Planificación en Entornos con Incertidumbre

Inteligencia Artificial (IS) 2018/19 - Propuesta de Trabajo

Ignacio Pérez Hurtado de Mendoza

12 de abril de 2019

1. Introducción

Planificar [5, 6] consiste en encontrar una secuencia de acciones para alcanzar un determinado objetivo cuando se ejecutan a partir de un determinado estado inicial. Existen multitud de aplicaciones en el mundo real y, en particular, veremos en este trabajo algunas relacionadas con la Robótica.

Comencemos definiendo algunos conceptos en un problema de planificación:

- Un **estado** $s \in S$ es una posible situación del problema de acuerdo a una determinada representación, siendo S el conjunto de todas las posibles situaciones en dicha representación.
- Una **acción** $a : S \to S$ es un operador básico para la transformación de estados. Denotaremos por A al conjunto de las acciones definidas para resolver el problema. Cabe mencionar que cada acción $a \in A$ tiene asociada una función de *aplicabilidad* $f_a : S \to \{0, 1\}$ que determina si la acción puede ser aplicable o no a un estado concreto.

De esta forma, un *plan* es una secuencia de acciones que, partiendo de un *estado inicial* $s_{init} \in S$, permite alcanzar un estado final $s_{goal} \in S_{goal}$ al que llamaremos *objetivo*, siendo $S_{goal} \subseteq S$ el conjunto de aquellos estados considerados como finales.

1.1. Planificación clásica

La *planificación clásica* [5] considera entornos que son:

- **Completamente observables**: El planificador conoce siempre el estado actual.
- **Deterministas**: El efecto de cada acción sobre cada estado es conocido y siempre es el mismo.

- **Finitos**: Los conjuntos de estados y acciones son finitos.
- **Estáticos**: La situación del problema sólo puede cambiar mediante la las acciones del planificador.
- **Discretos**: La situación del problema se puede describir de forma discreta:
  - Tiempo discreto
  - Acciones discretas
  - Objetos discretos
  - Efectos discretos

Por desgracia, muchos problemas de planificación interesantes de la vida real no pueden ser resueltos mediante planificación clásica, fallando en algunas o en todas las anteriores asunciones. Tomemos por ejemplo, el caso de un robot móvil de dos ruedas cuya misión es encontrar la puerta del despacho de un determinado profesor en el pasillo del Departamento de CCIA en la primera planta del módulo H de la ETSII. El robot tiene tres acciones: (1) *Avanzar a velocidad constante*, (2) *detenerse* y (3) *leer un nombre en una puerta*. Los sensores del robot son capaces de detectar puertas, capturar imágenes y saber cuantas vueltas ha dado cada una de sus ruedas. El robot cuenta con un software OCR para interpretar los nombres escritos en las imágenes capturadas. Cuando encuentre la puerta del profesor que busca, debe detenerse y mostrar en una pantalla el número de metros recorridos.

- El entorno no es *completamente observable*: Pues la percepción del robot depende de la calidad de los sensores y del procesamiento realizado con la información obtenida. Podrían haber problemas de calidad en las imágenes capturadas o errores en su procesamiento.
- Las acciones no son *deterministas*: Pues las condiciones del suelo y la calidad de los motores afectarían a las trayectorias realizadas en mayor o menor medida. Asimismo, podría existir un *bias* en los motores que no estuviera recogido en el modelo físico del robot.

Este trabajo se va a enfocar en la resolución de problemas de planificación en entornos *parcialmente observables*, *estocásticos*, *finitos*, *dinámicos* y *discretos*. Para ello, se van a utilizar dos formas nuevas de definir los problemas de planificación: Mediante *Procesos de Decisión de Markov (MDPs)* y mediante *Procesos de Decisión de Markov Parcialmente Observables (POMDPs)*.

1.2. Procesos de Decisión de Markov

Un Proceso de Decisión de Markov (MDP) [6, 7] se define como una tupla (S, A, T, R), en dónde:

- S es un conjunto finito de estados.

