Planificación Inteligente
Práctica 2:
Aprendizaje por Refuerzo en Planificación

Ángel Aso Mollar Eva Onaindía

Curso 2023-2024

Índice

1. Objetivo de la práctica

Esta práctica tiene como objetivo entender los beneficios y limitaciones de la aplicación de métodos tabulares de Aprendizaje por Refuerzo para los procesos de toma de decisiones en dominios concretos. Para ello, se utilizará una herramienta llamada PDDLGym que proporciona un entorno adecuado para el desarrollo de dichos métodos en dominios y problemas específicos de toma de decisiones, definidos utilizando el lenguaje PDDL.

El alumnado deberá comprender en detalle el algoritmo Q-learning como método fundamental para el aprendizaje de políticas de actuación en entornos con incertidumbre. Además, deberá revisar el dominio de aplicación que ha realizado en la práctica anterior y tendrá que modificarlo para que tenga cabida dentro del entorno PDDLGym.

Se deberá implementar el algoritmo Q-learning tabular. Esto incluye diseñar y establecer la estructura de la tabla Q, así como las actualizaciones de los valores Q a lo largo de las iteraciones. Se debe experimentar con los distintos parámetros y estrategias de exploración ε-greedy. Concretamente se debe hacer énfasis en la estrategia de exploración/explotación utilizada (variación del parámetro ε, por ejemplo) y se debe hacer una breve justificación de ésta.

Por último, el alumnado deberá evaluar y analizar los resultados obtenidos, reflexionando también sobre la aplicabilidad y limitaciones de estos métodos dentro del contexto a analizar. Se espera un análisis de tamaños admitidos de problemas, ya sea en forma gráfica o tabular, y una breve justificación del por qué de ese análisis.

2. PDDLGym

La herramienta que se utilizará en esta práctica de Planificación y Aprendizaje por Refuerzo es PDDLGym [2], la cual se presentó en el workshop Planning and Reinforcement Learning (PRL) de la International Conference on Automated Planning and Scheduling (ICAPS 2020).

Esta sección se divide en dos apartados. En la sección 2.1 se darán unas nociones básicas de la herramienta Gym sobre la que se construye PDDLGym. En la sección 2.2 se describirá la herramienta PDDLGym y como utilizarla para crear un entorno de Aprendizaje por Refuerzo (AR).

2.1. Gym

El tookit Gym de OpenAI [1] surge como un proyecto para unificar la investigación en AR proporcionando distintos benchmarks mediante una sola API. Gym, hoy en día actualizado a Gymnasium[1], es una herramienta de AR que presenta una interfaz cómoda de manejar junto con una serie de entornos comunes en AR como resolución de problemas clásicos de control, aprendizaje de robots a caminar, movimientos con un objetivo, videojuegos de Atari, lanzamiento de cohetes, etc.

Los benchmarks que proporciona Gym se representan mediante el concepto de entorno o Gym *environment*. Un *environment* contiene al agente de AR, el entorno y todas las funciones de interacción con el entorno, como, por ejemplo, iniciar un problema o realizar una acción. Además, la mayoría de entornos de AR de Gym tienen una función de renderizado que nos permite observar las acciones que el agente va realizando y cómo dichas acciones modifican su estado. La Figura 1 muestra un ejemplo de un entorno de alunizaje.

