Inteligencia Artificial

Pau Soler Valadés

Otoño 2019

Índex

1 ¿Qué es la Inteligencia Artificial?

2 Agentes

Definición 2.1. Un **Agente** percibe su entorno mediante sensores y actúa sobre o en él mediante actuadores.

Análogamente a un humano, los ojos, oídos y algunos órganos serian los **sensores** y los brazos, manos, voz los **actuadores**.

Definición 2.2. Llamamos **Percepciones** a los inputs de los sensores en un instante concreto.
El conjunto de percepciones totales de un agente se denomina **secuencia de percepciones**

Matemáticamente podemos ver el comportamiento de un agente como una función.

Definición 2.3. La **Función del Agente** f relaciona una o diversas percepciones $P*$ en una acción A.

$$f : P* \to A$$

La función del Agente será implementada por el **Programa del Agente**. La función es la descripción del comportamiento deseado, mientras que el programa es como se implementa este comportamiento.

2.1 Racionalidad

Un **agente racional** es el que hace lo correcto. ¿Cómo sabemos qué es hacer lo correcto? Analizando las consecuencias y actuando acorde.

Cuando un agente analiza un entorno, genera una secuencia de acciones acorde con lo que recibe de este y el agente actua sobre él, modificando su estado. Si la secuencia que el agente ha generado es adequada, significa que el agente ha tenido un buen **rendimiento**. Midiendo el **ratio de éxito** del agente en el entorno, podemos **evaluar** su efectividad.

Evidentemente, no hay una sola evaluación correcta para cada tarea, se debe tener en cuanta en el diseño cuales se adequan más en nuestro caso. Por ejemplo: supongamos una aspiradora que tenga que limplar la suciedad de una habitación. Si definimos la evaluación del rendimiento como " limpiar máximo por hora", el agente sera perfectamente racional y tendrá una tasa de éxito altísima limpiando la suciedad, tirándola al suelo y volviéndola a limpiar. Este problema se puede evitar con una mejor definición de objetivos, como "elimina toda la basura".

Teniendo todo lo dicho en cuenta, podemos definir un agente racional:

Definición 2.4. Un agente es **racional** si para cada secuencia de percepción dada, el agente seleciona una acción con expectativa de maximizar su tasa de éxito teniendo en cuenta la misma secuencia de percepción y el conocimiento previo del agente.

3 Problemas de Búsqueda

En el apartado anterior hemos tratado con **agentes reflexos**, los cuales basan sus acciones en un mapeado directo de estados a acciones. Estos agentes dejan de tener utilidad cuando el problema escala demasiado rápido en tamaño a almacenar, tiene muchos estados o tardaría demasiado en encontrar las respuestas correctas. A partir de ahora este documento va a tratar de **Agentes Basados en Objetivos**

Definición 3.1. Los **Agentes Basados en Objetivos** son agentes que usan estados simples, sin estructura interna, son sólo representaciones de sí mismos.

Propiedades. *Agentes Basados en Objetivos*

1. Planifican antes de actuar según un modelo interno de cómo cambiará el mundo basado en sus acciones.
2. Buscan secuencias de acciones que conducen a los estados deseados, sus **objetivos** o **metas**.
3. Son racionales siguiendo la definción dada en el apartado anterior.

Una vez definidos los agentes que usaremos, vamos a centrarnos en qué maneras podemos resolver problemas. Para empezar, debemos aprender a formalizarlos.

3.1 Etapas de la Resolución de Problemas con Objetivos

Las etapas las podemos resumir en las siguientes 4:

- **Formulación de Objetivos:** definir los estados objetivo, inical y los factores que pueden influir en el grado de satisfacción.
- **Formulación del Problema:** decidir qué acciones y qué estados considerar para cumplir con los objetivos. Este paso necessita definir:
  - **Estados:** Todos los possibles estados en los que el agente puede estar.
  - **Acciones:** Qué acciones puede realizar en el mundo en el que se encuentra.
  - **Función Sucesor:** Dado un estado s y una acción a, devuelve los estados possibles. $f_s(s, a) = \{s'_1, ..., s'_n\}$.
- **Búsqueda de la Solución:** calcular la mejor secuencia de acciones que llevan a algún estado objetivo des de el inicial.
- **Ejecución:** El agente ejecutará las acciones calculadas.