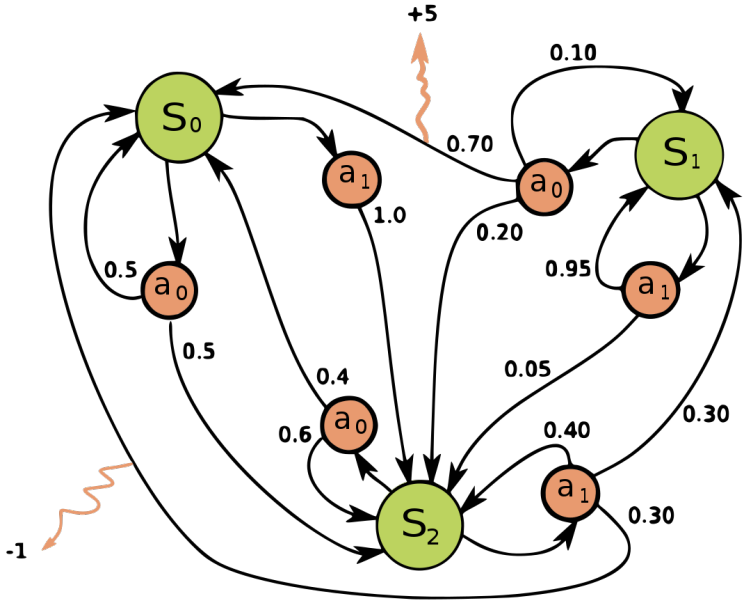

Figura 1: Un Proceso de Decisión de Markov

- A es un conjunto finito de posibles acciones.
- $T : S \times A \times S \to [0, 1]$ es la función de transición, dónde $T(s, a, s') = Pr(s'|s, a)$ representa la probabilidad de obtener el estado s' cuando se realiza la acción a partiendo del estado s.
- $R : S \times A \times S \to \mathbb{R}$ es la función de recompensa, dónde $R(s, a, s')$ es la recompensa obtenida tras ejecutar la acción a partiendo del estado s y alcanzando s'.

En los $MDPs$ no se define de forma explícita un objetivo o conjunto de estados finales, en su lugar se define una función de recompensa. Resolver un MDP significa encontrar un criterio o *política* para elegir acciones de tal forma que se maximice el límite del sumatorio de la función de recompensa en el infinito.

A modo de ejemplo, en la figura 1 se observa el diagrama de un MDP con tres estados S_0, S_1, S_2 y dos acciones a_0, a_1. Las probabilidades de la función de transición quedan representadas sobre las aristas del grafo, por ejemplo, la probabilidad de alcanzar el estado S_0 tras ejecutar la acción a_0 partiendo de S_1 es 0,7. La función de recompensa devuelve siempre cero salvo cuando se alcanza S_0 desde S_1 aplicando a_0 que devuelve un valor de 5 y cuando se alcanza S_1 aplicando a_1 desde S_2, devolviendo un valor de -1.

Una posible política para el anterior MDP podría ser la siguiente:

- Ejecutar siempre a_1 desde el estado S_0

- Ejecutar siempre a_0 desde el estado S_1
- Ejecutar siempre a_1 desde el estado S_2

1.3. Procesos de Decisión de Markov Parcialmente Observables

Un Proceso de Decisión de Markov Parcialmente Observable (POMDP) [6, 7, 3, 9] es una tupla (S, A, T, R, Z, O), en dónde:

- S es un conjunto finito de estados.
- A es un conjunto finito de posibles acciones.
- $T : S \times A \times S \to [0, 1]$ es la función de transición, dónde $T(s, a, s') = Pr(s'|s, a)$ representa la probabilidad de obtener el estado s' cuando se realiza la acción a partiendo del estado s.
- $R : S \times A \to \mathbb{R}$ es la función de recompensa, dónde $R(s, a)$ es la recompensa obtenida tras ejecutar la acción a partiendo del estado s.
- Z es un conjunto finito de posibles observaciones.
- $O : S \times A \times Z \to [0, 1]$ es la función de observación, dónde $O(s', a, z) = Pr(z|a, s')$ es la probabilidad de observar z cuando se ejecuta la acción a y el estado resultante es s'