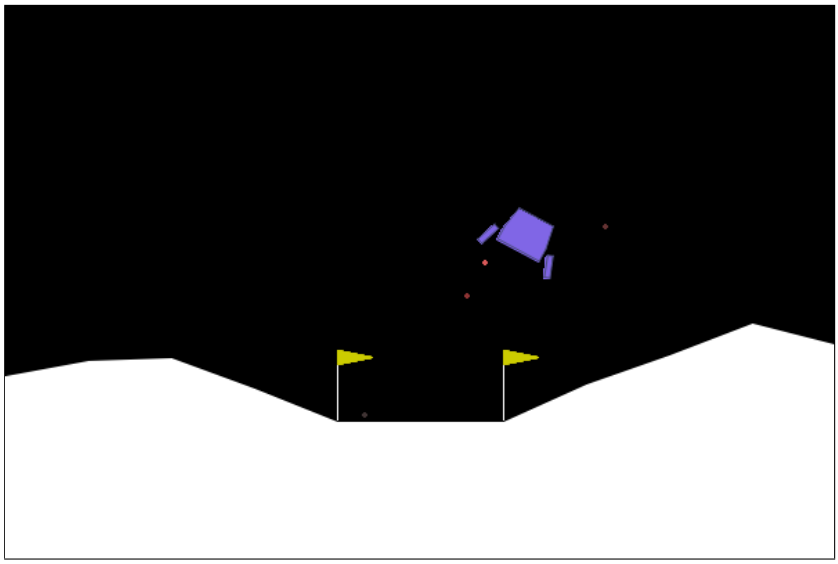

Figura 1: Entorno de alunizaje en Gym.

Lo más interesante del toolkit Gym es que tanto el espacio de estados como el espacio de acciones en los entornos de AR están predefinidos para

[1] `https://github.com/Farama-Foundation/Gymnasium`

cada tarea, por lo que el usuario no se tiene que preocupar de hacer ninguna modelización del entorno sino simplemente diseñar su algoritmo de AR y utilizarlo; es decir, el usuario solo tiene que especificar la acción dentro del espacio de acciones que se va a escoger para cada estado. Lo único que se debe programar, por tanto, es la política.

2.2. PDDLGym para tareas de planificación

PDDLGym es un framework de código abierto que está programado sobre el toolkit Gym para facilitar la construcción de un Gym *environment* a partir de un fichero de dominio PDDL. A través de los entornos construidos el agente de AR podrá resolver problemas de planificación especificados en PDDL mediante aprendizaje por refuerzo.

La librería original de PDDLGym da soporte a un gran número de dominios, introduciendo además funciones de renderizado para poder observar el proceso de planificación. Se puede crear un entorno nuevo a partir de un dominio especificado por nosotros en PDDL de una manera muy sencilla (ver sección 4).

Hay dos cosas importantes a tener en cuenta en PDDLGym: el espacio de estados (definido como espacio de observaciones) y el espacio de acciones.

El **espacio de observaciones** del entorno Gym creado con PDDLGym se gestiona como el espacio de estados de planificación donde cada estado es compuesto por sus literales junto con el objetivo y los objetos del problema que se está resolviendo.

El **espacio de acciones** se puede definir de dos formas:

1. Se puede equiparar el espacio de acciones AR íntegramente con el conjunto resultante de instanciar las acciones del dominio de planificación con los objetos del problema (acciones instanciadas o *grounded*).

2. Se puede hacer una distinción un poco más elaborada y considerar que el espacio de acciones AR contenga únicamente las acciones que utilizan objetos libres, es decir, objetos cuyo estado puede cambiar a lo largo del tiempo. Por ejemplo, una ciudad en un dominio de transporte no sería un objeto libre, puesto que su posición es fija, pero un camión sí sería un objeto libre.

En este trabajo, por simplicidad, utilizaremos la primera opción, puesto que la segunda implica la definición de predicados extra en el dominio y problema.

3. Instalación de PDDLGym

En esta sección se explican los pasos para instalar PDDLGym en nuestro ordenador. Los experimentos de esta práctica se pueden realizar en la máquina virtual Linux del DSIC, a la que se puede acceder a través del escritorio remoto desde la página Polilabs[2]. También puede realizarse en local.

Aspectos IMPORTANTES antes de inicial la práctica:

1. El código de esta práctica se ejecutará en CPU por lo que potencialmente funcionará en cualquier máquina y desde cualquier sistema operativo. No obstante, recomendamos que si se hace en local se compruebe que el procesador iguala al de la máquina virtual Linux del DSIC ya que los resultados de referencia de los dominios (ver sección 7) están calculados a partir de ejecuciones en esta máquina.