Ejemplo: Vacaciones en Magaluf

Imaginemos que nuestro agente está en Magaluf, disfrutando de sus vacaciones. Como está de vacaciones, el rendimiento de este depende de factores como el bronzeado, mejorar su Alemán, no caerse por el balcón, salir de fiesta y minimizar las resacas. Ahora mismo el problema que ha de solucionar es complejo y difícil de formalizar.

Ahora supongamos que el agente tiene un billete para marcharse a casa desde Palma. Si no llega a Palma a tiempo, perderá el dinero y no podrá volver a casa. Ahora el **objetivo ha sido formulado** clara y concisamente: toda actividad que no implique llegar a Palma es descartada imediatamente, y el problema de decisión del agente se ha simplificado.

Formulación de Objetivos: *Definir los objetivos correctos permite organizar el comportamiento y enfocar mejor el problema, simplificarlo y hacerlo útil.*

Ahora que hemos definido el estado objetivo y el estado inicial, debemos decidir **qué acciones puede realizar el sujeto**. Si las acciones para viajar a Palma desde Magaluf son "Desplaza tu pie 30 centímetros"o "gira el volante 30 grados"nuestro agente no va a llegar nunca porque hay tanto nivel de detalle en esas acciones que produce **incertidumbre**.

Sabemos que nuestro agente tiene coche y sabe conducir, así que las acciones más útiles serian conducir hasta otra ciudad, siendo las ciudades possibles de Mallorca los estados, conducir de una ciudad a otra la acción y la función sucessor siendo las carreteras que conectan los pueblos.

Formulación del Problema: *Definir las acciones relevantes para el contexto ayuda a simplificar más el problema, pudiendole dar así una solución más sencilla.*

Suponemos que el Agente tiene un mapa de Mallorca. Puede conducir hasta Palmanova, Torrenova o La Torrassa. A medida que lo mira consigue encontrar un camino para así coger el avión a tiempo. El agente con múltiples opciones de valor desconocido puede decidir qué hacer primero examinando futuras acciones que le llevaran hasta un estado de valor conocido. Pero aquí hemos realizado alguna que otra asunción: asumimos que el entorno es **observable** (el agente sabe siempre su posición), asumimos que el entorno es **discreto** (desde un estado dado hay un número finito de elecciones), asumimos que conocemos el entorno (porque hemos assumido que el agente tiene un mapa) y asumimos que el entorno es **determinista** (cada acción tiene una sola salida).

Bajo todas esas asunciones podemos decir que la solución a este problema es una secuencia concreta de acciones. Si os parecen demasiadas y que evidentemente todo problema bajo esa axiomática tiene solución, permitidme deciros que no. Sin salir del mismo ejemplo, podría pincharse la rueda del coche o que una carretera esté cortada y necessitemos

una ruta alternativa entre dos ciudades. Pero como el agente **percive un entorno parametrizado y determinista siempre que termina una acción** puede variar sin tener que hacer las dos últimas assumpciones.

> ***Búsqueda de la Solución:*** *Buscar es encontrar la secuencia de acciones que llegue a la meta. En nuestro caso, esto lo consigue un algorismo de búsqueda. Y no nos olvidemos que pese a tener la solución, aún queda conducir hasta Palma, siendo esto la* ***ejecución****.*

Ahora, ¿qué maneras tenemos de encontrar la ruta hasta Palma? Aquí es cuando entran en juego los algorítmos de búsqueda informada y no informada.

