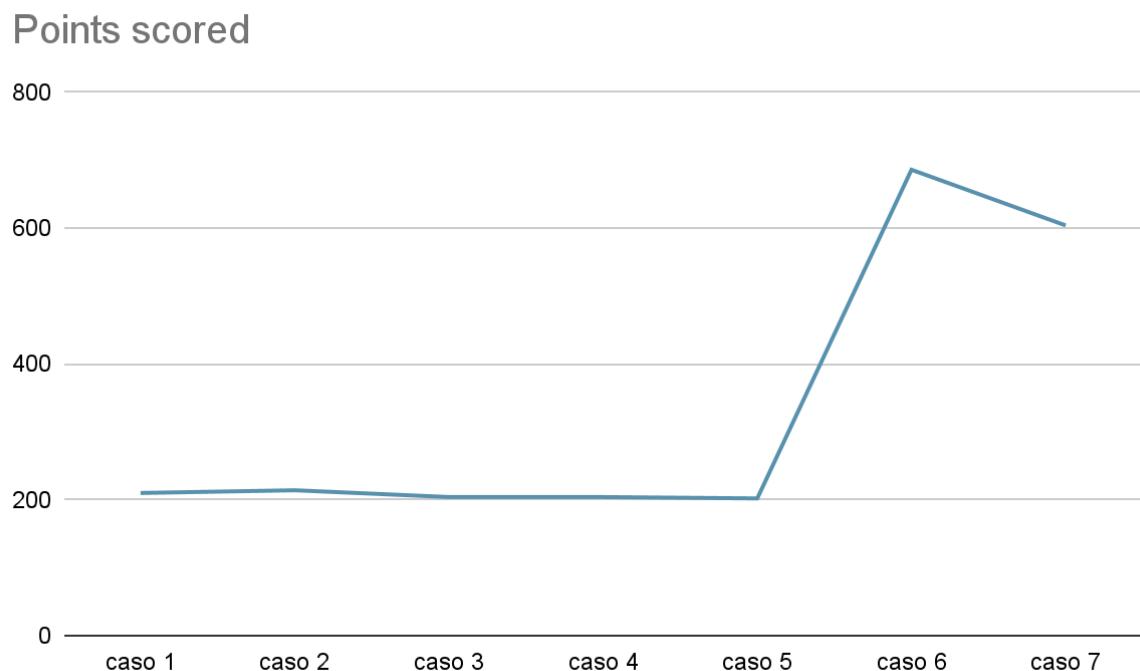
Esta vez sí que se aprecia que aumenta el tiempo en el caso 2 cuando aumenta el número de cajas necesarias. También se observan que hay picos de tiempo pasando del segundo en el caso 6 debido a que el dron siempre tiene que volver al depósito en cada entrega además de que como no se explica el enunciado se ha definido que las cajas se entregan de una en una una vez se encuentra el transportador en la localización deseada.

-Gráfica en la que se muestra el coste para cada caso anterior(x son el número del caso e y el coste):



Al igual que en el ejercicio anterior el coste disminuye cuando aumenta la capacidad del transportador al tener que realizar menos movimientos al poder llevar más cajas a la vez.

-Gráfica en la que se muestra el número de acciones para cada caso anterior(x son el número del caso e y el nº acciones):



Igual que en el ejercicio anterior el número de acciones es más o menos líneal hasta que se aumenta el número de transportadores y las necesidades de las localizaciones.

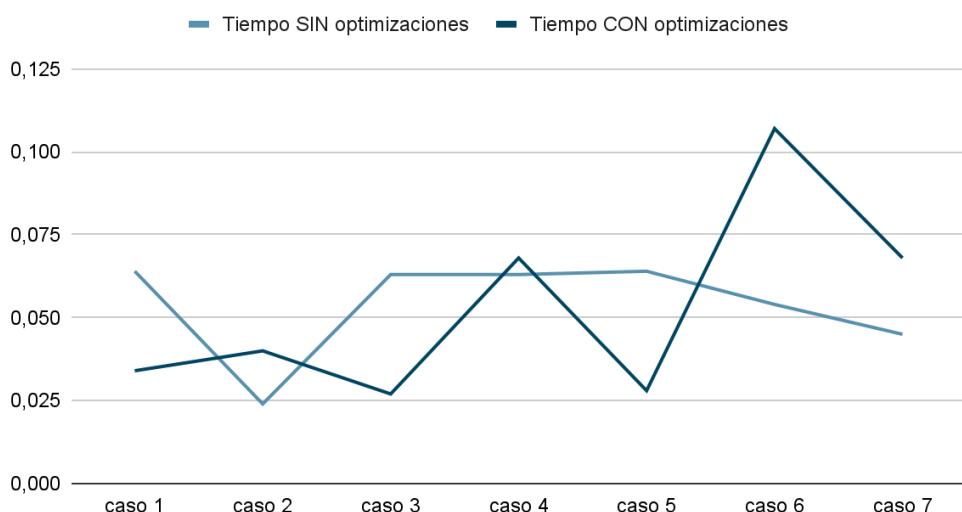
-Tabla comparativa de los resultados obtenidos con los datos anteriores con y sin mejoras

