Benemérita Universidad
Autónoma de Puebla

Facultad de Ciencias Físico Matemáticas
Posgrado en Ciencias Matemáticas

Aprendizaje por Refuerzo vía Procesos de Decisión de Markov: Caso Descontado

Trabajo recepcional en la modalidad de

T E S I S

que como requisito para obtener el grado de

Maestro en Ciencias (Matemáticas)

P R E S E N T A

Josué Juárez Morales

Asesor:

Dr. Hugo Adán Cruz Suárez

Puebla, Puebla. Julio 2024

DR. SEVERINO MUÑOZ AGUIRRE
SECRETARIO DE INVESTIGACIÓN Y
ESTUDIOS DE POSGRADO, FCFM-BUAP
P R E S E N T E:

Por este medio le informo que el C:

JOSUÉ JUÁREZ MORALES

estudiante de la Maestría en Ciencias (Matemáticas), ha cumplido con las indicaciones que el Jurado le señaló en el Coloquio que se realizó el día 21 de junio de 2024, con la tesis titulada:

Aprendizaje por Refuerzo vía Procesos de Decisión de Markov: Caso Descontado

Por lo que se le autoriza a proceder con los trámites y realizar el examen de grado en la fecha que se le asigne.

A T E N T A M E N T E.
H. Puebla de Z. a 25 de junio de 2024

DR. RAÚL ESCOBEDO CONDE
COORDINADOR DEL POSGRADO
EN MATEMÁTICAS.

Facultad
de Ciencias
Físico Matemáticas

Av. San Claudio y 18 sur, edif. FM1
Ciudad Universitaria, Col. San
Manuel, Puebla, Pue. C.P. 72570
(222) 229 55 00 Ext. 7550 y 7552

A mis padres.

Agradecimientos

- Comienzo agradeciendo a mis padres, por su amor y apoyo que he recibido de ellos todos estos años de formación. Ellos son la motivación para seguir adelante y convertirme en la persona que quiero ser.

- A mi amada familia y amigos cercanos, de quienes obtengo cálida compañía, valiosos consejos de vida, y toda la ayuda y apoyo incondicional que yo pueda desear.

- Mención especial a mis compañeros (ahora grandes amigos) de generación y amistades que hice en esta nueva ciudad. Ellos me enseñaron mucho, e hicieron de lado cualquier sentimiento de soledad que haya podido tener en estos dos años. Sin duda son parte responsable del logro que he alcanzado.

- No puedo dejar de agradecer a mi asesor Dr. Hugo Adán. Por haberme aceptado como tesista; por su conocimiento transmitido; por su ayuda, apoyo y disponibilidad siempre presente; por su infinita paciencia de la que fui afortunado estos dos años de estudio.

- Agradezco a los miembros de mi comisión revisora. Sus comentarios y correcciones realizados agregan valor a este trabajo de tesis.

- A la FCFM de la Benemérita Universidad Autónoma de Puebla por haberme recibido. A los profesores de mis cursos, por sus valiosas cátedras. Espero poder poner en orgullo el nombre de la facultad.

- Se agradece al CONACYT (ahora CONAHCYT) por el apoyo económico recibido para mis estudios de maestría.

Resumen

El objetivo de este trabajo es dar una descripción matemática del Aprendizaje por Refuerzo; un paradigma inspirado en la interacción con el ambiente como medio de aprendizaje. Se presentada la teoría de Procesos de Decisión de Markov y el problema de control óptimo, la base que transcribe el paradigma de aprendizaje al lenguaje matemático. Las técnicas de Programación Dinámica y la Aproximación Estocástica son representadas mediante los algoritmos de Iteración de Valores y Q-learning respectivamente. Ambos algoritmos son implementados para la solución en distintos sistemas tomados como ejemplos. Finalmente se comentan posibles trabajos futuros que den continuación a este.

Palabras Clave— Aprendizaje por Refuerzo; Proceso de Decisión de Markov; Programación Dinámica.

Índice general

Introducción

El *Aprendizaje por Refuerzo* (AR), es un paradigma basado en la idea de que la obtención de conocimiento es motivada por la recepción de recompensas debido al hacer del aprendiz [35]. Tal paradigma es inspirado en la forma que los seres vivos aprenden: con base en a la exploración e interacción con su ambiente[1] [30]. Por ejemplo, un niño, aprende a caminar probando un conjunto de movimientos con su cuerpo; a lo mejor una serie de pasos logran desplazarlo a su lugar deseado, desencadenando emociones de felicidad por obtener su logro; tal vez decide explorar los que resultan unos movimientos equivocados y hacen que se tropiece, recibiendo señales de dolor en el proceso. El niño aprende de sus acciones y de la respuesta recibida por estas, y para su próxima caminata, conoce la sucesión de movimientos correctos para obtener lo deseado.

Los *Procesos de Decisión de Markov* (PDMs) son una herramienta en la teoría de decisión [26, 35]. En PDMs, un *ambiente* o *sistema* es modelado como un conjunto de estados y acciones (en literatura del AR, se dice que tales acciones son realizadas por un *agente* controlador del sistema) que pueden ser aplicadas al sistema para modificar su estado presente. Una componente importante del PDM es una función recompensa, cuya responsabilidad es gratificar o penalizar las decisiones tomadas. De acuerdo al ejemplo del niño que aprende a caminar, él es el agente controlador, y debe realizar a su cuerpo (el sistema) los movimientos (acciones) que logren su objetivo de aprender a caminar. El proceso de aprendizaje será en base a sus movimientos realizados y la retroalimentación que obtiene de su entorno.

En conjunto, los elementos de un PDM permiten dar una descripción con elementos matemáticos a las ideas que sigue él AR [30]. El objetivo deseado por el agente es

[1]No es el único tipo de aprendizaje presente en los seres vivos. En el contexto de la psicología, las teorías del aprendizaje difieren respecto al proceso involucrado [12].

codificado en los elementos del PDM. La meta es controlar el sistema de forma que se logre lo propuesto por el agente. De acuerdo al paradigma de AR, tal meta es lograda realizando las acciones que proporcionen la mayor cantidad de recompensa.

En el contexto de computación, se entiende como aprendizaje por refuerzo a algoritmos dentro del área de aprendizaje máquina [35]. El objetivo de estos es obtener un PDM que se comporte de la forma esperada, según un criterio de optimización que depende de la suma de las recompensas obtenidas por el control aplicado al sistema. Particularmente, este trabajo se enfoca en un criterio llamado ≪caso descontado≫ [26, 34].

El aumento en el poder de cómputo ha disparado el uso de AR a un sin fin de aplicaciones como la robótica [38], videojuegos [27, 35], y otras en la industria [5]. Recientemente, con el auge de las redes neuronales profundas, han aparecido las primeras soluciones de AR basadas en aprendizaje profundo (deep learning) [24].

Dado el rápido desarrollo del AR en los últimos años, las referencias de esta área son principalmente enfocadas a la parte algorítmica, dejando de lado los bloques básicos que justifican el funcionamiento de estos. El objetivo de este trabajo es presentar un desarrollo matemático del AR, complementando con implementaciones a sistemas que sirvan como ejemplo para relacionar los conceptos expuestos.

El primer capítulo de este trabajo aborda la teoría de los PDM, cuya relevancia es importante para el marco abstracto del AR; a la vez se plantea el objetivo que busca resolver el AR en términos del control óptimo: una optimización de las ≪funciones de valor≫ cuya dependencia es la recompensa obtenida con base en las acciones realizadas. El segundo capítulo es respecto a la *Programación Dinámica* (PD), una técnica basada en la recursividad de las funciones de valor y que es capaz de resolver el problema de control óptimo; al final del capítulo se exponen ejemplos de sistemas capaces de ser descritos como un PDM y resueltos mediante PD. El tercer capítulo es respecto a la teoría de las ≪Q-funciones≫ y su relación con PD; también, se plantea otra técnica de AR basada en simulaciones del sistema a resolver; después, se presentan ejemplos de sistemas adecuados para ser resueltos mediante dicha técnica basada en simulaciones. El último capítulo es para discutir cuestiones complementarias a este trabajo, problemas que la teoría presentada no es capaz de resolver, y posibles líneas de investigación; todas

como posible trabajo futuro que de continuación a éste. Finalmente se da un apéndice, cuyo tema son dos resultados utilizados en el contenido principal del trabajo.

Todas las implementaciones mencionadas en las secciones de ejemplos, fueron realizadas en el lenguaje de programación Python 3, junto con las librerías de cómputo científico comunes. Los códigos están disponibles para consulta dentro del repositorio de GitHub: `https://github.com/JosueJuarez/PD-AR`.

Capítulo 1

Procesos de Decisión de Markov

En este capítulo se presenta la teoría referente a Procesos de Decisión de Markov. Se trata de un modelo matemático, el cual describe un proceso estocástico cuya evolución es debida a un control realizado al sistema mediante acciones en periodos de tiempo secuenciales[1].

Es por esta razón que los PDMs constituyen la base matemática sobre la cual los algoritmos de aprendizaje por refuerzo trabajan. Pues el primer paso para resolver un problema de control, es describir el sistema como un PDM; posteriormente sigue la elección de un algoritmo de AR adecuado.

Las definiciones y el desarrollo presentado en este capítulo siguen las ideas de Puterman [26, capítulos 2 y 5]. Otras exposiciones pueden consultarse en [4, 34]. Las referencias [6, 30, 35] contienen capítulos más breves respecto al tema.

1.1. Modelo de Decisión de Markov

El siguiente objeto matemático es el puente para dar descripción con elementos abstractos al sistema que se desea controlar. Dependiendo el objetivo a lograr, el primer paso a realizar es una correcta elección de los siguientes elementos presentados a fin de tener una correcta descripción del sistema.

[1]Tales modelos también se encuentran en la literatura como *multi-stage decision process* [26].

Definición 1.1.1. El *Modelo de Decisión de Markov estacionario* (MDM) se define como la colección:

$$(X, A, \{A(x) \,|\, x \in X\}, p, r),$$

donde:

- $(X, \mathcal{F})$ es un espacio medible, donde X es un conjunto no vacío denominado *espacio de estados* y $\mathcal{F}$ su correspondiente σ-álgebra.
- $(A, \mathcal{G})$ es un espacio medible, donde A es un conjunto llamado *espacio de acciones* y $\mathcal{G}$ su correspondiente σ-álgebra.
- $\{A(x) \,|\, x \in X\}$ es una familia formada por los conjuntos $A(x)$ de *acciones admisibles* para cada estado $x \in X$; $A(x) \subseteq A$.
- Se define $\mathbb{K} := \{(x, a) \,|\, x \in X, a \in A(x)\}$ el conjunto de parejas compuestas por estados-acciones admisibles.
- p es un kernel estocástico en X dado $\mathbb{K}$, es decir:
  a) $p(\cdot \,|\, x, a)$ es una medida de probabilidad en X, para todo $(x, a) \in \mathbb{K}$.
  b) $p(B \,|\, \cdot)$ es una función medible en $\mathbb{K}$, para todo $B \in \mathcal{F}$.
- $r : \mathbb{K} \to \mathbb{R}$ es una función conocida llamada *función recompensa*.