3.2 Denominaciones Análogas IA vs. Algorismica

Todo problema de búsqueda de puede expressar gráficamente como un grafo. En este grafo los vértices (nodos) són los possibles estados y los arcos (edges) representa la función sucessor. Estos grafos son tan grandes que en contadas ocasiones de puede contruir en memoria. Aquí yace el *quid* de la **diferencia entre la algorísmica y la IA**:

En la primera dado un grafo, investigamos maneras de recorerrlo entero y analizamos cual es más adecuado para cierto tipo de exploraciones. En canvio en IA, el grafo no viene dado, lo vamos construyendo a medida que avanzamos y tampoco lo vamos a construir todo, sólo queremos encontrar la primera solución possible. A continuación daremos unas definiciones para introducir los algoritmos de búsqueda:

Definición 3.2. *Algoritmos*

- **Espacio de Estados:** Lugar conceptual dónde se realiza la búsqueda.
- **Expandir:** Acción de saber los vecinos de un nodo mediante la función sucesor. Análogo a explorar.
- **Frontera:** Lista de nodos visitados que tienen que ser expandidos. Análogo a pendientes de visitar.
- **Nodos Cerrados:** Nodos que ya se han expandido. Se usa para evitar cíclos. Análogo a visitados.

3.3 Búsqueda No Informada

Definición 3.3. Un **Algoritmo de Búsqueda No Informada** (o **algoritmo de búsqueda ciega**) es un algoritmo que no recive más información que la descripción del

problema. Sólo puede generar sucesores y distinguir entre un estado objetivo y uno que no lo és.

Al no tener ninguna información extra sobre el problema, los resultados seran *a priori* menos óptimos que con algoritmos de búsqueda informada, pero ganamos en generalidad, resultando algoritmos más verátiles y aplicables a muchos más problemas.

Su clasificación se distingue por el orden en que se exploran estas soluciones.

3.3.1 Búsqueda Primero en Anchura

El BFS prioriza la expansión a los nodos cuya profundidad sea menor.

Implementación

La implementación de la frontera debe ser una cola FIFO (*First-In First-Out*). Es decir, una lista.

[TODO QUE NO TODO: UNA TAULA DE COM ES FA UN BFS COM VOLEN AQUESTS PRIMMIRATS]

3.3.2 Búsqueda Primero en Profundidad

El DFS prioriza la expansión a los nodos cuya profundidad sea mayor.

Implementación

La frontera debe ser una lista LIFO (*Last-In First-Out*). Es decir, un stack (pila).

[TODO QUE NO TODO: UNA TAULA DE COM ES FA UN DFS COM VOLEN AQUESTS PRIMMIRATS]

3.4 Búsqueda Informada

Definición 3.4. Un **Algoritmo de Búsqueda Informada** es un algoritmo que recive (aparte de la descripción del problema) información característica sobre el problema que estamos tratando.

Estos tipos de algoritmos encuentran soluciones mucho más rápido que los de búsqueda no informada a costa de individualizar el problema. La primera aproximación que tomaremos es la búsqueda del **primero-mejor**: esta filosofía dicta que el nodo que se selecione para expandir dependerá de una función evaluación:

Definición 3.5. Una **Función Evaluación** $f(n)$ es la función que evalua los estados de un problema. No es única para todos los problemas: variará en función de ellos. La elección de esta determinará qué estrategia de búsqueda se debe seguir.

En *primero-mejor* el nodo con la menor evaluación será el primero seleccionado. Pero si intoducimos más información sobre el problema, damos con el concepto de **heurística**:

Definición 3.6. Una **Función Heurística** $h(n)$ es el coste estimado del camino menos costoso desde un nodo cualquiera (nodo n) hasta el nodo objetivo.

Propiedades. *Función Heurística*

Más adelante las trataremos en profundidad, pero de momento daremos las siguientes propiedades:

- Son arbitrarias y no generales: dependen de cada problema.
- Siempre de debe cumplir que $h(n) > 0$, y si n es un nodo objetivo $h(n) = 0$.

3.4.1 Búsqueda Voraz Primero-Mejor

La búsqueda voraz con primero-mejor intenta expandir el nodo que está más cerca de la meta, el que parece que va a hacernos llegar a la solución con menos coste. Dicho de otra manera, evalua los nodos únicamente minimizando la heurística, así que $f(n) = h(n)$.

Implementación

La frontera tiene que ser una cola prioritaria, ordenada por orden creciente de la heurística: se sacan primero los nodos con menor heurística.

