SID Primavera 2025
Práctica 3 (Laboratorio)

Sergio Álvarez
Javier Vázquez

Mayo/Junio 2025

1. Introducción

En las sesiones de laboratorio, en la parte de aprendizaje por refuerzo multiagente, vemos diversos algoritmos considerados *fundacionales* debido a que son la base de métodos más modernos (y más complejos). El primero que hemos visto es Joint Action Learning with Game Theory (JAL-GT), que consiste en un Q-Learning donde el estado puede ser una observación parcial (o un histórico de observaciones, aunque por simplificar no vamos a ir por esta vía) y la acción es una acción conjunta. Los valores Q se utilizan para construir conceptualmente una matriz de recompensas, sobre la que se pueden aplicar soluciones de concepto como Minimax, eficiencia de Pareto o equilibrio de Nash.

2. Objetivo de la práctica

A partir de un código que os pasamos[1] y que define un *baseline* o modelo de referencia, se os pide que realicéis un estudio del rendimiento del algoritmo JAL-GT sobre el entorno POGEMA[2].

El objetivo principal es el de analizar, a partir de ciertas preguntas que os planteamos, las hipótesis y la experimentación que planteéis y el análisis que realicéis, cómo se comporta JAL-GT, con la parametrización que os proponemos, de manera comparativa, teniendo en cuenta otras posibles parametrizaciones, soluciones de concepto y algoritmos.

2.1. El entorno: POGEMA

POGEMA (*Partially-Observable Grid Environment for Multiple Agents*) es un entorno bidimensional multiagente con observabilidad parcial. El entorno está

[1] https://drive.google.com/file/d/1AZNZSpECvQhRNKU_hf88Rpjzu52iq6wC/view?usp=share_link

[2] https://github.com/AIRI-Institute/pogema

representado por una cuadrícula con una anchura y una altura determinadas, con la presencia de obstáculos que los agentes no pueden cruzar ni ocupar. Como mínimo, el perímetro del mapa está formado por obstáculos (son los muros que delimitan el mapa). Además, puede haber posiciones aleatorias en la cuadrícula que también son obstáculos y, de existir, vienen determinados por la semilla a partir de la cual se crea el entorno.

Los agentes comienzan en posiciones aleatorias y tienen un objetivo a alcanzar. Estas posiciones también están determinadas a partir de la misma semilla. Por defecto, sólo se entrega una recompensa con valor 1 cuando un agente llega a su objetivo correspondiente, tras lo cual ese agente desaparece. Por lo tanto, si hay dos agentes y los dos cumplen su objetivo, la recompensa total del entorno será 2. No hay recompensa ni un agente desaparece si accede a una posición en la que hay un objetivo de otro agente.

La ejecución del entorno es, por defecto, paralela y no por turnos. Es decir, todos los agentes reciben sus observaciones y deciden sus acciones a la vez. Dos agentes no pueden ocupar la misma posición en el mismo paso de ejecución. Si dos agentes pueden acceder a la misma posición y lo intentan en el mismo paso, el agente con identificador más pequeño accede a la posición y el otro no se mueve.

La observabilidad es parcial y viene determinada por un parámetro que se pasa al entorno en su creación: el radio de observación. Todos los agentes tienen el mismo radio de observación. Cada turno, cada agente recibe observaciones correspondientes a una cuadrícula $(x - r, y - r, x + r, y + r)$, donde (x, y) es la posición del agente después de la última acción y r es el radio de observación. Es decir, si $r = 1$ entonces la cuadrícula es de 3×3, si $r = 2$ la cuadrícula es de 5×5 y así sucesivamente. La coordenada del agente siempre está en el centro de la cuadrícula.

La observación que recibe cada agente consta de tres matrices (*Numpy arrays*):

- Matriz de obstáculos con un valor de 1 si hay obstáculo en esa posición, 0 si no lo hay.
- Matriz de agentes con un valor de 1 si hay un agente en esa posición (incluyendo el agente que recibe la observación), 0 si no lo hay.
- Matriz de objetivo con un único valor de 1 en la posición de la cuadrícula más cercana al objetivo de ese agente. No aparecen objetivos de otros agentes y siempre hay un 1 (y sólo uno).