Dependiendo del sistema a modelar, el conjunto de estados y acciones puede ser continuo o numerable (finito o infinito). En general, **a lo largo de este trabajo se consideran finitos o numerables**, aunque bien, al inicio de cada capítulo se especifica como serán considerados.

La dinámica del MDM se describe de la siguiente manera:

1. Dado un estado inicial $x_0 \in X$, se aplica una acción $a_0 \in A(x_0)$.
2. A continuación, el sistema realiza una transición al estado $x_1 \in X$ de acuerdo con la probabilidad de transición p y se genera una recompensa $r_0 = r(x_0, a_0)$.
3. Cada decisión es realizada en un punto del tiempo llamado *época*. El conjunto de épocas se denota por $\{0, 1, 2, \ldots, T\}$, donde T es un entero positivo o infinito, el

cual se denomina *horizonte*. El proceso se repite de esa misma manera en cada época.

Dada alguna época t, la *historia del proceso* hasta la época t se guarda en el vector $h_t = (x_0, a_0, x_1, a_1, \ldots, x_{t-1}, a_{t-1}, x_t)$. Definiendo $\mathbb{H}_t := \mathbb{K}^t \times X = \mathbb{K} \times \mathbb{H}_{t-1}$, $t = 1, 2, 3, \ldots$; $\mathbb{H}_0 := X$; como el *conjunto de historias admisibles* hasta la época t, se tiene que $h_t \in \mathbb{H}_t$.

Observación 1.1.1.

a) Puterman [26, página 20] presenta una definición más general del MDM, donde sus elementos son dependientes de la época de decisión a diferencia de la definición dada aquí. Por esa razón se le ha dado el adjetivo *estacionario*, pues cada uno de sus componentes no cambian conforme transcurren las épocas.

b) En los problemas de horizonte finito, dado que no se realiza acción en la última época, es necesario añadir como elemento del MDM a una función recompensa terminal $r_T : X \to \mathbb{R}$.

1.2. Reglas y Políticas de Decisión

En la dinámica del MDM descrita, es necesario realizar una acción para efectuar una transición entre dos estados. ¿Cómo escoger tal acción? Pues bien, la elección se realiza siguiendo una regla; estas se clasifican dependiendo la información que requieran y la forma en que asignan la acción.

Las decisiones pueden realizarse de manera aleatoria, es decir, se escogería una acción con probabilidad dada por una función de distribución. Denotamos $\mathcal{P}(A)$ como el conjunto de distribuciones de probabilidad sobre los subconjuntos de A. De esta manera, escoger una acción aleatoriamente, significa tomar una distribución $\varphi(\cdot) \in \mathcal{P}(A)$ y realizar la acción $a \in A$ con probabilidad $\varphi(a)$.

Definición 1.2.1. Se define una *regla de decisión estocástica* en una época t, como el mapeo:

$$d_t : \mathbb{H}_t \to \mathcal{P}(A); \quad t = 0, 1, \ldots, T.$$

El conjunto de todas las reglas de decisión estocásticas en una época t, se denota como D_t^{HR}.

Se tiene el esquema siguiente: en una época t siguiendo la regla de decisión estocástica d_t, se escoge la acción a_t con una probabilidad $\pi_t(\{a_t\} \mid h_t)$.

Cuando la decisión no depende de toda la historia del proceso, sino, únicamente en el estado anterior, entonces el mapeo:

$$d : X \to \mathcal{P}(A);$$

recibe el nombre de *regla de decisión markoviana estocástica*. El conjunto de todas las reglas de decisión markovianas estocásticas se denota como D^{MR}.

Existen los casos en que la distribución de probabilidad está concentrada en alguna acción particular ($\pi_t(\{a\} \mid h_t) = 1$, para algún $a \in A$); cuando esto sucede, la elección de la acción a dado la historia h_t es completamente determinista. Lo anterior motiva para definir una *regla de decisión determinista* en una época t, como el mapeo:

$$d_t : \mathbb{H}_t \to A; \text{ tal que } d_t(h_t) \in A(x_t); \quad t = 0, 1, \ldots, T.$$

Cuando la decisión determinista no depende de toda la historia, entonces es una *regla de decisión markoviana determinista*:

$$d : X \to A; \text{ tal que } d(x) \in A(x).$$

Se denota al conjunto de todas las reglas de decisión deterministas a la época t como D_t^{HD}, a las markovianas deterministas como D^{MD}.

Observación 1.2.1. Para un MDM no estacionario, las reglas de decisión de clase MR y MD son también dependientes de las épocas de decisión; es decir, D_t^{MR} representaría al conjunto de reglas de decisión markovianas en la época t. Sin embargo, para el caso estacionario, estos conjuntos coinciden en todas las épocas de decisión. Cuando sea necesario se dejará el subíndice t solamente para enfatizar la época en que sucede la decisión.

Definición 1.2.2. Una *política de decisión* de clase K es una sucesión de reglas de

decisión:

$$\pi \coloneqq (d_0, d_1, d_2, \ldots, d_{T-1}), \quad d_t \in D_t^K;$$

siendo K una de las clases HR, HD, MR, MD.

Como se verá más adelante en la sección 1.5, las políticas de clase MR y MD cumplen un papel importante en la teoría de los PDMs.

Una política de decisión se le dice *estacionaria* cuando $d_t = d$ para cualquier época t; tal política tendría entonces la forma $\pi_d = (d, d, \ldots, d)$. El conjunto de todas las políticas de decisión de clase K se denota por $\Pi^K \coloneqq D_1^K \times \cdots \times D_{T-1}^K$; mientras que todas las políticas de decisión estacionarias deterministas se denotan por Π^{SD} y a las estacionarias estocásticas como Π^{SR}.

Por las definiciones anteriores, reconocemos las siguientes jerarquías en las políticas:

$$\Pi^{SD} \subset \Pi^{SR} \subset \Pi^{MR} \subset \Pi^{HR};$$
$$\Pi^{SD} \subset \Pi^{MD} \subset \Pi^{MR} \subset \Pi^{HR};$$
$$\Pi^{SD} \subset \Pi^{MD} \subset \Pi^{HD} \subset \Pi^{HR}.$$

Una política contiene el plan a seguir por un agente a fin de controlar el MDM. Son elementos esenciales para el agente, pues son suficientes para determinar el comportamiento del sistema.

1.3. Proceso de Decisión de Markov

Dado un MDM estacionario $(X, A, \{A(x) \,|\, x \in X\}, p, r)$ y un conjunto de épocas $\{0, 1, 2, \ldots, T\}$; denotamos el espacio producto $\Omega \coloneqq (X \times A)^\infty$ y la σ-álgebra producto $\mathcal{H} \coloneqq (\mathcal{F} \times \mathcal{G})^\infty$. Un elemento $\omega \in \Omega$ es de la forma $(x_0, a_0, x_1, a_1, x_2, a_2, \ldots)$ y recibe el nombre de *realización del proceso*.

Definimos las siguientes variables aleatorias:

$$X_t : \Omega \to X, \qquad Y_t : \Omega \to A, \qquad Z_t : \Omega \to \mathbb{H}_t,$$
$$\omega \mapsto x_t; \qquad \omega \mapsto a_t; \qquad \omega \mapsto h_t;$$

es decir, X_t (Y_t) es el estado ocupado por el (la acción realizada al) sistema en la época t dada la realización ω del proceso. Mientras que Z_t es la historia del proceso hasta la época t dada la realización.

Observación 1.3.1. Respecto a las probabilidades de transición, en lo subsecuente denotaremos a $p(X_{t+1} = j \,|\, X_t = i, Y_t = a)$ como $p(j \,|\, i, a)$.

Sea $\pi \in \Pi^{HR}$ una política de decisión estocástica y $x \in X$ un estado inicial. El teorema de Ionescu-Tulcea[2] asegura la existencia y unicidad de una medida de probabilidad $\mathbb{P}_x^\pi$ sobre el espacio de medible $(\Omega, \mathcal{H})$, tal que para todo $\omega \in \Omega$, cumple lo siguiente:

$$\mathbb{P}_x^\pi(X_0 = x) = 1; \tag{1.3.1}$$

$$\mathbb{P}_x^\pi(Y_t = a \,|\, Z_t = h_t) = \pi_t(a \,|\, h_t); \tag{1.3.2}$$

$$\mathbb{P}_x^\pi(X_{t+1} = x \,|\, Z_t = h_t, Y_t = a_t) = p(x \,|\, x_t, a_t). \tag{1.3.3}$$

Definición 1.3.1. El proceso estocástico $(\Omega, \mathcal{H}, P_x^\pi, \{X_t\})$, recibe el nombre de *Proceso de Decisión de Markov*.

Observación 1.3.2. Cuando el estado inicial x es escogido de acuerdo a una distribución de probabilidad inicial $\varphi_0 \in \mathcal{P}(X)$[3] entonces la ecuación (1.3.1) se escribe como:

$$\mathbb{P}_x^\pi(X_0(\omega) = x) = \varphi_0(x).$$

Si $\pi \in \Pi^{HR}$ es una política estocástica, como puede observarse en la ecuación (1.3.3), la variable aleatoria X_{t+1} depende de la historia del proceso h_t únicamente para la elección de la acción $a_t \sim \pi_t(\cdot \,|\, h_t)$, es decir:

$$X_{t+1} = F'(h_t, a_t);$$

siendo $F' : \mathbb{H}_t \times A \to X$. Por otro lado, si $\pi \in \Pi^{MR}$ es markoviana, entonces la dependencia es únicamente sobre x_t, pues $a_t \sim \pi_t(\cdot \,|\, x_t)$:

$$X_{t+1} = F(x_t, a_t);$$

[2]Para más detalle respecto a la medida producto, ver la referencia [2, capítulo 2].

[3]$\mathcal{P}(X)$ representa el conjunto de distribuciones de probabilidad sobre los subconjuntos de X.

$F : \mathbb{K} \to X$; y por lo tanto $\{X_t\}$ es una cadena de Markov.

1.4. Control Óptimo

Los PDMs proveen la estructura matemática para describir un sistema y su dinámica. El siguiente paso es controlar tal sistema, de manera que éste siga algún criterio predeterminado a largo plazo, o bien que llegue a un estado final deseado.

Tales criterios y objetivos se encuentran implícitos en la función recompensa que se escoja para el MDM [35]. Ésta nos dice qué acción es mejor tomar en algún estado dado, por lo tanto, son la base para modificar una política: si siguiendo una política, en alguna época obtenemos recompensa baja debido a la acción elegida, entonces para la próxima vez se escoge una política que provea mayor recompensa. Sin embargo, no se debe perder de vista, que el enfoque de aprendizaje es obtener la mayor recompensa a largo plazo y no al revés.

1.4.1. Función de Valor Descontado

Con el fin de relacionar el enfoque de aprendizaje y un PDM se dan las siguientes definiciones.

Definición 1.4.1. Dado $x \in X$ y $\pi \in \Pi^K$ arbitrarios. Sea $(\Omega, \mathcal{H}, P_x^\pi, \{X_t\})$ el respectivo PDM con estado inicial $X_0 = x$ y sea $\gamma \in [0, 1)$. Las funciones $v, v_T : X \times \Pi^K \to \mathbb{R}$ definidas como:

$$v_T(x, \pi) \coloneqq \mathbb{E}_x^\pi \left(\sum_{t=0}^{T-1} \gamma^t r(X_t, Y_t) + \gamma^T r_T(X_T) \right), \quad T < \infty; \tag{1.4.1}$$

$$v(x, \pi) \coloneqq \mathbb{E}_x^\pi \left(\sum_{t=0}^{\infty} \gamma^t r(X_t, Y_t) \right), \quad T = \infty; \tag{1.4.2}$$

reciben el nombre de *función de valor descontado*.