Ejemplo

[TODO QUE NO TODO: POSAR EXEMPLE POWER POINT]

Este algorismo no es óptimo, puede tener búcles infinitos provocados por la ambición del mismo: a cada paso intenta estar más cerca de la meta, en vez de hacer un análisi del problema y tener una buena heurística.

3.4.2 Búsqueda A*

A* (pronunciado "A-estrella") es el algorismo de primero-mejor más conocido y óptimo con la heurística adecuada. Este evalua los nodos combinando el coste para llegar al nodo n ($g(n)$) y el coste de llegar al nodo desde la meta, la heurística:

$$f(n) = g(n) + h(n)$$

Con esta configuración, $f(n)$ representa el coste estimado de la solución más óptima hacia el nodo n. Intuitivamente, si estmos intentando encontrar la solución más óptima, tendríamos que escoger el nodo que minimize $g(n) + h(n)$. Resulta que no solo intuitivamente esto es correcto. Matematicamente, si la heurística cumple una serie de requisitos (de los que hablaremos más adelante), A* es NP-completo y óptimo.

Ejemplo:

[TODO QUE NO TODO: POSAR EXEMPLE POWER POINT]

Optimalidad

Para que A* sera óptimo la heurística debe ser **admissible**:

Definición 3.7. Una **Heurística Admisible** es aquella que nunca sobreestima el coste para llegar a la meta. És decir: $h(n) \geq g(n)$.

Las heurísticas admisibles se tildan también de **Optimistas**. Es decir: piensan que el coste de entcontrar la solución es menos del que es. Para entender esto usemos el contrarecíproco: si sobreestimamos el coste para llegar a la meta, significa que $h(n) \leq g(n)$, así que no haria falta que exisitiera $h(n)$ porque podríamos usar directamente el coste $g(n)$ en su lugar.
Un gran ejemplo es la heurística del ejemplo de A*, la linea recta entre dos puntos es siempre la distancia más corta, así que estés en la ciudad que estés, $h(n) \geq g(n)$ mostrando su optimismo.

La otra condición para la optimalidad de A* en búsqueda de grafos es que la heurística debe ser **consistente**:

Definición 3.8. Una **Heurística Consistente** es aquella que para cada nodo n y cada sucesor n' de n generada por cualquier acción a, el coste estimado no sea mayor que la suma de los costes de llegar hasta n' desde n ($c(n, a, n')$) y el coste estimado hasta el nodo objetivo (h(n')).

$$h(n) \leq c(n, a, n') + h(n')$$

Se puede observar que esto es una forma de *desigualdad triangular*, que estipula que la suma de los lados siempre será más pequeña o igual que la hipotenusa. La explicación de por qué se cumple esto es sencilla: si del nodo n a G_n hay un camino n' menos costoso

que $h(n)$, estaríamos violando que la heurística es siempre menor al coste, es decir, la **condición de admisibilidad**.

Teorema. *La búsqueda A* en árbol es óptima si y sólo si la heurística $h(n)$ es admisible.*

Demostración (TODO NO TODO: DEMO DE LA OPTIMADIDADA, ESQUE NO ENTRA)**.**

Teorema. *La búsqueda A* en grafos es óptima si y sólo si la heurística $h(n)$ es consistente.*

Demostración (TODO NO TODO: DEMO DE LA OPTIMADIDADA, ESQUE NO ENTRA)**.**

4 Búsqueda En Juegos

En este apartado trataremos los algoritmos más eficientes para que una IA juegue a un **juego** en entornos con **múltiples agentes**. Su objetivo será calcular una **política** para decidir qué movimiento realizar en cada estado. Pero, para empezar, ¿qué es un juego en IA?

Definición 4.1. Un **Juego** se define como un entorno *competitivo* donde los objetivos de los agentes entran en *conflicto*: ambos quieren ganar.

En el campo de IA, classificamos los juegos según la información que tenemos disponible y los resultados de nuestras acciones. Deterministas vs. Estocásticos, con información perfecta vs información imperfecta y de suma zero o no suma zero.