2. Se recomienda utilizar un entorno `conda` para evitar inconsistencias con otros paquetes que podamos tener instalados (en la máquina virtual Linux no hace falta ya que se borra todo pasado un tiempo)

Los pasos para instalar PDDLGym son:

1. Clonamos el repositorio PDDLGym desde el Github oficial:

   ```
   git clone https://github.com/tomsilver/pddlgym.git
   ```

2. En la carpeta clonada instalamos PDDLGym utilizando `pip`:

   ```
   cd pddlgym
   pip install -e .
   ```

3. ¡Y listo! Podemos comprobar que funciona utilizando el script `demo.py`.

   ```
   python pddlgym/demo.py
   ```

   Como se puede observar, la demo ejecuta diversos problemas de PDDLGym de distintos entornos tomando decisiones aleatorias.

IMPORTANTE. Se recomienda guardar PDDLGym, a ser posible, en nuestro directorio personal W para que no se borren las modificaciones. Hay que tener en cuenta que si trabajamos en los escritorios remotos del DSIC (como Polilabs) habrá que repetir `pip install -e .` cada vez que entremos.

[2] `https://polilabsvpn.upv.es/uds/page/login`

4. Creación de un entorno en PDDLGym

En esta sección se explica cómo crear un entorno en PDDLGym para modelar una tarea de planificación (dominio y problema codificados en PDDL) con el conocido dominio del mundo de bloques. A la hora de definir un entorno de planificación en PDDLGym debemos tener en cuenta que PDDLGym impone algunas limitaciones en el uso del lenguaje PDDL (ver detalles en sección 7).

El ejemplo del mundo de bloques que se muestra en esta sección se encuentra ya disponible en el repositorio de la herramienta PDDLGym. No obstante, se mostrarán todos los pasos del proceso de creación puesto que nuestro objetivo es crear un entorno desde el inicio para ejemplificar el uso de la herramienta.

En primer lugar, definimos el dominio en un fichero PDDL al que llamaremos `bloques.pddl`. El dominio es el siguiente:

```
(define (domain bloques)
    (:requirements :strips :typing)
    (:types bloque)
    (:predicates (on ?x - bloque ?y - bloque)
                 (ontable ?x - bloque)
                 (clear ?x - bloque) (handempty)
                 (holding ?x - bloque))
    (:action pick-up
        :parameters (?x - bloque)
        :precondition (and (clear ?x) (ontable ?x) (handempty))
        :effect (and (not (ontable ?x)) (not (clear ?x))
                     (not (handempty)) (holding ?x))
    )
    (:action put-down
        :parameters (?x - bloque)
        :precondition (and (holding ?x) (not (handempty)))
        :effect (and (not (holding ?x)) (clear ?x)
                     (handempty) (ontable ?x))
    )
    (:action stack
        :parameters (?x - bloque ?y - bloque)
        :precondition (and (holding ?x) (clear ?y) (not (handempty)))
```

```
        :effect (and  (not (holding ?x)) (not (clear ?y))
                      (clear ?x) (handempty) (on ?x ?y))
    )
    (:action unstack
        :parameters (?x - bloque ?y - bloque)
        :precondition (and (on ?x ?y) (clear ?x) (handempty))
        :effect (and (holding ?x) (clear ?y) (not (clear ?x))
                     (not (handempty)) (not (on ?x ?y)))
    )
)
```

Y creamos un pequeño problema en un fichero que llamaremos `pfile1.pddl`:

```
(define (problem bloques)
    (:domain bloques)
    (:objects a b c - bloque)
    (:init
        (clear a) (clear b) (clear c)
        (ontable a) (ontable b) (ontable c)
        (handempty)
    )
    (:goal (and (on a b) (on b c)))
)
```

Los pasos a seguir son los siguientes:

1. Nos situamos en la carpeta `pddlgym` clonada anteriormente (observad que dentro de esta carpeta hay otra también llamada `pddlgym`)
2. Cambiar al directorio o carpeta `pddlgym/pddl` (`cd pddgym/pddl`)
3. Copiamos el fichero `bloques.pddl` en la carpeta (directorio) actual
4. Creamos una carpeta con el mismo nombre del archivo del dominio `bloques` (sin la extensión .pddl)
5. Copiamos el fichero del problema `pfile1.pddl` en la carpeta `bloques`

Una vez realizados estos pasos ya tenemos el fichero de dominio y problema en las ubicaciones correctas. Podemos observar que la carpeta `pddl` contiene otros dominios y problemas que vienen por defecto con la herramienta PDDLGym y que también podemos probar.