Observación 1.4.1. La esperanza $\mathbb{E}_x^\pi$ es respecto a la medida de probabilidad P_x^π.

El propósito de los términos γ^t es penalizar ≪*descontar*≫ el valor de la recompensa para acciones realizadas en épocas muy tardías. Por tal, la constante γ recibe el nombre

de *factor de descuento*. El caso $\gamma = 0$ en la literatura se le refiere como *miope*, en el sentido de que solamente son de interés las recompensas inmediatas [35, página 14].

La esperanza de la suma de las recompensas descontadas es conocida en la literatura como *criterio descontado*; siendo el más utilizado debido a las diversas interpretaciones (tasa de interés, probabilidad de supervivencia, etc.) y a las conveniencias matemáticas que facilita [35].

Otros criterios encontrados en la literatura:

- *Recompensa total acumulada* ($T < \infty$):
$$\mathbb{E}_x^\pi \left(\sum_{t=0}^{T-1} r(X_t, Y_t) + r_T(X_T) \right).$$
- *Recompensa promedio*:
$$\lim_{T\to\infty} \mathbb{E}_x^\pi \left(\frac{1}{T} \sum_{t=0}^{T} r(X_t, Y_t) \right).$$

Estos criterios junto al caso descontado son los principales y más utilizados, pues son capaces de resolver la mayoría de problemas encontrados en la literatura [35]. Por otro lado, como se aprecia en el título de este trabajo, **solamente se estudia el criterio descontado**. Teoría para estos otros criterios puede encontrarse en las referencias [22, 26, 34].

Si bien se presentó la función de valor descontado para el caso de horizonte finito, **en este trabajo se van a considerar solamente MDM de horizonte infinito**. Puterman [26, capítulo 4], también desarrolla teoría y algoritmos de programación dinámica para MDM de horizonte finito.

1.4.2. Problema de Control Óptimo

Dado que las funciones de valor toman en cuenta las decisiones seguidas por la política y las recompensas obtenidas. La función de éstas es cuantificar el buen desempeño a largo plazo de una política para todos los estados del sistema. ¿Qué política nos proporciona la mayor cantidad de recompensa descontada?

Definición 1.4.2. Una política de decisión $\pi^* \in \Pi^{HR}$ que cumple:

$$v(x, \pi) \leq v(x, \pi^*), \quad \forall x \in X \text{ y } \forall \pi \in \Pi^{HR};$$

recibe el nombre de *política de decisión óptima.*

Definición 1.4.3. Sea $\varepsilon > 0$. Una política de decisión $\pi^* \in \Pi^{HR}$ que cumple:

$$v(x,\pi) \leq v(x,\pi^*) + \varepsilon, \quad \forall x \in X \text{ y } \forall \pi \in \Pi^{HR};$$

recibe el nombre de *política de decisión ε-óptima.*

Definición 1.4.4. La función $v^* : X \to \mathbb{R}$ definida como:

$$v^*(x) \coloneqq \sup_{\pi \in \Pi^{HR}} v(x,\pi); \tag{1.4.3}$$

recibe el nombre de *recompensa óptima* o *función de valor óptimo.*

De este modo, una política $\pi^* \in \Pi^{HR}$ es óptima (ε-óptima) cuando satisface:

$$v^*(x) = v(x,\pi^*), \quad \forall x \in X;$$
$$(v^*(x) \leq v(x,\pi^*) + \varepsilon \quad \forall x \in X).$$

1.5. Políticas de Decisión Markovianas

Los siguientes resultados muestran que basta considerar políticas de decisión markovianas para el estudio de los PDMs.

Teorema 1.5.1. *Sea $\pi \in \Pi^{HR}$ arbitraria. Se tiene que para cada $x \in X$, existe una política de decisión $\pi' \in \Pi^{MR}$ que satisface*

$$\mathbb{P}^{\pi'}(X_t = j, Y_t = a \,|\, X_0 = x) = \mathbb{P}^{\pi}(X_t = j, Y_t = a \,|\, X_0 = x); \tag{1.5.1}$$

para $t = 0, 1, 2, \ldots$

Demostración. Sea $x \in X$ fijo. Para cada $s \in X$ y $a \in A(s)$, se define la regla de decisión $d_t{}' \in D_t^{MR}$ de forma que

$$\pi_t(a \,|\, s) = \mathbb{P}^{\pi}(Y_t = a \,|\, X_0 = x, X_t = s); \quad t = 0, 1, 2, \ldots \tag{1.5.2}$$

Sea $\pi' = (d_0', d_1', d_2', \ldots)$, entonces, por (1.5.2) se cumple que

$$\mathbb{P}^{\pi'}(Y_t = a \,|\, X_t = s) = \pi_t(a \,|\, s) = \mathbb{P}^{\pi}(Y_t = a \,|\, X_0 = x, X_t = s).$$

Se va a demostrar la ecuación (1.5.1) mediante inducción. Para $t = 0$ se cumple. Suponemos se cumple para $t = 0, 1, 2, \ldots, n-1$. Calculamos:

$$\mathbb{P}^{\pi}(X_n = s \,|\, X_0 = x) = \sum_{j \in X} \sum_{a \in A(j)} \mathbb{P}^{\pi}(X_{n-1} = j, Y_{n-1} = a \,|\, X_0 = x) p(s \,|\, j, a) \tag{1.5.3}$$

$$= \sum_{j \in X} \sum_{a \in A(j)} \mathbb{P}^{\pi'}(X_{n-1} = j, Y_{n-1} = a \,|\, X_0 = x) p(s \,|\, j, a) \tag{1.5.4}$$

$$= \mathbb{P}^{\pi'}(X_n = s \,|\, X_0 = x). \tag{1.5.5}$$

Entonces

$$\begin{aligned}
\mathbb{P}^{\pi'}(X_n = s, Y_n = a \,|\, X_0 = x) &= \mathbb{P}^{\pi'}(Y_n = a \,|\, X_0 = x, X_n = s) \mathbb{P}^{\pi'}(X_n = s \,|\, X_0 = x) \\
&= \mathbb{P}^{\pi}(Y_n = a \,|\, X_0 = x, X_n = s) \mathbb{P}^{\pi}(X_n = s \,|\, X_0 = x) \\
&= \mathbb{P}^{\pi}(X_n = s, Y_n = a \,|\, X_0 = x).
\end{aligned}$$

En conclusión el resultado es válido. □

Teorema 1.5.2. *Supongamos $\pi \in \Pi^{HR}$. Entonces para cada $x \in X$ existe $\pi_x' \in \Pi^{MR}$ para la cual*

$$v(x, \pi) = v(x, \pi_x').$$

Demostración. Construimos π_x' como en el Teorema 1.5.1 y observamos que

$$\begin{aligned}
v(x, \pi) &= \mathbb{E}^{\pi}\left(\sum_{t=0}^{\infty} \gamma^t r(X_t, Y_t) \,|\, X_0 = x \right) \\
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}^{\pi}(r(X_t, Y_t) \,|\, X_0 = x) \\
&= \sum_{t=0}^{\infty} \sum_{s \in X} \sum_{a \in A(s)} \gamma^t r(s, a) \mathbb{P}^{\pi}(X_t = s, Y_t = a \,|\, X_0 = x) \\
&= \sum_{t=0}^{\infty} \sum_{s \in X} \sum_{a \in A(s)} \gamma^t r(s, a) \mathbb{P}^{\pi_x'}(X_t = s, Y_t = a \,|\, X_0 = x)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}^{\pi'_x}(r(X_t, Y_t) \,|\, X_0 = x) \\
&= \mathbb{E}^{\pi'_x}\left(\sum_{t=0}^{\infty} \gamma^t r(X_t, Y_t) \,|\, X_0 = x\right) \\
&= v(x, \pi'_x);
\end{aligned}
$$

por lo tanto, el Teorema 1.5.2 es válido. □

1.6. Representación Vectorial de un PDM

Sea $(\mathcal{V}, \|\cdot\|)$ el espacio vectorial normado de funciones reales acotadas sobre X con la norma del supremo[4]. Cuando X es discreto, los elementos de $\mathcal{V}$ se identifican como vectores y los operadores lineales sobre $\mathcal{V}$ como matrices[5].

Observación 1.6.1. Lo anterior es un abuso de notación. Estrictamente, un elemento u de $\mathcal{V}$ es una función $u : X \to \mathbb{R}$. Por otro lado se tiene al conjunto de sucesiones $\{(u(x_1), u(x_2), \ldots) : u \in \mathcal{V};\ x_i \in X, i = 1, 2, \ldots\}$. Sin embargo, con el propósito de simplificar, no se hace distinción entre estos espacios. En particular, para un conjunto X finito, la notación se reduce a la de vectores.

Sea $d \in D^{MD}$ y $x \in X$, definimos los siguientes elementos:

$$r_d(x) \coloneqq r(x, d(x)); \qquad\qquad p_d(j \,|\, x) \coloneqq p(j \,|\, x, d(x));$$

y para $d \in D^{MR}$:

$$r_d(x) \coloneqq \sum_{a \in A(x)} \pi(a \,|\, x) r(x, a); \qquad p_d(j \,|\, x) \coloneqq \sum_{a \in A(x)} \pi(a \,|\, x) p(j \,|\, x, a).$$

Si X es finito con un número de estados $|X|$, denotamos por r_d al vector de tamaño $|X|$ cuyos entradas son $r_d(x)$; así mismo a P_d a la matriz $|X| \times |X|$ cuya (j, s)-entrada

[4] $\|u\| \coloneqq \sup_{x \in X} |u(x)|, \quad u \in \mathcal{V}$.

[5] Si H es una matriz:

$$\|H\| \coloneqq sup_{i \in X} \sum_{j \in X} |H(j \,|\, i)|;$$

donde $H(j \,|\, i)$ es el (i, j)-ésimo elemento de H.

es $p_d(s \mid j)$. r_d y P_d reciben el nombre de *vector recompensa* y *matriz de transición* correspondiente a la decisión d.

Lema 1.6.1. *Supongamos que X es discreto, $|r(x,a)| \leq M \in \mathbb{R}^+$ para todo $(x,a) \in \mathbb{K}$, y $\gamma \in [0,1]$. Entonces para todo $u \in \mathcal{V}$ y $d \in D^{MR}$ se cumple:*

$$r_d + \gamma P_d u \in \mathcal{V}.$$

Demostración. Por suposición $\|r_d\| \leq M \ \forall d \in D^{MR}$ i.e. $r_d \in \mathcal{V}$. Por otro lado, P_d es una matriz de probabilidad y entonces $\|P_d\| = 1$, luego $\|P_d u\| \leq \|P_d\|\|u\| = \|u\|$, por lo tanto $P_d u \in \mathcal{V}$. Concluimos que $r_d + \gamma P_d u \in \mathcal{V}$. □

Observación 1.6.2. El término $r_d + \gamma P_d u$ lo interpretamos como la recompensa esperada total descontada en un paso usando la regla de decisión d y recompensa terminal u.