Para poder generar políticas o funciones de valor a partir de estas observaciones, es necesario convertirlas en estados discretos (si no utilizamos modelos más complejos). Para ello, hemos incluido una función `obs_to_state` con un ejemplo de conversión. En teoría, deberíamos estar usando históricos de observaciones como estados, ya que la observabilidad es parcial, pero en el entorno POGEMA esta simplificación es válida y hace que podamos plantear una experimentación viable con los modelos que usamos basados en *Numpy arrays*.

Por defecto, todas las ejecuciones de evaluación generan animaciones en formato SVG (se pueden abrir en un navegador, o en vista previa en algunos sistemas operativos) donde se puede observar el comportamiento de cada agente. Interpretar cualquier política se hace muy complicado con el tamaño del espacio de estados así que es recomendable utilizar estas animaciones para el análisis.

Antes de empezar a experimentar, dedicad un tiempo a mirar todos los parámetros de configuración de la estructura `exp_config` y entender bien en qué afecta cada uno.

2.2. Qué incluye el código *baseline*

En el código que os pasamos se incluyen los siguientes elementos:

- Una versión del algoritmo JAL-GT parecida a la del notebook de la sesión de laboratorio, con algunas optimizaciones y mejoras (`algorithms.py`).
- Una clase para almacenar información relevante del modelo de juego, incluyendo el espacio de acciones conjuntas (`game_model.py`).
- Varios conceptos de solución implementados (`solution_concepts.py`):
  - Minimax
  - Equilibrio (mixto[3]) de Nash
  - Bienestar (*welfare*)
  - Óptimo de Pareto[4]
- Un fichero de funciones auxiliares que sólo incluye la función para dibujar gráficas con Seaborn que hemos usado durante el curso (`utils.py`).
- El código para convertir observaciones en estados (`obs_to_state`), configurar el entorno (estructura `exp_config`), y entrenar y evaluar, en dos versiones: un fichero ejecutable (`main.py`) y un notebook (`baseline.ipynb`).
- Un fichero de dependencias (`requirements.txt`), que podéis instalar desde vuestro entorno virtual con el comando `pip install -r requirements.txt`.

Podéis modificar estos ficheros tanto como lo necesitéis para vuestra experimentación y análisis.

[3] No es un equilibrio mixto calculado de manera formal como lo hicimos en sesiones de problemas debido al coste de construir y resolver las ecuaciones lineales. Por eficiencia lo hemos implementado de la siguiente manera: si hay más de un equilibrio de Nash para un agente, se reparte la probabilidad uniformemente respecto a todos los que haya.

[4] De manera análoga a como lo hacemos para el equilibrio de Nash, si hay más de un óptimo de Pareto repartimos la probabilidad uniformemente.

2.3. Preguntas planteadas

Estas son las preguntas que deberían dirigir vuestro análisis:

- ¿Qué parámetros, y con qué valores, hacen que el algoritmo converja mejor en cuanto a recompensa, individual y colectiva? ¿Y en cuanto a tiempo de entrenamiento?
- ¿Cómo se comportan los agentes cuando están entrenados para optimizar su comportamiento respecto de cada concepto de solución incluido (minimax, Nash, *welfare*, Pareto)? ¿Para qué tipo de coordinación es más adecuado cada uno (intereses comunes, conflictivos, mixtos)?
- ¿Qué ocurre si entrenamos dos agentes con dos conceptos de solución diferentes? ¿Son capaces de coordinarse, o de conseguir recompensas individuales independientemente de la política del otro agente?
- ¿Cómo se compara JAL-GT, que es un algoritmo multiagente debido a que calcula la Q en base a acciones conjuntas, con IQL (*independent Q-Learning*)[5]?
- ¿Cómo es capaz de generalizar el algoritmo? Es decir: con el resto de parámetros fijados, ¿cómo es capaz el algoritmo de converger a medida que escala el tamaño del entorno (2×2, 3×3, 4×4, 8×8, ...), el número de agentes (2, 3, 4, ...), la complejidad de la representación del estado, etc.[6]?
- ¿Es capaz el algoritmo de generalizar de manera que tenga éxito evaluando mapas que no ha visto en entrenamiento?

Si lo creéis necesario, útil o relevante, podéis plantear preguntas adicionales a estas[7]. Los parámetros que podéis tener en cuenta para realizar el análisis pueden incluir (pero no tienen por qué limitarse a):

- El número de agentes
- El tamaño del entorno
- La probabilidad de obstáculos
- El algoritmo
- La solución de concepto

[5] La mejor manera de hacer esta comparativa es entrenando a los agentes con el mismo algoritmo Q-Learning que usasteis para FrozenLake, con cada agente ignorando el hecho de que el entorno es no-estacionario debido a la presencia de otros agentes.