A continuación, debemos realizar una serie de modificaciones para que PDDLGym tenga en cuenta el nuevo dominio que hemos creado. Para ello, vamos a la carpeta `pddlgym` dentro de la carpeta clonada y en el fichero `__init__.py` buscamos la lista donde están todos los dominios y añadimos el nuestro:

```
for env_name, kwargs in [
    ("bloques", {'operators_as_actions': True,
                 'dynamic_action_space': True}),
...
```

La opción `operators_as_actions` se pone a `True` para equipar las acciones que utilice el agente de AR con las acciones de un planificador clásico; es decir, estaríamos utilizando la opción 1 del espacio de acciones que se describe en la seccion 2.2.

La opción `dynamic_action_space` se pone a `True` para que en un estado sola sean accesibles las acciones que son aplicables en dicho estado, es decir, las acciones cuyas precondiciones se satisfacen en el estado. Si esta opción se pone a `False` entonces el conjunto completo del espacio de acciones del problema estaría accesible en todos los estados.

Una vez realizados todos los pasos mencionados, el código que se muestra a continuación genera automáticamente el entorno PDDLGym para resolver problemas de planificación del dominio `bloques`.

```
import pddlgym

# Creo entorno pddlgym con el dominio
env = pddlgym.make("PDDLEnvBloques-v0")

# Fijo el problema al primero alfabéticamente (por si hay varios)
env.fix_problem_index(0)

# Con la función reset() empiezo un episodio devolviendo al agente
# al estado inicial.
```

```
state, debug_info = env.reset()

# También podemos ver las acciones aplicables en el estado
print(env.action_space.all_ground_literals(state))

done = False

while not done:

   # Escojo una accion aleatoria en el espacio de acciones
   # del estado a aplicar
   action = env.action_space.sample(state)

   # La muestro por pantalla
   print(action)

   # La aplico y genero un nuevo estado
   state, reward, done, info = env.step(action)
```

Creamos el archivo `randlearning_bloques.py` (el nombre viene de 'random learning') con el código mostrado que permitirá crear un agente AR para resolver el problema definido en el entorno y lo copiamos en la carpeta `pddlgym` clonada. Al ejecutar el fichero (`python randlearning_bloques.py`) nos devolverá un plan aleatorio para el problema que hemos definido en el fichero `pfile1.pddl`.

Como se observa en el código del archivo `randlearning_bloques.py`, el entorno creado es reconocido por PDDLGym con el nombre '`PDDLEnvBloques-v0`'. De forma genérica, cuando creamos un entorno de esta forma a partir de un fichero de dominio `pddl`, el entorno será accesible con el nombre `PDDLEnv`*Dominio*`-v0`, donde *Dominio* será el nombre del dominio con **la primera letra en mayúscula**.

5. Desarrollo de la práctica

Para el desarrollo de la práctica el alumnado tendrá que implementar el algoritmo Q-learning de Aprendizaje por Refuerzo para resolver problemas de planificación de un dominio, es decir, para aprender a planificar. Esta

implementación se hará utilizando el módulo PDDLGym anteriormente introducido.

El objetivo es entrenar un agente de AR de tal forma que su modo de actuación desemboque en una resolución óptima o casi óptima del problema. Para ello, se proporciona el fichero `practica.py` (adjuntado en el Anexo A) que podrá servir como punto de partida para implementar el algoritmo Q-learning. Deberemos rellenar este fichero con nuestro código en las partes descritas a continuación, que vienen indicadas en el código del Anexo A mediante la palabra 'RELLENAR'.