Sea $x \in X$ y $\pi = (d_0, d_1, d_2, \ldots, d_{T-1}) \in D^{MR}$. Calculamos las siguientes probabilidades de transición:

$$\begin{aligned}
\mathbb{P}^\pi(X_1 = j \mid X_0 = x) &= \sum_{a \in A(x)} \pi_0(a \mid x) p(j \mid x, a) = p_{d_0}(j \mid x); \\
\mathbb{P}^\pi(X_2 = j \mid X_0 = x) &= \sum_{s \in X} \sum_{a \in A(s)} \pi_1(a \mid s) p(j \mid s, a) \mathbb{P}^\pi(X_1 = s \mid X_0 = x) \qquad (1.6.1) \\
&= \sum_{s \in X} p_{d_1}(j \mid s) p_{d_0}(s \mid x) = [P_{d_1} P_{d_0}](j \mid x); \\
\mathbb{P}^\pi(X_3 = j \mid X_0 = x) &= \sum_{s \in X} \sum_{a \in A(s)} \pi_2(a \mid s) p(j \mid s, a) \mathbb{P}^\pi(X_2 = s \mid X_0 = x) \qquad (1.6.2) \\
&= \sum_{s \in X} p_{d_2}(j \mid s) [P_{d_1} P_{d_0}](s \mid x) = [P_{d_2} P_{d_1} P_{d_0}](j \mid x); \\
&\vdots \\
\mathbb{P}^\pi(X_t = j \mid X_0 = x) &= [P_{d_{t-1}} \cdots P_{d_1} P_{d_0}](j \mid x); \qquad (1.6.3)
\end{aligned}$$

denotaremos a $P_\pi^t(j \mid x) \coloneqq [P_{d_{t-1}} \cdots P_{d_1} P_{d_0}](j \mid x)$.

La ecuación (1.6.3) nos dice que bajo la política markoviana π, la cadena de Markov $\{X_t\}$ tiene como probabilidad de transición en t pasos del estado $X_0 = x$ al estado $X_t = j$ el elemento (x,j)-ésimo del producto de las t primeras matrices de transición, definida como la matriz P_π^t.

Con lo anterior se puede tener una representación vectorial de la esperanza respecto a la medida producto. Dado $X_0 = x$ y $u \in \mathcal{V}$:

$$\mathbb{E}_x^\pi(u(X_t)) = \sum_{j \in X} \mathbb{P}^\pi(X_t = j \,|\, X_0 = x)u(j) = P_\pi^t(j \,|\, x)u(j);$$

de esta forma:

$$\mathbb{E}^\pi(u(X_t))(x) = [P_\pi^t u](x).$$

Capítulo 2

Programación Dinámica

La Programación Dinámica fue desarrollada por Bellman [3] como una técnica para resolver problemas de optimización. La idea principal de PD es que un problema de optimización puede descomponerse en sub-problemas; el conjunto de soluciones óptimas a estos sub-problemas constituye la solución óptima al problema original.

En general, el término PD es muy amplio y su significado depende del área de aplicación. En el caso de AR, se entiende por PD como el conjunto de técnicas y algoritmos para resolver un PDM. En específico para un MDM donde se conoce su modelo por completo [30, 35]. La descomposición en sub-problemas aparece debido a la recursividad que hay en las funciones de valor.

El desarrollo de este capítulo sigue al de Puterman [26, capítulo 6]. Otras referencias con desarrollos completos de PD son [4, 11, 34]. Las referencias [30, 35] contienen un tratado más breve sobre PD.

Como se vio en la sección 1.5, dado $x \in X$ y $\pi \in \Pi^{HR}$ existe una política $\pi' \in \Pi^{MR}$ con el mismo valor de la función de valor. Entonces

$$v^*(x) = \sup_{\pi \in \Pi^{HR}} v(x, \pi) = \sup_{\pi \in \Pi^{MR}} v(x, \pi);$$

por lo tanto, se puede restringir el estudio de los algoritmos de programación dinámica a las políticas markovianas.

Recordamos que en este trabajo se consideran MDM de horizonte infinito. Además, en este capítulo se tienen en cuenta las siguientes suposiciones al MDM:

Suposición 2.0.1.

a) Conjunto de estados X finito o numerable.

b) Función de recompensa acotada: existe $M \in \mathbb{R}$ tal que $|r(x,a)| \leq M$, para todo $(x,a) \in \mathbb{K}$.

c) Factor de descuento $\gamma \in [0,1)$.

2.1. Recursividad en las Funciones de Valor

En el siguiente resultado aparece la noción de PD, pues se tiene que para realizar el cálculo de una función de valor, se puede utilizar el resultado de otra función de valor anterior.

Proposición 2.1.1. *Sea* $\pi = (d_0, d_1, d_2, d_3, \ldots) \in \Pi^{MR}$. *Entonces*

$$v(x,\pi) = \mathbb{E}_x^{\pi}(r(X_0, Y_0) + \gamma v(X_1, \pi')), \quad x \in X; \tag{2.1.1}$$

siendo $\pi' = (d_1, d_2, d_3, \ldots)$.

Demostración.

$$\begin{aligned}
v(x,\pi) &= \mathbb{E}^{\pi}\left(\sum_{t=0}^{\infty} \gamma^t r(X_t, Y_t) \,|\, X_0 = x\right) \\
&= \mathbb{E}^{\pi}(r(X_0, Y_0) \,|\, X_0 = x) + \gamma \mathbb{E}^{\pi}\left(\sum_{t=1}^{\infty} \gamma^{t-1} r(X_t, Y_t) \,|\, X_0 = x\right) \\
&= \mathbb{E}^{\pi}(r(X_0, Y_0) \,|\, X_0 = x) + \gamma \mathbb{E}^{\pi}\left(\mathbb{E}^{\pi'}\left(\sum_{t=1}^{\infty} \gamma^{t-1} r(X_t, Y_t) \,|\, X_1 = x_1\right) | \, X_0 = x\right) \\
&= \mathbb{E}^{\pi}(r(X_0, Y_0) \,|\, X_0 = x) + \gamma \mathbb{E}^{\pi}(v(X_1, \pi') \,|\, X_0 = x) \\
&= \mathbb{E}_x^{\pi}(r(X_0, Y_0) + \gamma v(X_1, \pi')).
\end{aligned}$$

□

De acuerdo a la sección 1.6, la ecuación (2.1.1) se denota en forma vectorial como

$$v^{\pi} = r_{d_0} + \gamma P_{d_0} v^{\pi'};$$

incluso, si $\pi = (d, d, d, \ldots)$, para $d \in D^{MR}$; entonces coincide con π' y lo anterior se simplifica a

$$v^{\pi} = r_d + \gamma P_d v^{\pi}.$$

Definición 2.1.1. Bajo las condiciones del Lema 1.6.1 se define el operador $L_d : \mathcal{V} \to \mathcal{V}$ como:

$$L_d u \coloneqq r_d + \gamma P_d u.$$

Por lo tanto, vemos que v^{π} (π estacionaria) es un punto fijo del operador L_d en $\mathcal{V}$. Los siguientes resultados completan la idea: dada una política estacionaria, su función de valor resuelve el sistema de ecuaciones anterior.

Proposición 2.1.2. *Dado $d \in D^{MR}$. El operador L_d es una contracción*[1] *en $\mathcal{V}$.*

Demostración. Sean $u, w \in \mathcal{V}$. Supongamos que $L_d u \leq L_d w$, entonces para $x \in X$

$$\begin{aligned}
[L_d w](x) - [L_d u](x) &= \gamma[P_d w](x) - \gamma[P_d u](x) \\
&= \gamma[P_d(w - u)](x) \\
&\leq \gamma \|P_d\| \|w - u\| \\
&\leq \gamma \|w - u\|.
\end{aligned}$$

Suponiendo ahora que $L_d u \geq L_d w$, y repitiendo el mismo argumento, se obtiene que

$$|[L_d w](x) - [L_d u](x)| \leq \gamma \|w - u\|;$$

tomando el supremo sobre $x \in X$ implica que:

$$\|L_d w - L_d u\| \leq \gamma \|w - u\|.$$

□

Teorema 2.1.3. *Supongamos que $\gamma \in [0, 1)$. Entonces, para cualquier política de decisión estacionaria $\pi_d = (d, d, d, \ldots)$, $d \in D^{MR}$, v^{π_d} es la única solución en $\mathcal{V}$ de*

$$u = r_d + \gamma P_d u.$$

[1] Ver Definición A.1.1

Además, v^{π_d} puede escribirse como

$$v^{\pi_d} = (I - \gamma P_d)^{-1} r_d.$$

Demostración. Ver referencia [26, capítulo 6, página 145]. □

2.2. Ecuaciones de Bellman

Con el fin de comenzar la búsqueda de acciones que proporcionen mayor recompensa descontada, se da la definición del siguiente operador, el cual tiene gran relevancia en la teoría de programación dinámica.

Definición 2.2.1 (Operador de Bellman)**.** Considerando las Suposiciones 2.0.1, definimos el operador $L : \mathcal{V} \to \mathcal{V}$:

$$Lu(x) \coloneqq \sup_{a \in A(x)} \left\{ r(x,a) + \gamma \sum_{j \in X} p(j \mid x, a) u(j) \right\}, \quad x \in X. \tag{2.2.1}$$

Observación 2.2.1. Cuando el conjunto A es finito, el supremo es obtenido dentro del conjunto considerado, y por lo tanto podemos escribir las ecuaciones de Bellman como el máximo.

A continuación, se relaciona el operador de Bellman con la función de valor óptimo. Más adelante se apreciará la utilidad del resultado para encontrar políticas óptimas.

Teorema 2.2.1. *Dadas las Suposiciones 2.0.1, se cumple que*

$$Lv^*(x) = v^*(x), \quad \forall x \in X;$$

es decir, v^* es solución al sistema de ecuaciones

$$u(x) = \sup_{a \in A(x)} \left\{ r(x,a) + \gamma \sum_{j \in X} p(j \mid x, a) u(j) \right\}, \quad \forall x \in X, u \in \mathcal{V}.$$

Al conjunto de estas ecuaciones se conoce como ecuaciones de programación dinámica o ecuaciones de optimalidad o ecuaciones de Bellman.