Los $POMDPs$ son extensiones de los $MDPs$ en dónde no se conoce con certeza el estado actual, en su lugar el planificador debe construir y actualizar en tiempo de ejecución una distribución de probabilidad sobre el conjunto de estados S representando así su creencia (o *belief*) sobre el estado actual. En un $POMDP$, cada vez que se realiza una acción se desconoce el estado concreto que se alcanza, pero se obtiene una observación $z \in Z$ que permite actualizar el *belief*, es por ello que se denominan *parcialmente observables*. Nótese también que la función de recompensa no considera el estado alcanzado.

1.4. El problema del tigre

Uno de los ejemplos más clásicos de POMDPs es el problema del tigre [10] que se ilustra en la figura 2.

Imagina que te encuentras ante dos puertas, tras una de ellas hay un tigre hambriento, tras la otra un camino para escapar. Debes abrir una puerta, pero no sabes en qué puerta está el tigre. Como acciones puedes abrir una de las dos puertas o escuchar. Al escuchar puedes percibir que el tigre se encuentra a la izquierda o a la derecha. Pero hay que tener en cuenta que debido al estrés y los nervios, tus sentidos te pueden engañar y puedes escuchar al tigre en un lado cuando en realidad se encuentra al otro. Para colmo, el tigre se va impacientando y es capaz de romper la puerta al cabo de un tiempo. ¿Qué secuencia de acciones deberías realizar para escapar lo antes posible?

El problema del tigre se puede formalizar como un $POMDP = (S, A, T, R, Z, O)$ de la siguiente forma:

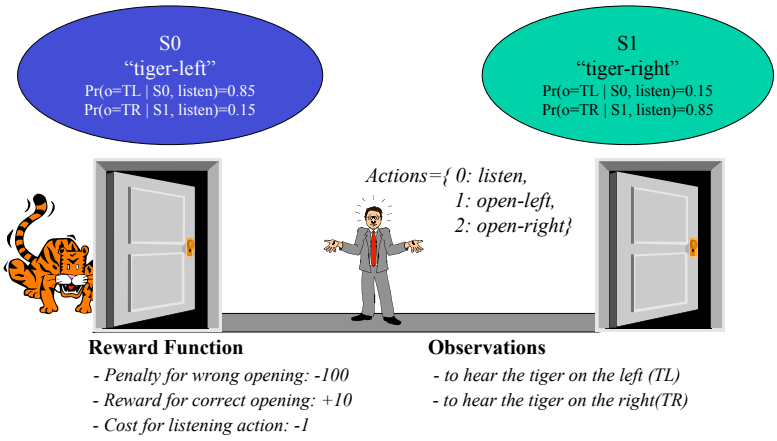

Figura 2: El Problema del tigre

- $S = \{tigre\_izquierda, tigre\_derecha\}$
- $A = \{escuchar, abrir\_izquierda, abrir\_derecha\}$
- $T(s, a, s') = 1$ si $s = s'$ para cualquier $a \in A$
- $T(s, a, s') = 0$ si $s \neq s'$ para cualquier $a \in A$
- $R(tigre\_izquierda, abrir\_izquierda) = -100$
- $R(tigre\_derecha, abrir\_derecha) = -100$
- $R(tigre\_izquierda, abrir\_derecha) = 10$
- $R(tigre\_derecha, abrir\_izquierda) = 10$
- $R(s, escuchar) = -1$ para cualquier $s \in S$
- $Z = \{rugidos\_a\_la\_izquierda, rugidos\_a\_la\_derecha\}$
- $O(tigre\_izquierda, escuchar, rugidos\_izquierda) = 0{,}85$
- $O(tigre\_izquierda, escuchar, rugidos\_derecha) = 0{,}15$
- $O(tigre\_derecha, escuchar, rugidos\_derecha) = 0{,}85$
- $O(tigre\_derecha, escuchar, rugidos\_izquierda) = 0{,}15$