5.1. Definición de la Q-table

La Q-table es la estructura principal dentro del proceso de entrenamiento del algoritmo Q-learning. Dentro de nuestro contexto, para la correcta definición de la Q-table hay varios aspectos que deben tenerse en cuenta:

- La tabla tendrá una fila por cada posible estado del problema y una columna por cada acción, todas inicializadas a cero.
- En PDDLGym se puede acceder al espacio completo de acciones mediante el atributo de entorno `e.action_space._all_ground_literals`. Cabe destacar que este atributo solamente es accesible si se hace primeramente una llamada a la función del espacio de acciones del entorno `e.action_space.all_ground_literals(state)`, que devuelve todas las acciones aplicables en un estado dado.
- El número de columnas de la tabla se tiene que definir inicialmente puesto que se debe garantizar que el algoritmo pueda explorar todas las acciones que son aplicables en un estado desde el inicio del aprendizaje.
- Las filas de la tabla, sin embargo, se pueden definir de manera incremental, por lo que **no es necesario** llevar un conteo explícito del espacio de estados desde el inicio.

El algoritmo Q-learning entrenará la Q-table de tal forma que, al final de las iteraciones, se podrá encontrar una solución al problema simplemente escogiendo dentro de la fila que corresponda a cada estado la acción que otorgue mayor valor Q.

En la Figura 2 podemos ver la tabla Q ya entrenada para el dominio y problema del mundo de bloques que se ha definido en la Sección 4. Se observa que la matriz tiene un tamaño 22×18 donde 22 es el número de estados visitados y 18 es la dimensión del espacio de acciones del problema:

```
(pick-up a) (pick-up b)(pick-up c)(put-down a) ...
(stack a b) (stack a c)(stack b a) ...
(unstack a b)(unstack a c)(unstack b a) ...
```

Nótese que el espacio de acciones de cualquier problema del mundo de bloques donde se definan tres bloques serán las mismas 18 acciones. En general, para cualquier dominio, el tamaño del espacio de acciones (con salvedad de cambio de nombres) únicamente depende de los objetos del problema y sus tipos.

Las filas de la tabla se inicializan a cero y se entrenan teniendo en cuenta la recompensa.

```
NUM_ACCIONES: 18
Score:  1.0
[[0.95 0.97 0.95 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.96 0.00 0.00 0.94 0.94 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.96 0.00 0.00 0.00 0.00 0.94 0.94 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.93 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.95]
 [0.00 0.00 0.00 0.00 0.96 0.00 0.00 0.00 0.96 0.98 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.95 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.97 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.96 0.00 0.00 0.00 0.00 0.00 0.94 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.99 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.97 0.00 0.00]
 [0.00 0.00 0.00 0.98 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.93 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.95 0.00]
 [0.00 0.00 0.00 0.00 0.94 0.00 0.00 0.00 0.00 0.92 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.93 0.00 0.00]
 [0.00 0.93 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.95 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.93 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.95 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.94 0.00 0.00 0.00 0.00 0.92 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.93 0.00]
 [0.00 0.00 0.00 0.94 0.00 0.00 0.00 0.92 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.93 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.94 0.00 0.00 0.00 0.92 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.93 0.00 0.00 0.00]
 [0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.95]]
ESTADOS_VISITADOS: 22
```

Figura 2: Q-table ya entrenada para el dominio `bloques`.

5.2. Parámetros de exploración

A la hora de decidir la mejor acción a aplicar en un algoritmo de Aprendizaje por Refuerzo es interesante hacer un balance entre exploración y ex-

plotación. La **explotación** implica utilizar lo ya aprendido por el agente para intentar buscar una solución inmediatamente mejor, es decir, le permite aprovechar y explotar el conocimiento ya obtenido suponiendo que ha encontrado un buen camino. Por otro lado, la **exploración** se basa en tomar ciertas decisiones que se consideran subóptimas en el contexto actual, con la esperanza de encontrar mejores caminos a futuro que no han sido previamente explorados. Sin exploración, los agentes podrían quedarse en políticas subóptimas en su proceso de optimización.