Nº Transportadores	Límite transportadores (en orden)	Nº lugares que necesitan caja	Nº cajas necesita cada lugar (en orden)	Tiempo SIN optimizaciones	Tiempo CON optimizaciones	Coste del plan SIN optimizaciones	Coste del plan CON optimizaciones	Nº Acciones SIN optimizaciones	Nº Acciones CON optimizaciones
1	10	1	50	0,064	0,034	550	550	210	210
1	10	1	51	0,024	0,04	650	650	214	214
1	40	1	50	0,063	0,027	280	280	204	204
3	10, 20, 30	1	50	0,063	0,068	360	250	206	204

3	20, 50, 100	1	50	0,064	0,028	270	150	204	202
3	20, 50, 100	3	20, 50, 100	0,054	0,107	470	470	686	686
1	150	2	50, 100	0,045	0,068	500	500	604	604

-Gráfica que compara los tiempos del problema sin optimizar con el optimizado.

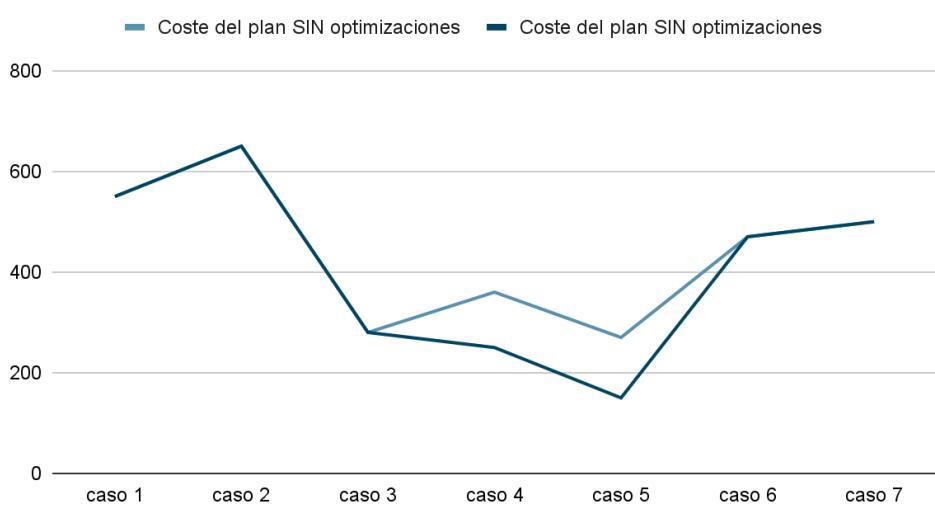
Points scored



En esta gráfica se observa cómo el tiempo del problema optimizado es generalmente menor al sin optimizar excepto en los últimos casos que al tener transportadores de 20,50 y 100 y las mismas necesidades siempre carga los transportadores en orden y en el último caso que solo hay un transportador de límite 150 aumenta el tiempo al tener que volver siempre al depósito.

-Gráfica que compara los costes del problema sin optimizar con el optimizado.

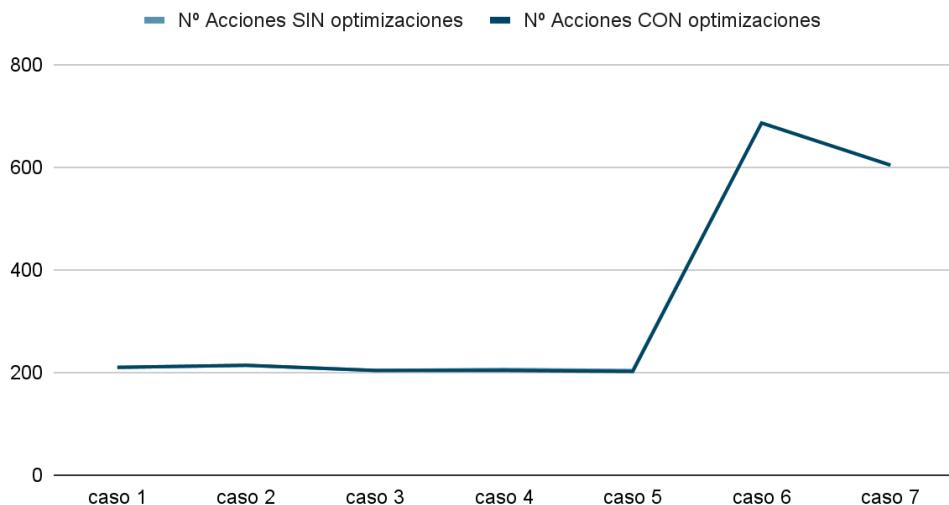
Points scored



En esta gráfica se observa que el coste del problema optimizado es siempre igual o menor al original siendo siempre mejor el optimizado ya que escoge el transportador que mejor se ajusta.

-Gráfica que compara el número de acciones realizadas del problema sin optimizar con el optimizado.

Points scored



Se observa que el número de acciones es el mismo en ambos casos excepto en el caso 4 y 5 en los cuales en el problema optimizado se llevan a cabo dos acciones menos.

En conclusión, las modificaciones realizadas mejoran generalmente la planificación al disminuir el coste del problema (o igualar) y/o disminuir el tiempo necesario para generar el plan en la mayoría de los casos. En cualquier caso los cambios de optimización nunca empeoran la planificación sino que la mejoran o igualan.