Demostración. Sean $x \in X$ y $\pi = (d_0, d_1, d_2, d_3, \ldots) \in \Pi^{MR}$ arbitrarios. Continuando el desarrollo de la ecuación (2.1.1)...

$$\begin{aligned} v(x,\pi) &= \mathbb{E}_x^{\pi}(r(X_0,Y_0) + \gamma v(X_1,\pi')) \\ &= \sum_{a\in A(x)} \pi_0(a\,|\,x)\left[r(x,a) + \gamma \sum_{j\in X} p(j\,|\,x,a)v(j,\pi')\right] \\ &\leq \sum_{a\in A(x)} \pi_0(a\,|\,x)\left[r(x,a) + \gamma \sum_{j\in X} p(j\,|\,x,a)v^*(j)\right] \\ &\leq \sum_{a\in A(x)} \pi_0(a\,|\,x)\left[\sup_{a'\in A(x)}\left\{r(x,a') + \gamma \sum_{j\in X} p(j\,|\,x,a')v^*(j)\right\}\right] \\ &= \sup_{a'\in A(x)}\left\{r(x,a') + \gamma \sum_{j\in X} p(j\,|\,x,a')v^*(j)\right\}; \end{aligned}$$

siendo $\pi' = (d_1, d_2, d_3, \ldots)$. Como x y π son arbitrarios concluimos que

$$v^*(x) \leq \sup_{a'\in A(x)}\left\{r(x,a') + \gamma \sum_{j\in X} p(j\,|\,x,a')v^*(j)\right\}, \quad \forall x \in X. \tag{2.2.2}$$

Dado $x \in X$, sea $a^* \in A(x)$ tal que

$$r(x,a^*) + \gamma \sum_{j\in X} p(j\,|\,x,a^*)v^*(j) = \sup_{a\in A(x)}\left\{r(x,a) + \gamma \sum_{j\in X} p(j\,|\,x,a)v^*(j)\right\};$$

y sea $\varepsilon > 0$ arbitrario. Definimos la política $\pi = (d_0, d_1, d_2, d_3, \ldots) \in \Pi^{MR}$ de forma que cumpla lo siguiente: $d_0 \in D^{MR}$ es tal que $\pi_0(a^*\,|\,x) = 1$ y además

$$v(j,\pi) \geq v^*(j) - \varepsilon, \quad \forall j \in X.$$

Luego

$$\begin{aligned} v(x,\pi) &= r(x,a^*) + \gamma \sum_{j\in X} p(j\,|\,x,a^*)v(j,\pi') \\ &> r(x,a^*) + \gamma \sum_{j\in X} p(j\,|\,x,a^*)(v^*(j) - \varepsilon) \\ &= r(x,a^*) + \gamma \sum_{j\in X} p(j\,|\,x,a^*)v^*(j) - \gamma\varepsilon \end{aligned}$$

$$= \sup_{a\in A(x)} \left\{ r(x,a) + \gamma \sum_{j\in X} p(j \mid x,a) v^*(j) \right\} - \gamma\varepsilon;$$

siendo $\pi' = (d_1, d_2, d_3, \ldots)$. Haciendo $\varepsilon \to 0$ obtenemos que

$$v(x,\pi) \geq \sup_{a\in A(x)} \left\{ r(x,a) + \gamma \sum_{j\in X} p(j \mid x,a) v^*(j) \right\};$$

como $v^*(x) \geq v(x,\pi)$, entonces tenemos

$$v^*(x) \geq \sup_{a\in A(x)} \left\{ r(x,a) + \gamma \sum_{j\in X} p(j \mid x,a) v^*(j) \right\}, \quad \forall x \in X. \tag{2.2.3}$$

Por lo tanto de (2.2.2) y (2.2.3) concluimos el resultado. □

La representación vectorial del operador de Bellman y el Teorema 2.2.1 es presentado en la siguiente subsección (Teorema 2.2.4).

Proposición 2.2.2. *El operador de Bellman es una contracción en $\mathcal{V}$.*

Demostración. Sean $w, u \in \mathcal{V}$ y $x \in X$ arbitrarios. Supongamos que $Lw(x) \geq Lu(x)$, y sea

$$a_x \in \operatorname*{arg\,máx}_{a\in A(x)} \left\{ r(x,a) + \gamma \sum_{j\in X} p(j \mid x,a) w(j) \right\}.$$

Entonces

$$\begin{aligned} Lw(x) - Lu(x) &\leq r(x,a_x) + \gamma \sum_{j\in X} p(j \mid x,a_x) w(j) - r(x,a_x) - \gamma \sum_{j\in X} p(j \mid x,a_x) u(j) \\ &= \gamma \sum_{j\in X} p(j \mid x,a_x)(w(j) - u(j)) \\ &\leq \gamma \sum_{j\in X} p(j \mid x,a_x) \|w - u\| \\ &= \gamma \|w - u\|. \end{aligned}$$

Suponiendo ahora $Lu(x) \geq Lw(x)$, y repitiendo el mismo argumento, se obtiene que

$$|Lw(x) - Lu(x)| \leq \gamma \|w - u\|;$$

tomando el supremo sobre $x \in X$ implica que:

$$\|Lw - Lu\| \leq \gamma \|w - u\|.$$

□

2.2.1. Representación Vectorial de las Ecuaciones de Bellman

La representación vectorial es útil para mostrar la relación del operador de Bellman con las reglas de decisión y el operador dado en la Definición 2.1.1. Para más detalles respecto a esta sección consúltese Puterman [26, capítulo 6].

Definición 2.2.2 (Operador de Bellman vectorial)**.** Bajo las Suposiciones 2.0.1, definimos el operador $L : \mathcal{V} \to \mathcal{V}$ por:

$$Lu \coloneqq \sup_{d \in D^{MD}} \{r_d + \gamma P_d u\} = \sup_{d \in D^{MD}} L_d u. \tag{2.2.4}$$

Observación 2.2.2.

a) De la misma forma que en las Observaciones 2.2.1; si el supremo es obtenido dentro del conjunto en cuestión, el operador se escribe como el máximo.

b) El supremo es evaluado respecto al ordenamiento parcial componente a componente en $\mathcal{V}$[2].

c) El Lema 1.6.1 nos asegura que $Lu \in \mathcal{V}$ para todo $u \in \mathcal{V}$.

d) La definición considera el supremo sobre reglas decisión deterministas, pues se obtiene el mismo valor si se tomara el supremo respecto a las reglas estocásticas. La siguiente Proposición formaliza esto.

Proposición 2.2.3. *Para todo $u \in \mathcal{V}$ y $\gamma \in [0, 1]$,*

$$\sup_{d \in D^{MD}} \{r_d + \gamma P_d u\} = \sup_{d \in D^{MR}} \{r_d + \gamma P_d u\}.$$

[2]Si $h, u \in \mathcal{V}$. $h \geq u$ sii $h(x) \geq u(x), \quad \forall x \in X$.

Demostración. Ver referencia [26, capítulo 6, página 147]. □

De esta forma las ecuaciones de programación dinámica son las siguientes.

Teorema 2.2.4. *Dadas las Suposiciones 2.0.1, se cumple que*

$$Lv^* = v^*;$$

es decir, v^* es solución al sistema de ecuaciones

$$Lu = u, \quad u \in \mathcal{V}. \tag{2.2.5}$$

Demostración. Este es el mismo resultado que el del Teorema 2.2.1, solamente que escrito en forma vectorial. □

Es decir, soluciones a las ecuaciones de optimalidad son puntos fijos de L en $\mathcal{V}$; y de haber solución, esta debe de ser la función de valor óptimo v^*. En la sección siguiente se demuestra esta afirmación.

2.3. Soluciones a las Ecuaciones de Bellman

Se ha mostrado que funciones son solución a la ecuación de punto fijo de los operadores definidos. Lo siguiente es asegurar la unicidad de dichas soluciones.

Teorema 2.3.1. *Bajo las Suposiciones 2.0.1 se tiene lo siguiente.*

a) Existe un único elemento $u^* \in \mathcal{V}$ el cual satisface $Lu^* = u^*$. Además, $u^* = v^*$.

b) Para cada $d \in D^{MR}$, existe un único elemento $u \in \mathcal{V}$ el cual satisface $L_d u = u$. Además, $u = v^{\pi_d}$, siendo $\pi_d = (d, d, d, \ldots)$.

Demostración.

a) Dado que $\mathcal{V}$ es un espacio lineal normado y completo, y la Proposición 2.2.2 dice que L es una contracción en $\mathcal{V}$, entonces el Teorema de Punto Fijo de Banach[3] asegura la existencia y unicidad de un elemento $u^* \in \mathcal{V}$ satisfaciendo $Lu^* = u^*$. Finalmente, el Teorema 2.2.4 nos dice que $u^* = v^*$.

[3]Ver Teorema A.1.1

b) El resultado es directo por el paso anterior tomando $D^{MD} = \{d\}$. □

Puterman [26, capítulo 6, página 147] presenta una demostración utilizando la notación vectorial.

2.4. Existencia de Políticas Óptimas

El siguiente Teorema es de utilidad, asegura poder identificar políticas óptimas simplemente observando si su respectiva función de valor es solución a las ecuaciones de Bellman.

Teorema 2.4.1. *Una política de decisión $\pi^* \in \Pi^{HR}$ es óptima si y sólo si v^{π^*} es solución a las ecuaciones Bellman* (2.2.5)*.*

Demostración. Supongamos que π^* es óptima, entonces $v^{\pi^*} = v^*$ y por el Teorema 2.3.1(a), se tiene que v^{π^*} satisface $Lu = u$.

Supongamos ahora que $Lv^{\pi^*} = v^{\pi^*}$, entonces por el Teorema 2.2.4 sabemos que $v^{\pi^*} = v^*$ y por lo tanto π^* es óptima. □

Definición 2.4.1. Dado $u \in \mathcal{V}$. Una regla de decisión $d_u \in D^{MD}$ se dice que es u-*perfeccionista*[4] si:

$$r_{d_u} + \gamma P_{d_u} u = \max_{d \in D^{MD}} \{r_d + \gamma P_d u\}, \quad (L_{d_u} u = Lu);$$

donde se entiende que el supremo es obtenido dentro el conjunto en cuestión ($Lu = \max_{d \in D^{MD}} \{r_d + \gamma P_d u\}$).

Escrita en términos de componentes, d_u es u-perfeccionista si es tal que:

$$d_u(x) = \arg\max_{a \in A(x)} \left\{ r(x,a) + \gamma \sum_{j \in X} p(j \mid x, a) u(j) \right\}, \quad \forall x \in X.$$

[4]Este nombre es el dado por Puterman en [26, página 152]; en otras referencias, por ejemplo, [30, 35], se les encuentra con el nombre de *avariciosas* (*greedy* en inglés).

Definición 2.4.2. Se define una regla de decisión *conservadora* $d^* \in D^{MD}$ si ésta es v^*-perfeccionista, es decir, si:

$$L_{d^*}v^* = r_{d^*} + \gamma P_{d^*}v^* = v^*$$

Teorema 2.4.2. *Sea X discreto, y supongamos que en la ecuación* (2.2.4) *el supremo es obtenido para todo $u \in \mathcal{V}$. Entonces*

a) existe una regla de decisión conservadora $d^* \in D^{MD}$;

b) para d^* conservadora, la política de decisión estacionaria $\pi_{d^*} = (d^*, d^*, d^*, \ldots)$ es óptima; y

c) $v^* = \sup_{d \in D^{MD}} v^{\pi_d}$, $\pi_d = (d, d, d, \ldots)$.

Demostración.

a) Dado que $v^* \in \mathcal{V}$, entonces por suposición se tiene que $\exists d^* \in D^{MD}$ tal que $L_{d^*}v^* = Lv^*$, y por Teorema 2.2.4, $L_{d^*}v^* = v^*$, por lo tanto d^* es conservadora.

b) Definiendo la política estacionaria $\pi_{d^*} = (d^*, d^*, d^*, \ldots)$, el Teorema 2.1.3 nos dice que $v^{\pi_{d^*}} \in \mathcal{V}$ es la única función que cumple $L_{d^*}v^{\pi_{d^*}} = Lv^{\pi_{d^*}}$ y por lo tanto debe de ser que $v^{\pi_{d^*}} = v^*$, entonces π_{d^*} es óptima.

c) Es una consecuencia directa de (b). □

Se observa que bajo las condiciones del Teorema anterior, se restringe la búsqueda de políticas óptimas a las estacionarias deterministas.