En esta práctica cada equipo de trabajo debe realizar su propio algoritmo de exploración, basándose en la exploración ε-greedy. Para ello, un agente AR explorará opciones nuevas con una cierta probabilidad ε, y explotará las ya conocidas con una probabilidad $1-\varepsilon$. Se debe proponer una cierta estrategia de cambio de este umbral ε, justificando el por qué de la decisión.

5.3. Actualización de la tabla

El algoritmo Q-learning utiliza la información del estado alcanzado tras la ejecución de una acción así como la recompensa obtenida en cada instante para actualizar continuamente sus valores. En este apartado hay que implementar la actualización de los valores Q del algoritmo Q-learning.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

5.4. Planificar

Cuando la tabla Q ya está entrenada, lo único que queda por hacer es devolver para cada estado la acción que tenga mayor valor Q. Para ello, se imprimirá por pantalla la acción que el algoritmo entrenado devuelve en cada paso. La lista de acciones que devuelve es **un plan para el problema** con el que hemos entrenado al agente de Aprendizaje por Refuerzo.

6. Codificación PDDL en PDDLGym

En líneas generales se podrá reutilizar la especificación del dominio PDDL realizada en la práctica 1 para este trabajo. No obstante, es necesario conocer previamente las limitaciones y funcionalidades del lenguaje PDDL que

admite PDDLGym para determinar si tenemos que hacer algún cambio en la codificación de nuestro dominio PDDL.

A continuación se detallan los aspectos más relevantes que deben tenerse en cuenta a la hora de especificar en PDDL el dominio y problema para que funcionen en PDDLGym.

6.1. Requerimientos

1. Los tipos que se especifican en la sección `:types` deben ir precedidos OBLIGATORIAMENTE de un espacio en blanco. Por ejemplo, la definición `school home - place` estaría bien, pero `school home -place` no es correcta. Lo mismo sucede en la definición de tipos de los argumentos en los predicados: `(at_school ?x -person ?y -school)` no estaría bien, pero `(at_schol ?x - person ?y - school)`, donde se utiliza el espacio en blanco antes del tipo, es correcto.

2. En los parámetros de las acciones, la definición de objetos según el tipo debe ser uno a uno, es decir, PDDLGym no permite definiciones de múltiples objetos del mismo tipo seguidos. Por ejemplo,

   ```
   :parameters (?ori ?dest - location)
   ```

   no está permitido, hay que poner

   ```
   :parameters (?ori - location ?dest - location)
   ```

6.2. Funcionalidades permitidas

1. **Precondiciones negadas**. A diferencia de lo que ocurre con el planificador OPTIC, PDDLGym sí admite el uso de precondiciones negadas en la codificación de las acciones del dominio.

6.3. Limitaciones

Son varias las limitaciones del lenguaje PDDL en PDDLGym. La más importante y que afectará a la mayoría de codificaciones de los equipos es la primera, la imposibilidad de utilizar la expresión `(either ...)` para unificar tipos.

1. PDDLGym **no admite definiciones con either**; por tanto, si hemos empleado la expresión `either` para definir nuestro dominio en la

práctica 1, tendremos que redefinir los predicados y las acciones. Por ejemplo, dado un predicado como:

`(at ?x - person ?y - (either school home work))`

tendremos que generar tres predicados, uno por cada valor del campo `either`:

```
(at_school ?x - person ?y - school)
(at_home ?x - person ?y - home)
(at_work ?x - person ?y - work)
```

Del mismo modo, si existe una acción con un argumento definido con `either` se debe desglosar la acción manteniendo la lógica del problema.

2. PDDLGym **no admite literales con argumentos repetidos**, es decir, literales que tengan dos o más argumentos que sean el mismo objeto. Por ejemplo, un literal como `(top o1 o1)` no se permite en PDDLGym. En este caso debe especificarse otro predicado para generar un literal diferente. Por ejemplo, se puede especificar el predicado `(top_ ?x - object)` que daría lugar a un literal como `(top_ o1)`, con el cual indicaríamos que encima del objeto `o1` está el propio objeto `o1`.