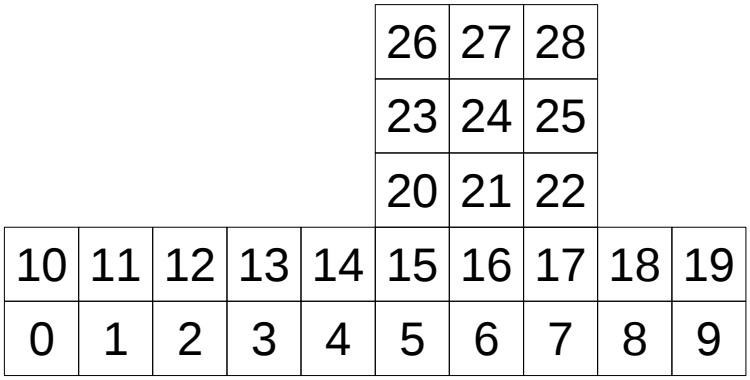

Figura 3: El mapa en el problema del tag

1.5. El problema del tag

El problema del *tag* [2] está basado en el popular juego del *lasertag*. En este problema un robot r_a se mueve intentando capturar a otro robot r_b. El mapa consta de 29 casillas y se representa en la figura 3. Inicialmente ambos robots se colocan en casillas seleccionadas aleatoriamente. El robot r_a no sabe dónde está el robot r_b (a no ser que se encuentren en la misma casilla), sin embargo, el robot r_b tiene conocimiento *omnisciente* y sabe en todo momento cuál es la casilla en dónde se encuentra el robot r_a. Las acciones del robot r_a son: *mover norte (N), mover sur (S), mover este (E), mover oeste (O) y capturar (C)*. Para ejecutar las primeras cuatro acciones, la casilla resultante no puede estar fuera del mapa; para ejecutar la última acción, ambos robots se tienen que encontrar en la misma casilla. El robot r_b sólo dispone de las primeras cuatro acciones (no puede capturar al robot r_a). El juego está sincronizado y en cada jugada, los dos robots realizan una acción de manera simultanea. El juego termina cuando el robot r_a captura al robot r_b o bien cuando se haya alcanzado un número prefijado de jugadas sin que se haya producido la captura.

El problema desde el punto de vista del robot r_a se puede plantear como un $POMDP = (S, A, T, R, Z, O)$ de la siguiente forma:

- $S = \{(r_a = i, r_b = j)\} \cup \{(r_a = i, r_b = \textit{tagged})\}$ siendo i el índice de la casilla ocupada por el robot r_a y j el índice de la casilla ocupada por el robot r_b para $0 <= i, j < 29$. Adicionalmente el robot r_b puede estar en estado *tagged* o capturado si el robot r_a ha ejecutado la acción *capturar (C)* en la misma casilla que el robot r_b, en cuyo caso el juego ha finalizado. Es decir, $29 \times 29 + 29 = 870$ posibles estados.
- $A = \{N, S, E, O, C\}$ las 5 posibles acciones del robot r_a descritas anteriormente.