En resumen: las suposiciones del Teorema 2.4.2 implica la existencia de reglas de decisión conservadoras y por lo tanto, políticas deterministas estacionarias óptimas. El siguiente Teorema proporciona otras condiciones para las cuales también existen el mismo tipo de políticas óptimas.

Teorema 2.4.3. *Supongamos que existe*

a) una regla de decisión conservadora, o

b) una política de decisión óptima.

Entonces existe una regla de decisión determinista estacionaria la cual es óptima.

Demostración.

a) Es el resultado del Teorema 2.4.2(b).

b) Supongamos $\pi^* = (d, \pi')$, $d \in D^{MR}$ es la política óptima. Luego

$$v^{\pi^*} = L_d v^{\pi'} \leq L_d v^{\pi^*} \leq L v^{\pi^*} = v^{\pi^*};$$

donde la última igualdad es por el hecho de que π^* es óptima. Se tiene entonces que $L_d v^{\pi^*} = v^{\pi^*}$, pero $v^{\pi^*} = v^*$ y por lo tanto d es conservadora. El resto de la prueba sigue directo de (a). □

La demostración de los siguientes Teoremas puede consultarse en [26, capítulo 6]. El primero de ellos da condiciones suficientes para obtener el supremo en la ecuación (2.2.4) y por lo tanto políticas óptimas.

Teorema 2.4.4. *Suponiendo el conjunto X discreto, y cualquiera de los siguientes afirmaciones*

a) $A(x)$ es finito para todo $x \in X$, o

b) $A(x)$ es compacto, $r(x, a)$ es continua en a para cada $x \in X$, y para cada $x, j \in X$, $p(j \mid x, a)$ es continua en a, o

c) $A(x)$ es compacto, $r(x, a)$ es semi-continua superiormente en a para cada $x \in X$, y para cada $x, j \in X$, $p(j \mid x, a)$ es semi-continua en a.

Entonces existe una política de decisión determinista estacionaria óptima.

Demostración. Ver referencia [26, capítulo 4, página 90]. □

Teorema 2.4.5. *Suponiendo X finito o numerable, entonces, para todo $\varepsilon > 0$ existe una política de decisión determinista estacionaria ε-óptima.*

Demostración. Ver referencia [26, capítulo 6, página 156]. □

2.5. Iteración de Valores

El siguiente algoritmo es la culminación de los resultados de la sección anterior. Fijando $\varepsilon > 0$; se genera con el operador L, la sucesión de funciones (v^n) de acuerdo al Teorema de Punto Fijo de Banach; una vez alcanzado el criterio de paro, se escoge la regla de decisión v^n-perfeccionista. El resultado es una política de decisión determinista estacionaria ε-óptima.

Algoritmo 2.5.1 (Iteración de valores (*Value Iteration*))**.**

1. Fijamos $v^0 \in \mathcal{V}$, $\varepsilon > 0$ y $n = 0$.

2. Para cada $x \in X$, calculamos $v^{n+1}(x)$ como:

$$v^{n+1}(x) = \max_{a \in A(x)} \left\{ r(x,a) + \gamma \sum_{j \in X} p(j \mid x, a) v^n(j) \right\}.$$

3. Si

$$\left\| v^{n+1} - v^n \right\| < \varepsilon(1-\gamma)/2\gamma; \tag{2.5.1}$$

   ir al paso 4. En caso contrario incrementar n en una unidad y regresar al paso 2.

4. Para cada $x \in X$, escogemos

$$d_\varepsilon(x) \in \arg\max_{a \in A(x)} \left\{ r(x,a) + \gamma \sum_{j \in X} p(j \mid x, a) v^n(j) \right\}; \tag{2.5.2}$$

   y el algoritmo concluye.

El siguiente Teorema es respecto a la convergencia del algoritmo.

Teorema 2.5.1. *Sea $v^0 \in \mathcal{V}$, $\varepsilon > 0$ y (v^n) generada mediante el Algoritmo 2.5.1 para $n \geq 1$. Entonces*

a) $\|v^n - v^*\| \to 0$ *cuando* $n \to \infty$.

b) Para todo N finito tal que la ecuación (2.5.1) *se cumple para todo $n \geq N$; entonces la política de decisión determinista estacionaria $\pi_{d_\varepsilon} = (d_\varepsilon, d_\varepsilon, d_\varepsilon, \ldots)$ con d_ε escogida de acuerdo a* (2.5.2)*, es ε-óptima, y*

c) $\|v^{n+1} - v^*\| \leq \varepsilon/2$ *siempre que* (2.5.1) *sea cierta.*

Demostración.

a) Es consecuencia directa del Teorema de Punto Fijo de Banach.

b) Sea N tal que (2.5.1) es cierto para todo $n \geq N$, y sea d_ε escogida de acuerdo a (2.5.2). Calculamos

$$\|v^{\pi_{d_\varepsilon}} - v^*\| \leq \left\|v^{\pi_{d_\varepsilon}} - v^{n+1}\right\| + \left\|v^{n+1} - v^*\right\|;$$

por otro lado, el Teorema 2.1.3 dice que $L_{d_\varepsilon} v^{\pi_{d_\varepsilon}} = v^{\pi_{d_\varepsilon}}$, además, por construcción $L_{d_\varepsilon} v^{n+1} = Lv^{n+1}$, entonces

$$\begin{aligned}
\left\|v^{\pi_{d_\varepsilon}} - v^{n+1}\right\| &= \left\|L_{d_\varepsilon} v^{\pi_{d_\varepsilon}} - v^{n+1}\right\| \\
&\leq \left\|L_{d_\varepsilon} v^{\pi_{d_\varepsilon}} - Lv^{n+1}\right\| + \left\|Lv^{n+1} - v^{n+1}\right\| \\
&= \left\|L_{d_\varepsilon} v^{\pi_{d_\varepsilon}} - L_{d_\varepsilon} v^{n+1}\right\| + \left\|Lv^{n+1} - Lv^{n}\right\| \\
&\leq \gamma\left\|v^{\pi_{d_\varepsilon}} - v^{n+1}\right\| + \gamma\left\|v^{n+1} - v^{n}\right\|;
\end{aligned}$$

es decir

$$\left\|v^{\pi_{d_\varepsilon}} - v^{n+1}\right\| \leq \frac{\gamma}{1-\gamma}\left\|v^{n+1} - v^{n}\right\| < \frac{\varepsilon}{2};$$

de forma similar se obtiene que

$$\left\|v^{n+1} - v^*\right\| < \frac{\varepsilon}{2};$$

lo cual demuestra (c). Finalmente se observa que

$$\|v^{\pi_{d_\varepsilon}} - v^*\| < \varepsilon.$$

□

2.6. Error en las Políticas Aproximadas

Singh & Yee, [28]; presentaron cotas superiores para el error entre la función de valor óptimo y la función de valor siguiendo una política de decisión u-perfeccionista, $u \in \mathcal{V}$.

Teorema 2.6.1. *Dados los conjuntos de estados y acciones, X y A finitos, y $\gamma \in [0,1)$. Si $u \in \mathcal{V}$ es tal que $\|v^* - u\| \leq \varepsilon$; y $\pi_{d_u} = (d_u, d_u, d_u, \ldots)$, con d_u una regla de decisión u-perfeccionista. Entonces*

$$v^*(x) - v(x, \pi_{d_u}) \leq \frac{2\gamma\varepsilon}{1-\gamma}, \quad \forall x \in X.$$

Demostración. Ver referencia [28]. □

Con el resultado anterior y notando que mediante el algoritmo de iteración de valores, se tiene que para $v^{n+1} \in \mathcal{V}$, $\|v^{n+1} - v^*\| \leq \varepsilon/2$, entonces la política de decisión π_{d_ε} (la cual se construyó de manera que sea v^{n+1}-perfeccionista) es en realidad $(\gamma\varepsilon/(1-\gamma))$-óptima.

2.7. Desventajas de Programación Dinámica

Se escogió presentar el algoritmo de iteración de valores por el desarrollo natural que surge a partir de los conceptos e ideas del PDM y el AR. Sin embargo, en la literatura clásica los algoritmos de PD se dividen en tres clases: iteración de valores, iteración de políticas (*policy iteration*) y búsqueda de políticas (*policy search*) [6].

En iteración de valores, mediante el proceso de iteración, se busca la función de valor óptima, de la cual se construye la regla y política de decisión óptima. En cambio, para iteración de políticas, se genera una sucesión de políticas de decisión, donde en cada iteración se mejora la política de acuerdo al cálculo de sus respectivas funciones de valor [6]. Por otro lado los algoritmos de búsqueda de políticas utilizan métodos directos de aproximación [6].

En las referencias [4, 26, 34] desarrollan teoría respecto a iteración de políticas, en [6] desarrolla más la idea de búsqueda de políticas y proporciona más referencias.

A pesar de esta clasificación, todos los algoritmos de PD se basan en la estructura del PDM y por lo tanto comparten las siguientes desventajas que se discuten a continuación.

2.7.1. MDM Perfecto

Al inicio del capítulo se mencionó que los algoritmos de programación dinámica se caracterizan por necesitar el MDM perfecto o completamente conocido. Observamos tal necesidad en el operador de Bellman y en la sucesión de funciones generada mediante iteración de valores. En cada paso se promedia la función anterior, por lo que es necesario conocer todas las probabilidades de transición.

No siempre es posible conocer el MDM por completo (la literatura hace mención a esto como la *maldición del modelo –curse of modeling–* [4]). En situaciones de la vida real, la complejidad de un sistema hace imposible saber las dinámicas de este. Para tales casos, una posible solución es realizar simulaciones o experimentar con el sistema, a fin de realizar una estimación de las probabilidades de transición. Una vez se tengan estas, se procede a los algoritmos de PD. A esta técnica se le conoce como *AR indirecto* (*indirect reinforcement learning* en inglés [35]).

2.7.2. Dimensión del MDM

En iteración de valores, es necesario mantener en la memoria los valores de las dos últimas aproximaciones a la función de valor óptimo, es decir, se necesita reservar memoria para $2|X|$ valores numéricos. Algoritmos donde cada elemento de la función a aproximar es guardado en una lista (o tabla, como se verá en capítulo 3) reciben el adjetivo de *tabulares* (*tabular methods* o *lookup-tables methods* en inglés [30, 35]).

Como se mencionó anteriormente, los algoritmos de PD se caracterizan por necesitar de todas las probabilidades de transición. Para un conjunto de acciones finito, hay $|A||X|^2$ probabilidades de transición. Como se verá en uno de los ejemplos del capítulo 2.8, en ciertos sistemas, las probabilidades de transición pueden definirse como una función definida por casos, sin embargo, si la naturaleza del sistema no admite la definición de alguna función, no queda más remedio que guardar individualmente los $|A||X|^2$ valores numéricos de las probabilidades. Si los conjuntos X y A son muy grandes, tal cosa

resulta imposible. A esta problemática se le conoce en la literatura como *maldición de dimensionalidad* (*curse of dimensionality*) [4].