3. Se puede **usar constantes en PDDLGym pero sin tipificar**. Por ejemplo, en el dominio se puede escribir una expresión como:

   `(:constants FRIO TEMPLADO CALOR)`

   pero no se puede escribir:

   `(:constants FRIO TEMPLADO CALOR - tiempo)`

4. Si se quiere utilizar el símbolo `=` para especificar restricciones de igualdad o desigualdad (previa incorporación de `(:requirements equality)`) hay que realizar un pequeño cambio en un fichero, debido a que PDDLGym presenta problemas con la utilización de `=`. El cambio hay que realizarlo en la función `_compute_all_ground_literals(self, state)` que se encuentra en el fichero `spaces.py`. Concretamente, hay que añadir las líneas que se muestran en rojo en el código:

```
def _compute_all_ground_literals(self, state):
    all_ground_literals = set()
```

```
for predicate in self.predicates:
    if predicate.name == '=': pass
    else:
        choices = [self._type_to_objs[vt] for vt in predicate.var_types]
        for choice in itertools.product(*choices):
            if len(set(choice)) != len(choice): continue
            lit = predicate(*choice)
            all_ground_literals.add(lit)
return all_ground_literals
```

7. Dominios

En esta sección se propone, para cada dominio, una serie de orientaciones para diseñar un problema abordable por un agente de Aprendizaje por Refuerzo.

7.1. Dominio *Puerto*

Para diseñar un problema abordable por el agente de AR se recomienda, a modo de orientación:

- Eliminar las alturas de las pilas.
- Utilizar solo dos pilas en cada muelle.
- Probar con un máximo de dos objetivos, dependiendo del número de contenedores.

7.2. Dominio *Robots*

De acuerdo a las pruebas realizadas por el profesorado con el dominio **Robots**, no es necesario aplicar ninguna restricción particular al dominio ni al problema original de servir dos tazas de té, una para Juan y otra para Pedro.

A modo de orientación, con la codificación empleada por el profesorado, el número de acciones de la tabla Q para el problema original es 36, y el problema es abordable con el algoritmo Q-learning. No obstante, si algún equipo experimenta dificultades, puede limitar el proceso de aprendizaje para resolver únicamente el objetivo de servir una taza de té a Pedro.

7.3. Dominio *Aeropuerto*

Para diseñar un problema abordable por el agente de AR se recomienda, a modo de orientación:

- Reducir el tamaño de la terminal eliminando las puertas 3, 4, 7 y 8.
- El número de equipajes que puede haber en un vagón se limita a 1 (se recomienda eliminar, por tanto, el conteo).
- Diseñar un solo objetivo en el problema, es decir, transportar un solo equipaje entre dos puntos de la terminal del aeropuerto. El equipaje puede ser sospechoso o no sospechoso.

7.4. Dominio *Transporte*

Para diseñar un problema abordable por el agente de AR se recomienda, a modo de orientación:

- Diseñar un solo objetivo en el problema, es decir, transportar un único paquete de un punto a otro.
- Limitar el problema a dos ciudades.
- Utilizar un único medio de transporte entre las dos ciudades, bien aviones o trenes.

7.5. Dominio *Ascensores*

Para diseñar un problema abordable por el agente de AR se recomienda, a modo de orientación:

- Eliminar capacidad de ascensores.
- Limitar el problema a dos bloques, con un ascensor por bloque (aparte del rápido).
- Probar con un máximo de dos o tres objetivos (dos o tres personas).

8. Entrega y evaluación de la práctica

Se deberá presentar una memoria de trabajo realizado que incluya los puntos que se muestran a continuación.

- **0.5 puntos**: Adaptación del dominio PDDL proposicional del trabajo práctico anterior a PDDLGym
- **1 punto**: Desarrollo del algoritmo Q-learning
  - **0.5 puntos**: Implementación y justificación de la técnicas de exploración/explotación
  - **0.5 puntos**: Implementación de la tabla y la actualización Q-learning
- **1.5 puntos**: Análisis de tamaños de problemas, comparativa con los resultados de un planificador y conclusiones del trabajo
  - **0.5 puntos**: Análisis gráfico/tabular de convergencia para tamaños de problema según hiperparámetros (learning rate, epsilon, factor de descuento)
  - **0.5 puntos**: Comparativa de resolución con resultados de planificador
  - **0.5 puntos**: Conclusiones del trabajo

Referencias

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym. *CoRR abs/1606.01540* (2016).