- T = función de transición. Por una parte, la posición del robot r_a es completamente observable y el efecto de cualquier acción sobre su posición será considerado determinista, por ejemplo: $Pr(r'_a = 10 | r_a = 0, a = N) = 1$. Téngase en cuenta que si se ejecuta la acción *capturar (C)*, el robot r_a no se moverá de su casilla actual, por ejemplo: $Pr(r'_a = 10 | r_a = 10, a = C) = 1$. Por otra parte, la posición del robot r_b no es observable y habría que simular su algoritmo para la toma de decisiones, para simplificar vamos a considerar lo siguiente para cada jugada:
  - Si el robot r_a ejecuta la acción *capturar (C)* y ambos robots estan en la misma casilla, el estado del robot r_b pasa a *tagged* y el juego termina; en cualquier otro caso:
  - El robot r_b se aleja del robot r_a de acuerdo a la distancia de Manhattan con una probabilidad 0,8 (si hay varias posibilidades para alejarse, todas son equiprobables).
  - El robot r_b se mantiene en su posición con una probabilidad 0,2

  Por ejemplo:
  - $Pr(r'_b = 16 | r_a = 0, r_b = 15, a = N) = 0{,}4$
  - $Pr(r'_b = 20 | r_a = 0, r_b = 15, a = N) = 0{,}4$
  - $Pr(r'_b = 15 | r_a = 0, r_b = 15, a = N) = 0{,}2$
  - $Pr(r'_b = 16 | r_a = 0, r_b = 15, a = S) = 0{,}4$
  - $Pr(r'_b = 20 | r_a = 0, r_b = 15, a = S) = 0{,}4$
  - $Pr(r'_b = 15 | r_a = 0, r_b = 15, a = S) = 0{,}2$
  - $Pr(r'_b = 16 | r_a = 0, r_b = 15, a = E) = 0{,}4$
  - $Pr(r'_b = 20 | r_a = 0, r_b = 15, a = E) = 0{,}4$
  - $Pr(r'_b = 15 | r_a = 0, r_b = 15, a = E) = 0{,}2$
  - $Pr(r'_b = 16 | r_a = 0, r_b = 15, a = O) = 0{,}4$
  - $Pr(r'_b = 20 | r_a = 0, r_b = 15, a = O) = 0{,}4$
  - $Pr(r'_b = 15 | r_a = 0, r_b = 15, a = O) = 0{,}2$
  - $Pr(r'_b = 16 | r_a = 0, r_b = 15, a = C) = 0{,}4$
  - $Pr(r'_b = 20 | r_a = 0, r_b = 15, a = C) = 0{,}4$
  - $Pr(r'_b = 15 | r_a = 0, r_b = 15, a = C) = 0{,}2$
  - $Pr(r'_b = tagged | r_a = 15, r_b = 15, a = C) = 1$

  Así pues, $T(s, a, s')$ representa una distribución de probabilidad conjunta de la siguiente forma: $T((r_a, r_b), a, (r'_a, r'_b)) = Pr(r'_a | r_a, r_b, a) \times Pr(r'_b | r_a, r_b, a)$.
- $R(s, a) = -1$ para cualquier estado s y $a \in \{N, S, E, O\}$, es decir, las acciones de movimiento se penalizan.

- $R(s, C) = 10$ si s representa un estado en dónde ambos robots se encuentran en la misma casilla. Con lo cual, el robot r_a ha capturado con éxito al robot r_b, el estado del robot r_b pasa a *tagged* y el juego termina.
- $R(s, C) = -10$ si s representa un estado en dónde los robots están en casillas diferentes, es decir, se ha producido un intento de captura inválido.
- $Z = \{z = i\} \cup \{z = yes\}$ para $0 <= i < 29$, en dónde la observación *yes* significa que ambos robots se encuentran en la misma casilla (el robot r_a puede observar directamente a r_b), en cualquier otro caso, la observación z informa sobre la casilla en la que se encuentra el robot r_a.
- O = función de observación. por ejemplo:
  - $Pr(z = yes|a = N, r'_a = 10, r'_b = 10) = 1$
  - $Pr(z = yes|a = S, r'_a = 10, r'_b = 10) = 1$
  - $Pr(z = yes|a = E, r'_a = 10, r'_b = 10) = 1$
  - $Pr(z = yes|a = O, r'_a = 10, r'_b = 10) = 1$
  - $Pr(z = 10|a = N, r'_a = 10, r'_b = 5) = 1$