2.8. Ejemplos

A fin de ejemplificar el modelo de decisión secuencial, presentamos el *mundo de la rejilla* (*gridworld*), que principalmente se enfoca en encontrar el camino más corto entre dos puntos. Es de utilidad en distintas áreas, como lo puede ser la robótica, sistemas de navegación, videojuegos, etc. [27, 36, 38].

Otro modelo presentado es el de *bandidos armados* (*multi-armed bandit*), el cual es de utilidad cuando en alguna situación, se debe de elegir entre distintas opciones posibles. El desarrollo presentado sigue el de Puterman [26, capítulo 3]; otras referencias a consultar que presentan una discusión breve y más completa respecto a los modelos de bandidos son [19, 30], respectivamente.

Para ambos ejemplos, se plantean los PDM correspondientes y sus soluciones son obtenidas mediante el Algoritmo 2.5.1.

2.8.1. Mundo de la Rejilla

Consideremos una rejilla de tamaño $N \times M$ y supongamos que un objeto se encuentra posicionado en una casilla inicial dentro de ella. El objetivo es llegar a una casilla distinta que escogemos como meta. También suponemos que el objeto puede dar un paso a la vez en dirección arriba, abajo, derecha, izquierda o realizando ninguna acción, de acuerdo con la configuración de la rejilla (ver Figura 2.8.1a). Lo anterior se puede describir mediante un MDM estacionario de la siguiente manera:

- El espacio de estados corresponde a las casillas accesibles de la rejilla:

$$X = \{(0,0),(1,0),\ldots,(N,0),\ldots,(0,1),\ldots,(n,1),\ldots,(0,M),\ldots,(N,M)\}.$$

  Denotamos a la casilla objetivo como x_{meta}.

- Representamos simbólicamente el espacio de acciones como los cuatro movimientos posibles en una casilla mas la acción hacer nada:

$$A = \{\uparrow, \downarrow, \leftarrow, \rightarrow, \circlearrowleft\}.$$

- Los subconjuntos de acciones admisibles son los siguientes: para una casilla $x \in X$ rodeada por las cuatro casillas adyacentes, $A(x) = A$; una esquina, por ejemplo, para la primera casilla, $A((0,0)) = \{\uparrow, \rightarrow, \circlearrowleft\}$; en cambio, una casilla $(0,l) \in X$, $0 < l < M$, en el borde lateral izquierdo, $A((0,l)) = \{\uparrow, \downarrow, \rightarrow, \circlearrowleft\}$; etcétera.

- Posicionado el objeto en una casilla $x \in X$, la acción elegida determina completamente la siguiente casilla donde el objeto es movido; entonces la función de transición está concentrada en dicha posición.

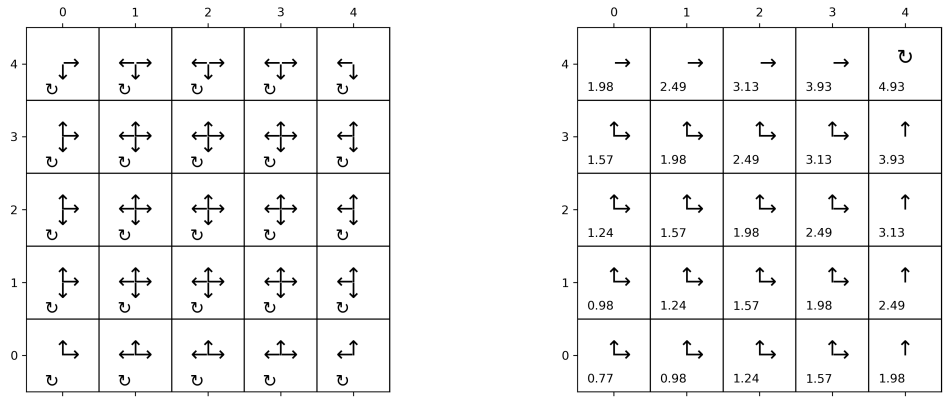
  Por ejemplo $p((1,0) \,|\, (0,0), \uparrow) = 1$, en cambio $p((0,1) \,|\, (0,0), \uparrow) = 0$. De manera análoga se define la función de transición para el resto de estados-acciones admisibles.

- Una función recompensa r de forma que su valor sea 0 para todo par de estados-acciones admisibles, excepto para el par $(x_{\text{meta}}, \circlearrowleft)$:

$$r(x,a) = \begin{cases} 1, & \text{si } (x,a) = (x_{\text{meta}}, \circlearrowleft); \\ 0, & \text{de otra forma.} \end{cases}$$

Dado un factor de descuento $\gamma \in (0,1)$, este incentiva a alcanzar la casilla objetivo en una trayectoria corta. De esta manera, la política óptima es aquella que nos lleva por el camino más corto.

Como se podrá apreciar en las siguientes secciones, el AR es capaz de resolver problemas relacionados al mundo de la rejilla. Estudios más recientes, han explorado versiones del algoritmo presentado en la sección 3.4 utilizando redes neuronales [23, 25, 29, 37, 38]. Los resultados y rendimientos obtenidos no muestran mucha ventaja respecto los algoritmos clásicos como los de Dijkstra o A* [25]. Sin embargo, no por eso deja de ser una línea de investigación activa el uso de AR para el mundo de la rejilla.



(a) Acciones posibles en la configuración de la rejilla. **(b)** Resultados obtenidos mediante iteración de valores.

Figura 2.8.1

2.8.2. Ejemplo: rejilla 5×5

Consideramos el MDM anterior con una rejilla de tamaño 5×5, con casilla objetivo $x_{\text{meta}} = (4, 4)$. No es complicado convencerse que las distintas reglas de decisión óptimas que controlan el sistema son las que se observan en la Figura 2.8.1b. Por ejemplo, una posible regla de decisión óptima es:

$$d^*((x,y)) = \begin{cases} \circlearrowleft, & \text{si } (x,y) = (4,4); \\ \uparrow, & \text{si } (x,y) = (4,l),\ 0 \leq l < 4; \\ \rightarrow, & \text{de otra forma.} \end{cases}$$

De esta forma, siguiendo la política $\pi^* = (d^*, d^*, d^*, \ldots)$ en la casilla objetivo $(4,4)$, ésta dictaría repetir constantemente la acción $\ll\circlearrowleft\gg$, por lo que la función de valor para esa casilla es:

$$\begin{aligned} v((4,4),\pi^*) &= \mathbb{E}\Big(r((4,4),\circlearrowleft) + \gamma r((4,4),\circlearrowleft) + \gamma^2 r((4,4),\circlearrowleft) + \gamma^3 r((4,4),\circlearrowleft) + \cdots\Big) \\ &= 1 + \gamma + \gamma^2 + \gamma^3 + \cdots = \frac{1}{1-\gamma}. \end{aligned}$$

En cambio, en la casilla $(0,0)$, la función de valor es:

$$v((0,0),\pi^*) = \mathbb{E}\Big(0 + \cdots + 0 + \gamma^8 r((4,4),\circlearrowleft) + \gamma^9 r((4,4),\circlearrowleft) + \gamma^{10} r((4,4),\circlearrowleft) + \cdots\Big)$$

$$= \gamma^8 + \gamma^9 + \gamma^{10} + \cdots = \frac{\gamma^8}{1-\gamma}.$$

En general, la función de valor evaluada en cualquier casilla toma el valor $\gamma^\ell/(1-\gamma)$, siendo ℓ el número de pasos necesarios para llegar a la casilla objetivo.

Fijando $\gamma = 0.8$ y $\varepsilon = 0.15$, iteración de valores converge a la función de valor observada en la Figura 2.8.1b en 19 iteraciones, aunque las reglas de decisión óptimas son encontradas en 9 iteraciones. A continuación, se describe brevemente el código hecho:

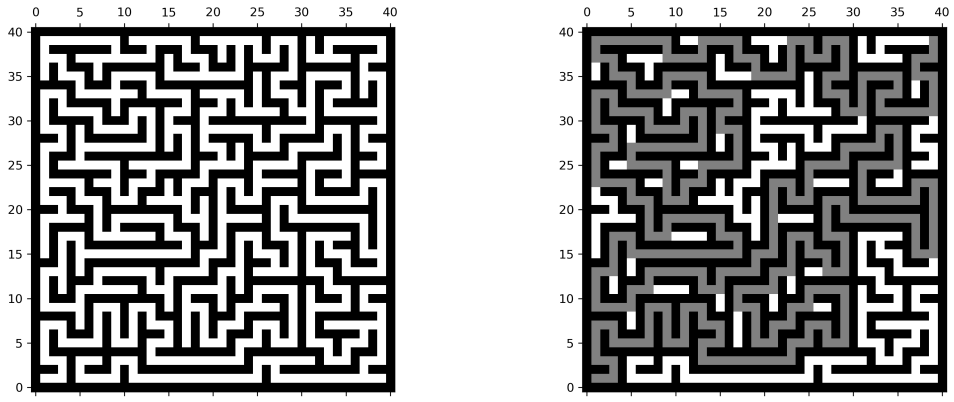
```
#carga de librerias necesarias
import numpy as np
from scipy.spatial import distance
import matplotlib.pyplot as plt
```

después se crean los elementos que describen el modelo:

```
#casilla inicial (0,0), casilla meta (N-1,N-1)
N = 5
goal = (N-1,N-1)
#conjunto de acciones : up, down, right, left, nothing (arriba,
    abajo, derecha, izquierda, no hacer nada)
actions = {"up" : (1,0), "dwn" : (-1,0),
                "rght": (0,1), "lft" : (0,-1), "nthng" : (0,0)}
vn = np.zeros((N,N)) #funcion inicial v_0 = 0 para todos los
    elementos
vni = np.zeros((N, N)) #funcion donde se guarda la nueva
    iteracion
decision_grid = [] #lista para guardar las acciones optimas
iterations = 1 #conteo de iteraciones
e = 0.15 #epsilon
g = 0.80 #gamma
```

mediante dos ciclos «for», se barre todo el espacio de estados y se calcula la primera iteración de valores:

```
for i in range(0,N):
        for r in range(0,N): #estado x = (i,r)
                values = BellmanOp((i,r), vn) #aqui se calcula el
                    operador de Bellman
```



(a) Laberinto tamaño 40×40.

(b) Solución comenzando desde $(1, 1)$.

Figura 2.8.2

```
                vni[i,r] = values["max"] #valor obtenido
                decision_grid.append(values["argmax"]) #la accion
                    codiciosa
```

se continua con todo el esquema de iteración de valores:

```
while distance.chebyshev(np.ravel(vni), np.ravel(vn)) >= e*(1-g)
   /(2*g): #norma del supremo
        vn = np.copy(vni)
        decision_grid = [] #nueva lista para las acciones
            actualizadas
        for i in range(0,N):
                for r in range(0,N): #estado x = (i,r)
                        values = BellmanOp((i,r), vn)
                        vni[i,r] = values["max"]
                        decision_grid.append(values["argmax"])
        iterations += 1
```

Siguiendo este esquema, solo es necesario modificar la estructura de las funciones y objetos que guardan los datos, para que el código quede adaptado a los siguientes ejemplos.