Definición 4.2. Un juego **Determinista** es un juego completamente predecible, en el que el la acción a en el estado e siempre dará el mismo conjunto de estados sucesores E'. $f_s(a, e) = \{e'_1, ..., e'_n\} = E'$.
Al contrario, un juego no determinista o **Estocástico** es un juego en el que el siguiente estado dados una acción y estado concreto depende de un factor no completamente controlable. Depende de una **probabilidad**.

Definición 4.3. Un juego de **Información Perfecta** es en el que los jugadores, de cualquier acción tomada en cualquier estado, pueden conocer todas los estados a los que eso lleva. Dicho de manera más informal, se pueden saber todas las consequencias que implica cada acción.
En cambio, un juego de **Información Imperfecta** es uno dónde no podemos saber todas o ninguna de las consequencias que tendrá cierta acción en cierto estado.

Definición 4.4. Un juego de **Suma Zero** es en que los beneficios o las pérdidas de un jugador son exactamente las pérdidas o beneficios del otro. Lo que uno gana, lo pierde el otro.

4.1 Juegos Deterministas

El tipo de problemas más comunes en IA son los *deterministas con información perfecta*: entornos completamente observables en los que cada acción implica consequencias que podemos evaluar. Pero no todos estos juegos son interessantes, pongamos por ejemplo un cubo de rubik.
Un cubo de rubik es un juego determinista con información perfecta de un jugador con

el objetivo de resolver el cubo. Estos tipos de problemas se resuelven solamente con una búsqueda informada que hemos tratado en el apartado anterior. En este problema, cada nodo del grafo de estados representa un valor: el mejor que podremos conseguir si vamos por esa rama.

Cojamos ahora uno de los juegos más famoso en IA: el Ajedrez. El ajedrez es un juego determinista con información perfecta de dos jugadores de suma zero.

El ajedrez es de suma zero porque cuando un jugador mata una pieza, el otro la pierde. Hay muchos otros juegos multijugador que tienen esta característica, el tres en ralla, las damas, el Go y la lista sigue.

Los problemas de la *búsqueda en juegos son interessantes por lo intratables que son.* Volviendo al ajedrez, cada estado (posición de las piezas en el tablero) tiene, de media, un factor de ramificación de 35 y suponiendo que en cada partida cada jugador realiza 50 movimientos, tenemos 35^{100} nodos en el árbol. Para hacernos una idea de lo grande que es ese número, ni aún que todos los átomos del universo estuviesen evaluando los nodos por milisegundo desde el principio del mismo, no podríamos tener una solución. Las **políticas** que se han desarrollado para hacer estos problemas tratables son lo que vamos a explicar a continuación.

4.1.1 Minimax

4.2 Juegos Estocásticos

Recordemos que los juegos estocásticos son aquellos que contienen un elemento aleatorio. Como ejemplo principal tenemos el Backgammon. Al principio del juego se tira un dado para determinar los movimientos legales en esa partida. Cuando un jugador ya ha tirado el dado, no hay manera de saber qué possibles movimientos tendrá el otro, ya que tendrá que tirar el dado.

Para construir el árbol en ese caso, debemos incluir **nodos probables** junto con los máximos i mínimos del minimax. És decir, construimos todos los nodos possibles con la probabilidad de que pasen. En cuanto el otro jugador tire el dado, podamos los que no representan lo que ha pasado.

Como no sabemos el valor que pueden tener esos nodos, no podemos evaluar mínimos y máximos. En cambio, podemos evaluar el

4.2.1 Expectiminimax

ε, ϵ

5 Problemas de Decisión Secuenciales

En este apartado nos concentraremos en **Problemas de Decisión Secuenciales**, en los que el Agente Inteligente depende de una *secuencia de decisiones*. Estos problemas conllevan incertidumbre, análisis del entorno y objetivos que tendremos que resolver o sobrellevar.

Problema del Agente Rejilla

Supongamos que nuestro agente está situado en el entorno de la figura. Cuando el agente alcanza la casilla con un +1 o un -1, termina la interacción con el entorno.