1.6. Algoritmos para resolver POMDPs

Lo explicado anteriormente en el apartado 1.3 e ilustrado con dos ejemplos no es más que la manera de representar un problema de planificación con incertidumbre mediante un POMDP. Una vez que se ha escrito la definición, existen diferentes algoritmos para resolverlo, es decir encontrar una secuencia de acciones que maximice el sumatorio de la función de recompensa. Existen dos aproximaciones [3]:

- Resolvedores *offline*: El algoritmo se ejecuta de forma previa resolviendo todas las posibilidades y determinando una política óptima.
- Resolvedores *online*: En dónde se mantiene en tiempo de ejecución una distribución de probabilidad sobre el estado real (conocida como *creencia* o *belief*), el planificador dispone de un tiempo limitado para determinar la siguiente acción a ejecutar, posteriormente se actualiza el *belief* con la observación obtenida.

Resolver un POMDP de forma *offline* es un problema intratable para instancias no triviales. Es por ello que la alternativa de los resolvedores *online* es mucho más popular, ya que son algoritmo en dónde la solución mejora cuánto más tiempo de computación se utilice. En [3] se puede leer una interesante recopilación de métodos. Cabe destacar el algoritmo Partially Observable Monte-Carlo Planning (POMCP) [4] que adapta la técnica de Monte-Carlo Tree Search (MCTS) [1] para la resolución de POMDPs (una variante de MCTS junto con Deep Learning es la que usan actualmente los programas AlphaGo y AlphaZero). Otro algoritmo popular es Point-Based Value Iteration PBVI[2].

1.7. Bibliotecas y software para resolver POMDPs

Existe una gran cantidad de software y bibliotecas para especificación y resolución de POMDP, en la web `http://www.pomdp.org/` se puede encontrar una extensa introducción, así como un formato de fichero para la especificación `http://www.pomdp.org/code/pomdp-file-spec.html`.

En Python encontramos, por ejemplo:

- POMDPy: `https://github.com/pemami4911/POMDPy`
- PyPOMDP: `https://github.com/namoshizun/PyPOMDP`

El autor del enunciado de este trabajo también es autor de una biblioteca en C++ para la resolución de POMDPs mediante el algoritmo POMCP, que se puede descargar bajo licencia LGPL-3.0 aquí: `https://github.com/Ignacio-Perez/lightpomcp`

2. Descripción del trabajo

2.1. Problemas a resolver

Será necesario seleccionar 4 problemas de planificación interesantes y plantearlos como POMDP, siendo necesariamente dos de ellos el problema del tigre y el problema del tag, los otros dos problemas adicionales se pueden elegir de entre los que se enumeran aquí: `http://www.pomdp.org/examples/`. El fichero de definición para el problema del tigre y el problema del tag se puede descargar de los siguientes enlaces:

- Problema del tigre: `https://github.com/Ignacio-Perez/lightpomcp/blob/master/models/tiger.pomdp`
- Problema del tag: `https://github.com/Ignacio-Perez/lightpomcp/blob/master/models/tag.pomdp`

2.2. Software a desarrollar

El alumno deberá desarrollar un software en Python que tenga lo siguiente:

2.2.1. Entrada

El programa tendrá un interfaz de usuario (que podría ser gráfico o de texto) que permita seleccionar lo siguiente:

- Problema a resolver:
  - Problema del tigre
  - Problema del tag
  - Problema adicional 1 (poner aquí el nombre del primer problema adicional elegido)

    - Problema adicional 2 (poner aquí el nombre del segundo problema adicional elegido)

- Algoritmo resolvedor:
    - POMCP
    - PBVI

- Parámetros del algoritmo: Dependiendo del algoritmo elegido, se permitirá al usuario introducir sus parámetros o dejar parámetros por defecto.