[2] Silver, T., and Chitnis, R. PDDLGym: Gym Environments from PDDL Problems. In *International Conference on Automated Planning and Scheduling (ICAPS) PRL Workshop* (2020).

Anexo A: Código de la práctica

```
import numpy as np
import pddlgym
import random


'''
      INICIALIZACIÓN DE PARÁMETROS

'''

# Crear entorno de PDDLGym a partir de nuestro dominio
env = pddlgym.make ("PDDLEnv<Domain>-v0")

# Fijar el problema del entorno
env.fix_problem_index(0)

# Iniciar el entorno con el agente para que esté en el lugar inicial
state, debug_info = env.reset()


# Definición de la Q-table

#########################################################################
#                                                                       #
#                 RELLENAR: DEFINICIÓN DE Q-TABLE                       #
#                                                                       #
#########################################################################

'''
      DEFINICIÓN DE HIPERPARÁMETROS DEL Q-LEARNING

'''

total_episodes = 1000
learning_rate = 0.2   # Alpha in Q-learning algorithm
max_steps = 2000
```

```
gamma = 0.99     # Discount factor

########################################################################
#                                                                      #
#                 RELLENAR: PARÁMETROS DE EXPLORACIÓN                  #
#                                                                      #
########################################################################

# Ejemplo
epsilon = 0.01



'''
      ALGORITMO Q-LEARNING
'''
# Entrenamos hasta un número máximo de episodios (reinicios)
for episode in range(total_episodes):

   state, debug = env.reset()

   # El agente irá tomando decisiones hasta un número máximo de pasos
   for step in range(max_steps):

      '''
            EXPLORACIÓN-EXPLOTACIÓN
      '''
      # Ejemplo
      if random.uniform(0,1) > epsilon:
         # EXPLOTACION
         pass
      else:
         # EXPLORACIÓN SEGÚN ALGORITMO
         action = env.action_space.sample(state)

      new_state, reward, done, info = env.step(action)
```

```
      ########################################################################
      #                                                                      #
      #                     RELLENAR: ACTUALIZAR TABLA                       #
      #                                                                      #
      ########################################################################


      state = new_state
      if done:
         break




'''
      APLICACIÓN DE LA Q-TABLE PARA SACAR UN PLAN CON LA POLÍTICA
'''

state, debug = env.reset()

while True:

   # Valor numérico del estado para sacar la fila de la tabla
   index = vistos.index(state)

   ########################################################################
   #                                                                      #
   #                       RELLENAR: PLANIFICAR                           #
   #                                                                      #
   ########################################################################

   accion_a_aplicar = None

   state, reward, done, info = env.step(accion_a_aplicar)

   # Acabo cuando llego al objetivo
   if done:
      break
```

Anexo B: Errores comunes

1. **AssertionError: Cannot mix hierarchical and non-hierarchical types**. Se debe comprobar que en la definición de :objects no existe ningún tipo que no vaya seguido de espacio. Por ejemplo, esto sería incorrecto: `greenContainer normalContainer -container`, puesto que falta el espacio entre `-` y `container`.

2. **AssertionError: Predicate and is not defined**. Este error suele darse cuando se copia algo desde Windows a Linux, puesto que se copian una serie de artificios que el parser de PDDLGym no reconoce. Lo ideal es abrir el archivo en Visual Studio Code (en escritorio remoto con la orden `code dominio.pddl`) y observar si existe algún caracter extraño.

3. **AssertionError: Cannot mix hierarchical and non-hierarchical types**. Este error sale cuando en el fichero de dominio se han definido subtipos. Los tipos que no pertenezcan a ningún subtipo tienen que ponerse explícitamente que pertenecen a `object`.