[TODO FIGURE 17.1 AIAMA]

Igual que en los *problemas de búsqueda*, las acciones disponibles para cada agente en un estado viene dada por la función $A(s)$, donde s es un estado. También assumimos que el agente sabe siempre dónde está (ambiente completamente observable). Pero nuestro problema es **estocástico**, así que la sencilla secuencia de soluciones deterministas [arriba, arriba, izquierda, izquierda, izquierda] no es possible, ya que **las acciones que el agente puede realizar no son consistentes** ni siempre las mismas.

Para resolver este problema podemos hacer que el agente siga un modelo como el de la figura (b): un 0.8 de las veces va arriba, un 0.1 a la izquierda y otro 0.1 a la derecha, siendo esos números la probabilidad de que el hagente realize dicha acción. Pero ahora la *función sucesor* no sirve para determinar qué estados tenemos disponbiles dada una acción a i un estado s para llegar a s', así que lo escribimos como la probabilidad condicionada de que dados un estado y una acción s, a passe el estado a': $P(s'|s, a)$. Eso último solo passa si nuestro problema (un cierto proceso estocástico) cumple con la **propiedad de Markov**, que es que ese proceso " Carezca de memoria"

Definición 5.1. La **Propiedad de Markov** es la propiedad de ciertos procesos estocásticos de que la distribución de probabilidad de una variable aleatoria depende solamente de su valor presente, siendo independiente de la historia de esa variable. Matemáticamente:

$$P(S_{t+1}|S_t, a_t, S_{t-1}, a_{t-1}, ..., S_0, a_0) = P(S_{t+1}|S_t, a_t)$$

En nuestra secuencia de decisión estocástica esto significa que $P(s'|s, a)$ depende solo del estado s y no de como ese agente ha pasado por otros estados antes.

Para terminar, solo nos falta dar un incentivo al agente por explorar y otro para evaluar su rendimiento. Como estamos en un problema secuencial, la *función evalaución* f_e no puede evaluar solo un estado, sinó que tiene que **evaluar una secuencia de ellos**. Lo

que si podemos poner es un incentivo para que el agente explore, una **Recompensa**. Por cada estado s que el agente visite, recibirá una recompensa $R(s)$. La naturaleza de esta puede ser positiva, negativa, grande o pequeña pero **debe ser acotada**:
Supongamos una *recompensa* de -0.04 en todos los estados (excepto los terminales, los cuales es -1 o 1) y que f_e es la suma de todas las recompensas recibidas. Si por ejemplo el agente llega al +1 en 10 turnos, $f_e(s_t) = -0.04 \times 10 + 1 = 0.6$. Nuestra recompensa negativa ha promovido que el agente investigase, que no estuviese cómodo en la posición en la que se encontraba. En cambio si nuestra $R(s) > 0$ (como se ve en la figura) no incentivamos a nuestro agente a moverse, así que rehúye los estados terminales. También en esa figura se ven otros valores de $R(s)$ y sus mismas consequencias en el agente.

Esto que hemos descrito es un **proceso de decisión de Markov**.

5.1 Proceso de Decisión de Markov

Recapitulando desde el ejemplo anterior, podemos enumerar las características que tiene un **Proceso de decisión de Markov (MDP)**.

Definición 5.2. Un **MDP** es un problema de decisión secuencial estocástico cumpliendo la propiedad de Markov que tiene:

- Conjunto de estados S.
- Conjunto de acciones A.
- Función de Transmision $T : A \times S \times S \to [0, 1]$ donde $T(a, s, s') = P(s'|a, s)$
- Función de recompensa $R : S \times A \times S \to \mathbb{R}$ (Se abrevia por $R(s)$)
- Estado Inicial y (a veces) Estado Terminal.

Una vez dado un PMD, debemos definir una **solución** a este.

Definición 5.3. Una **Política** $\pi : S \to A$ especifica qué debe hacer el agente para cualquier estado que este pueda alcanzar.

Una política da solución a un MDP dónde $\pi(s)$ es la acción recomendada por la política en un estado s. Una política se dice **completa** si $\forall s \in S \; \exists a \in A : \pi(s) = a$.

Una política que dé solo la mejor solución dado cualquier estado es una **política óptima**, y se denota por π^*. Una política representa al completo una *función de agente* y es la descripción de un agente reflejo.