- Simulación interactiva, silenciosa o benchmark: Se permitirá al usuario elegir el tipo de simulación, que podrá ser *simulación interactiva* en dónde se visualizará el estado paso a paso; *simulación silenciosa* que se ejecutará hasta un estado de parada y mostrará por pantalla el número de pasos y la recompensa acumulada o *benchmark* en donde se realizarán 30 simulaciones silenciosas mostrando los resultados al finalizar cada simulación y calculando valores medios y desviaciones típicas tanto para el número de pasos como para la recompensa acumulada.

2.2.2. Ejecución

Una vez introducida la entrada, se preguntará confirmación al usuario y en el caso de lanzar una simulación interactiva del problema, se seleccionará aleatoriamente un estado inicial para el problema que se representará mediante texto o de forma gráfica, por ejemplo, en el problema del Tag, se podría usar un diagrama para dibujar el mapa e indicar en dónde se encuentra cada robot. También se podrían usar caracteres ASCII para dicha representación. Se pedirá al usuario que pulse alguna tecla o algún botón. Inmediatamente se ejecutará el algoritmo seleccionado para elegir la siguiente acción (se permiten usar otras bibliotecas de Python para el resolvedor o una implementación propia), se mostrará por pantalla la acción, el reward acumulado y se actualizará la visualización del estado de acuerdo a las probabilidades de transición. En cualquier momento se permitirá salir de la simulación. La simulación se detiene si se llega a un estado final (por ejemplo, abrir una puerta en el problema del Tigre o bien capturar al robot oponente en el problema del Tag o llegar a un número de jugadas, el estado final es propio de cada problema).

2.3. Experimentación a realizar

Se realizará para cada uno de los 4 problemas una batería de 30 simulaciones con cada uno de los dos algoritmos de forma silenciosa hasta estado de parada y se calculará valor medio y desviación típica del número de pasos ejecutados y la recompensa acumulada. Se deberán documentar con tablas y/o gráficos los resultados, incluyendo una pequeña discusión sobre los mismos en su correspondiente sección de la documentación.

2.4. Documentación a entregar

Se ha dejado una plantilla de ejemplo para documentar el trabajo (no es obligatorio su uso, pero se recomienda). Se trata del fichero *plantilla-trabajo.doc*, en el que se puede ver la estructura de un trabajo de investigación. Este formato se detalla en su guía para autores [8], con plantillas para Word y Latex. Las secciones a incluir son: Resumen, Introducción, Preliminares, Metodología, Resultados, Conclusiones, Referencias y Anexos.

Como anexos se incluirá la explicación de los dos problemas POMDP adicionales (la explicación del problema del Tigre y del problema del Tag no es necesaria ya que se incluye en este documento, aunque se deberán añadir las correspondientes referencias) y una breve documentación del código entregado (no incluir el propio código fuente en la documentación ni en los anexos).

2.5. Presentación, demo y defensa

El día de la defensa se deberá realizar una pequeña presentación de 10 minutos (si el grupo se compone de dos alumnos, 5 minutos cada uno). En esta presentación seguirá la misma estructura que el documento, y se hará especial mención a los resultados obtenidos. Se podrá usar un portátil (personal del alumno) para realizar una demo, diapositivas y/o pizarra. En los siguientes 10 minutos de la defensa (20 en total), el profesor procederá a realizar preguntas sobre el trabajo, que podrán ser tanto del documento como del código fuente.

3. Criterios de evaluación

Para que el trabajo pueda ser evaluado, se deberán satisfacer los objetivos descritos en el apartado 2 para al menos un problema POMDP y un algoritmo resolvedor (en caso contrario, el trabajo obtendrá una nota de suspenso). El alumno debe entregar un fichero comprimido .zip que contenga:

- Una carpeta con el código fuente, el código deberá estar debidamente comentado. Dentro de dicha carpeta tiene que haber un fichero README.txt que resuma la estructura del código fuente e indique cómo usar la interfaz, con ejemplos. Asimismo se deberá indicar como reproducir los experimentos realizados. Es importante la coherencia de este fichero con la defensa.
- El documento, en formato PDF, deberá tener una extensión mínima de 8 páginas y máxima de 10 sin incluir los anexos, además deberá incluir toda la bibliografía consultada (libros, artículos, technical reports, código fuente de terceros, diapositivas, etc), todas las referencias que se incluyan deben de estar mencionadas en la documentación.