[6] Algunos de estos parámetros puede hacer que el entrenamiento se vuelva tan complejo que sea intratable. No hace falta que forcéis las cosas: si no podéis entrenar a una velocidad suficiente, podéis concluir que la complejidad es excesiva con los parámetros en cuestión.

[7] Podéis reemplazar alguna pregunta planteada, pero ha ser de manera debidamente justificada.

- El número de episodios a entrenar
- El factor de descuento
- La señal de recompensa[8]
- El coeficiente de exploración y su descuento
- La tasa de aprendizaje y su descuento
- La complejidad de la representación del estado a partir de la observación

Para evaluar la calidad de los entrenamientos, se pueden usar los siguientes elementos, aunque no tiene por qué ser exactamente esta lista (podéis añadir o reemplazar elementos si lo justificáis):

- Tiempo de entrenamiento por episodio
- Número de episodios
- Tiempo de entrenamiento total
- Recompensa individual obtenida para algún agente
- Recompensa colectiva
- Optimalidad de la política resultante

3. Plazos y evaluación

La entrega tiene que incluir el código de los algoritmos, así como también el código para lanzar los experimentos. Puede incluir notebooks, siempre y cuando la documentación en Markdown de los mismos sea equivalente, en calidad y contenido, al que habríais realizado de haber hecho la documentación en un PDF aparte. Las únicas dependencias que podéis utilizar son las mismas que ya se incluyen en el fichero `requirements.txt`.

La fecha de entrega será el día **13 de junio (13/06/2025)**, y consistirá en:

- El código de experimentación y los algoritmos.
- Una documentación en `.pdf` (o su equivalente en notebooks) incluyendo:
  - Las decisiones de diseño de los experimentos, incluyendo hipótesis y una justificación de parámetros a probar y con qué rangos.
  - Un resumen de los resultados empíricos.

[8] Podéis probar alternativas a la señal de recompensa por defecto pero, si lo hacéis, tenéis que comparar entre vuestras propuestas y la versión por defecto.

- Análisis comparativo de los algoritmos y sus parametrizaciones, contextualizando cada resultado con respecto a las propiedades de los algoritmos tal como se vieron en las sesiones de teoría y laboratorio.

- Un fichero README con las instrucciones para parametrizar y ejecutar los experimentos.

Criterio	Peso	Expectativa
Ejecución	40 %	El código funciona correctamente a partir de las instrucciones del README, permitiendo fácilmente parametrizar nuevos experimentos.
Implementación	20 %	Cualquier código implementado de manera adicional para automatizar los experimentos, así como los algoritmos que se utilizan (e.g. Q-Learning para IQL, nuevos conceptos de solución propuestos, etc.) o sus modificaciones (e.g. epsilon o alpha decay) son consistentes con los objetivos de experimentación y con el funcionamiento teórico de los algoritmos.
Diseño de experimentos	20 %	Los experimentos están planteados a partir de hipótesis y están justificados en base al comportamiento teórico de los diferentes algoritmos y a las características del entorno.
Análisis de resultados	20 %	El resultado de los experimentos se analiza de manera correctamente contextualizada con respecto a las hipótesis de partida y a los algoritmos en cuestión y a sus propiedades y se concluye con posibles recomendaciones de cómo entrenar agentes para entornos similares. Para el análisis se utilizan de manera correcta los outputs de los algoritmos, como por ejemplo recompensas, errores de diferencia temporal, el comportamiento de los agentes en las animaciones, etc.

Figura 1: Rúbrica de evaluación de la práctica

La nota base de la práctica se evaluará según la rúbrica descrita en la Figura 1. Esta nota tendrá un valor máximo de 10. Aparte de esta nota base, existe la posibilidad de sumar puntos extra que permitirían tener una nota mayor de 10:

- 2 puntos extra a los grupos que añadan, a la comparativa, o bien una versión de JAL-GT que use como representación del estado un modelo lineal (con actualización por aproximación lineal) o una red neuronal, o bien un algoritmo de gradiente de política[9].

[9] Se verán en la penúltima sesión de laboratorio

La nota de esta práctica contará 1/3 de la nota global de laboratorio. Si la nota es superior al 10, para hacer media no se reducirá a 10 sino que se mantendrá superior.