5.2 Utilidad de una Política

En el ejemplo anterior, el rendimiento del agente estaba medido por la suma de las recompensas de los estados visitados, pero puede ser evaluada de muchas otras maneras; a partir de ahora escribiremos la función utilidad como $U_h([s_0, ..., s_n])$, donde s_i son los estados visitados por nuestro agente.

Primero supongamos que tenemos un **horizonte finito**, que el juego termina en un tiempo N fijado, o dicho de otra manera, $U_h([s_0, ..., s_N])$ es el total de la evaluación [*NOTA: No confundir horizonte (in)finito con secuencias de ACCIONES (in)finitas. Las primeras determinan un tiempo máximo en el que se permite jugar, el segundo describe las acciones que sigue el agente para llegar (o no) a un estado terminal.*]. Esto hace que el agente tenga "Prisa"para llegar al estado terminal cuando N es pequeña y que cuando N sea significativamente mayor pueda dar más rodeos. Esto degenera en que la política del agente se vuelve **no-estacionaria**:

Definición 5.4. Una política **no-estacionaria** es que, en un horizonte finitio, la acción optima en un estado dado puede variar respecto el tiempo.

En canvio, diremos de una **Política Estacionaria** la que se produce cuando no hay límite de tiempo, ya que sólo depende del estado actual.

Deducimos de lo anterior que los **horizontes infinitos** (en los que no tenemos un límite de tiempo o acciones) nos daran problemas más sencillos de tratar. Siguiendo en esta línea, consideramos preferencias estacionarias sobre secuencias de estados para calcular su utilidad:

Definición 5.5. Una secuencia se dice **estacionaria** cuando dadas dos secuencias de estados/recompensas empiezan con el mismo estado/recompensa, tendran un solo orden: el mismo que tenian cuando empezaban con el mismo estado/recompensa.

$$[r, r_1, r_2...] \succ [\text{r,r'}_1, r'_2...] \iff [r_1, r_2...] \succ [r'_1, r'_2...]$$

Teorema. *Dadas preferencias estacionarias, únicamente existen dos maneras de definir la utilidad $U([s_i])$ de una secuencia de acciones:*

- *Aditivamente:* $U([s_0, s_1, s_2...]) = R(s_0) + R(s_1) + R(s_2) + ...$
- *Aditiva con descuento:* $U([s_0, s_1, s_2...]) = R(s_0) + \gamma R(s_1) + \gamma_2 R(s_2) + ...$

El teorema anterior ha sido anunciado para secuencias infinitas, pero nada nos garantiza que la adición de la utilidad converga ni que lo haga en un tiempo finito. ¿Cómo solucionamos este inconveniente?

La primera solución ya ha sido mentada, si determinamos un horizonte finito en T y desarrollamos una política no-estacionaria que depenga del tiempo restante.

La segunda opción es crear un **estado absorbente**. Ese estado es uno en que con probabilidad 1, si el agente llega a él morirá. Haciendo este cambio estamos forzando la secuencia infinita a tener fin, convirtiendo la utilitdad de todo estado en finita.

Y por tercera y última opción, mediante el segundo punto del teorema: adición con descuento. Dada $0 < \gamma < 1$, podemos expresar la utilidad como

$$U([s_0, ..., s_\infty]) = \sum_{t=1}^{\infty} \gamma^t R(s_t) \leq R_{max}/(1-\gamma)$$

donde si γ tiene valores cercanos a 0 indica que estamos mucho más interesados en las recompensas de los estados más inmediatos, mientras que con γ más cercana a 1 con las recompensas de los estados que vendran más tarde.

Matemáticamente se puede demostrar que usando el descuento los algoritmos si convergen, y por lo tanto obtengamos una solución.

5.3 Políticas y Utilidades Óptimas

Siguiendo con la elección de que la utilidad de un estado es la suma de las recompensas con descuento de la secuencia, podemos comparar políticas por el valor esperado. Dada una política π, la utilidad con descuento esperada es $U_\pi = \sum_{t=1}^{\infty} \gamma^t R(s_t)$. Como los estados

5.4 Iteración de Valores

6 Aprendizaje por Refuerzo

7 Referencias