Para la evaluación se tendrá en cuenta el siguiente criterio de valoración, considerando una nota máxima de 3 en total para el trabajo:

- Implementación (1,5 puntos): se valorará la claridad y buen estilo de programación, corrección y eficiencia de la implementación, así como calidad de los comentarios. La claridad del fichero README.txt también se valorará. En ningún caso, se evaluará un trabajo con código copiado directamente de internet o de otros compañeros. Sólo se permite usar código de internet para los algoritmos resolvedores y procesamiento de ficheros de especificación de POMDP, las fuentes deberán quedar debidamente referenciadas en la documentación.

- El documento - artículo científico (1,0 puntos): Se valorará positivamente el uso de la plantilla para seguir el formato de artículo científico u otros estilos como LNCS, SPIE, etc. Se valorará la calidad de los experimentos, el razonamiento de ellos, los resultados alcanzados y el uso del lenguaje. Se podrá realizar el castellano o en inglés. Igualmente no se evaluará el trabajo si se detecta cualquier copia de contenido, incluyendo *copiar y pegar* del presente documento.

- La presentación y defensa (0,5 puntos): Se valorará la claridad de la presentación y la buena explicación de los contenidos del trabajo. También será importante responder correctamente a las preguntas realizadas por el profesor sobre cualquier aspecto del trabajo.

- Extras (0,5 puntos, sin superar la nota máxima): Desarrollar una implementación propia de alguno de los algoritmos resolvedores o un interfaz gráfico. Se permite la posibilidad de incorporar cualquier otra mejora en el trabajo tras consultar con el profesor al menos dos semanas antes de la entrega.

IMPORTANTE: Cualquier **plagio, compartición de código** o uso de material que no sea original y del que no se cite convenientemente la fuente, significará automáticamente la **calificación de cero** en la asignatura para **todos los alumnos involucrados**. Por tanto, a estos alumnos **no se les conserva**, ni para la actual ni para futuras convocatorias, **ninguna nota** que hubiesen obtenido hasta el momento. Todo ello sin perjuicio de las correspondientes **medidas disciplinarias** que se pudieran tomar.

Referencias

[1] C. Browne et al. A survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4, 1 (2012).

[2] J. Pineau, G. Gordon and S. Thrun. Point-based Value Iteration: An Anytime Algorithm for POMDPs. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03), pages 1025–1032, Acapulco, Mexico, 2003. `https://www.cs.mcgill.ca/~jpineau/files/jpineau-ijcai03.pdf`

[3] S. Ross and J. Pineau. Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32 (2008) 663–704.

[4] D. Silver, J. Veness. Monte-Carlo Planning in Large POMDPs. *Advances in Neural Information Processing Systems*, 23 (2010) 2164–2172.

[5] J.L. Ruiz Reina et al. *Planificación*. Apuntes de la asignatura Inteligencia Artificial (IS) 2018 `https://www.cs.us.es/cursos/iais-2018/temas/Planificacion.pdf`

[6] S. Russel and P. Norvin. *Artificial Intelligence (A Modern Approach)* Precinte-Hall (2010). Third Edition. Capítulos 3, 10, 17.

[7] S. Thrun, W. Burgard and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)* The MIT Press (2005). Capítulos 15, 16.

[8] Plantilla IEEE. `https://www.ieee.org/conferences_events/conferences/publishing/templates.html`

[9] The POMDP Page `http://www.pomdp.org/`

[10] POMDP Tutorial `https://www.techfak.uni-bielefeld.de/~skopp/Lehre/STdKI_SS10/POMDP_tutorial.pdf`