2.8.3. Ejemplo: laberinto

En aplicaciones más realistas es natural la necesidad de considerar obstáculos. El MDM anterior no necesita modificación alguna, mas que la restricción del conjunto de estados a las casillas libres de obstáculo. Como ejemplo, se probó un laberinto de tamaño 40×40 y casilla objetivo $(39, 39)$ (Figura 2.8.2a). Fijando $\gamma = 0.95$, se encontró experimentalmente que se necesita $\varepsilon = 3.8 \times 10^{-10}$ para que el algoritmo realice las suficientes iteraciones (495) a fin de encontrar la regla de decisión óptima para todas las casillas accesibles. Aún así, el resultado es una regla de decisión que permite resolver el laberinto comenzando desde cualquier casilla deseada (Figura 2.8.2b).

Para ambos ejemplos, la implementación realizada permite considerar cualquier configuración del MDM para el mundo de la rejilla.

2.8.4. Bandidos Armados

En la literatura se conoce como «*bandido armado*» (multi-armed bandit) a un modelo de decisión en donde un observador tiene la posibilidad de elegir entre distintos sistemas o acciones. Dependiendo la elección, se obtiene una recompensa y el sistema escogido realiza su transición de estado de acuerdo a su distribución de probabilidad estacionaria; mientras que el estado de los otros sistemas no se ven afectados.

La descripción para un modelo de $K \geq 1$ bandidos armados se describe mediante un MDM de la forma siguiente:

- Sean X^i, $i = 1, 2, 3, \ldots, K$, los espacios de estados del i-ésimo sistema.
- El espacio de estados global es $X = X^1 \times X^2 \times \cdots \times X^K$, es decir, un elemento $x \in X$ es de la forma:

  $$x = (x^1, x^2, \ldots, x^K);$$

  donde cada uno de los x^i, representa el estado en que se encuentra el i-ésimo sistema.
- La acción que se realiza en cada época de decisión es escoger uno de los K sistemas, por lo que $A = \{1, 2, \ldots, K\}$, y $A(x) = A$, $\forall x \in X$.

- Si $i \in A$, p^i representa una probabilidad de transición correspondiente al sistema X^i. De igual forma, r^i es la función recompensa instantánea del sistema X^i.

- La función de transición global es entonces de la siguiente forma:

$$p((s^1, s^2, \ldots, s^K) \,|\, (x^1, x^2, \ldots, x^K), i) = \begin{cases} p^i(s^i \,|\, x^i), & \text{si } s^m = x^m, \forall m \neq i; \\ 0, & \text{de otra forma.} \end{cases}$$

- Mientras que la función recompensa global es tal que:

$$r((x^1, x^2, \ldots, x^K), i) = r^i(x^i).$$

El nombre que recibe este modelo viene de la situación, donde un apostador juega una serie de K máquinas tragamonedas (los bandidos). El jugador debe escoger qué máquinas jugar y en que orden, a la vez que debe ir estimando las probabilidades de victoria de cada tragamoneda.

Los modelos de bandidos armados han encontrado varias aplicaciones en áreas como las de salud, finanzas, comercio, telecomunicaciones, etc. [5]. Una de las principales aplicaciones son en sistemas de recomendación, como lo pueden ser noticias, compras, películas, música, etc. Tales recomendaciones son descritas como un bandido armado, donde la decisión es realizada junto con información adicional (contexto), como lo puede ser, los gustos del consumidor. Estos modelos se encuentran en la literatura como *bandidos contextuales* (*contextual multi-armed bandits*) [21]. La relevancia de estos modelos es tal que se desarrollan algoritmos específicos para su solución [7-9, 18-21].

2.8.5. Ejemplo: bandido 2-armado

El siguiente es un ejemplo modificado de la referencia [26, capítulo 6, página 275]. Un jugador puede escoger entre dos máquinas, las cuales toman los estados $\{x_0, x_1, x_2, x_3\}$ y $\{y_0, y_1, y_2, y_3\}$ respectivamente. En la primera máquina, los estados transitan entre

ellos de acuerdo a la matriz de probabilidad:

$$P_1 = \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} x_0 & x_1 & x_2 & x_3 \end{array} \\ \begin{pmatrix} 0.3 & 0.4 & 0.2 & 0.1 \\ 0.2 & 0.3 & 0.5 & 0.0 \\ 0.1 & 0.0 & 0.8 & 0.1 \\ 0.4 & 0.0 & 0.0 & 0.6 \end{pmatrix} \end{array};$$

mientras que a cada estado x_i, entrega recompensa $i + 7$, $i = 0, 1, 2, 3$.

Por otro lado, en la segunda máquina, estos transitan de acuerdo a la matriz:

$$P_2 = \begin{array}{c} \\ y_0 \\ y_1 \\ y_2 \\ y_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_0 & y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.7 \\ 0.1 & 0.1 & 0.1 & 0.7 \\ 0.1 & 0.1 & 0.1 & 0.7 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \end{array};$$

y al estado y_i, concede recompensa $i + 17$, $i = 0, 1, 2$ y a y_3, le corresponde recompensa cero.

Como puede observarse, en la primera máquina, los estados transitan reiteradamente entre ellos a la vez que entrega una recompensa moderada. La segunda máquina en cambio, otorga más recompensa; pero a riesgo de quedar atrapada en el estado absorbente y_3, el cual no concede recompensa. Dado un factor de descuento γ, el jugador debe escoger cuales máquinas jugar a fin de maximizar la recompensa total.

Lo anterior se plantea como un modelo de bandido 2-armado de la siguiente manera:

- Espacios de estados $X^1 = \{x_0, x_1, x_2, x_3\}$; $X^2 = \{y_0, y_1, y_2, y_3\}$; espacio global $X = X^1 \times X^2$;
- y espacio de acciones $A = \{1, 2\}$.

- La función de transición global es:

$$p((x_l, y_n) \,|\, (x_k, y_m), j) = \begin{cases} P_1(x_k, x_l), & \text{si } j = 1, \text{ y } y_m = y_n; \\ P_2(y_m, y_n), & \text{si } j = 2, \text{ y } x_k = x_l; \\ 0, & \text{de otra forma;} \end{cases}$$

  donde $P_l(x_i, x_j)$, $l = 1, 2$; representa la probabilidad de transición de x_i a x_j (y_i a y_j), es decir, el elemento (i, j) de P_l.

- Las funciones recompensa son:

$$r_1(x_i) = i + 7, \quad i = 0, 1, 2, 3;$$

$$r_2(y_i) = \begin{cases} i + 17, & i = 0, 1, 2; \\ 0, & i = 3. \end{cases}$$

  y función recompensa global:

$$r((x_l, y_m), j) = \begin{cases} r_1(x_l) & \text{si } j = 1; \\ r_2(y_m) & \text{si } j = 2. \end{cases}$$

Dados $\gamma = 0.95$ y $\varepsilon = 0.15$, iteración de valores encuentra la regla de decisión ε-óptima d_ε en 151 iteraciones. Los resultados se muestran en la Tabla 2.8.1. Por ejemplo, la regla de decisión para el estado (x_2, y_1), es $d_\varepsilon(x_2, y_1) = 2$, y comenzando a jugar a partir de ese estado siguiendo la política $\pi = (d_\varepsilon, d_\varepsilon, d_\varepsilon, \ldots)$, obtendría aproximadamente $v^{151}(x_2, y_1) = 186.77$ unidades de recompensa. De forma análoga se lee el resto de la tabla.

	y_0		y_1		y_2		y_3	
(x_i, y_j)	d_ε	v^{151}	d_ε	v^{151}	d_ε	v^{151}	d_ε	v^{151}
x_0	2	182.67	2	183.67	2	184.67	1	170.42
x_1	2	184.05	2	185.05	2	186.05	1	171.9
x_2	2	185.77	2	186.77	2	187.77	1	173.76
x_3	2	185.86	2	186.86	2	187.86	1	173.85

Tabla 2.8.1. Regla de decisión y recompensa descontada esperada encontrada mediante iteración de valores.

Capítulo 3

Aprendizaje por Refuerzo

Como se mencionó en la introducción, en la literatura es común referirnos a aprendizaje por refuerzo como la familia de algoritmos para los cuales no es necesario conocer por completo su respectivo PDM. Esto marca una clara separación respecto a los algoritmos de PD.

Sin embargo, es necesario mencionar que el algoritmo presentado en este capítulo (sección 3.4) corresponde a una familia contenida en los algoritmos de AR; y además, de las ideas presentadas, surgen otros tipos de algoritmos con enfoques muy distintos.

La mayor parte de las definiciones y resultados pueden encontrarse con mayor o menor detalle en las referencias [4, 6, 11, 30, 35].

Recordamos que en este trabajo solo se presenta el caso de horizonte infinito; además de que en este capítulo se van a considerar las siguientes suposiciones:

Suposición 3.0.1.

a) Espacio de estados X y conjunto de acciones A finitos.

b) Función de recompensa cuya varianza es acotada: existe $R \in \mathbb{R}$ tal que:

$$\mathrm{Var}(r(X_t, Y_t)) \leq R, \quad t \geq 0.$$

c) Factor de descuento $\gamma \in [0, 1)$.

La Suposición 3.0.1(a) es utilizada en todo el capítulo.

3.1. Q-funciones de Valor

Una función de valor retorna el rendimiento esperado de una política de decisión comenzando desde un estado inicial. Con el propósito de medir el mismo rendimiento, pero ahora considerando un estado y acción inicial, se introduce la siguiente definición.

Definición 3.1.1. Dados $(x, a) \in \mathbb{K}$ y $\pi \in \Pi^K$ arbitrarios. Sea $(\Omega, \mathcal{H}, P_x^\pi, \{X_t\})$ el respectivo PDM con estado-acción inicial $(X_0, Y_0) = (x, a)$. Dado $\gamma \in [0, 1)$, la función $Q : \mathbb{K} \times \Pi^K \to \mathbb{R}$ definida como:

$$Q((x,a),\pi) \coloneqq \mathbb{E}^\pi\left(\sum_{t=0}^{\infty} \gamma^t r(X_t, Y_t) \,|\, X_0 = x, Y_0 = a\right) \tag{3.1.1}$$

$$= r(x,a) + \mathbb{E}^\pi\left(\sum_{t=1}^{\infty} \gamma^t r(X_t, Y_t) \,|\, X_0 = x, Y_0 = a\right); \tag{3.1.2}$$

recibe el nombre de *Q-función de valor*. A fin de simplificar los argumentos, se denotará como $Q^\pi(x, a)$.

Su finalidad es la misma que las funciones de valor: evaluar el rendimiento de una política de decisión partiendo de un par estado-acción inicial. Ambas funciones se relacionan de acuerdo al siguiente lema.

Lema 3.1.1. *Dada una política de decisión arbitraria $\pi = (d_0, d_1, d_2, d_3, \ldots) \in \Pi^K$, se tiene que*

$$v(x,\pi) = \sum_{a \in A(x)} Q^\pi(x,a)\pi_0(a \,|\, x); \tag{3.1.3}$$

$$Q^\pi(x,a) = r(x,a) + \gamma \mathbb{E}^\pi\big(v(X_1, \pi') \,|\, X_0 = x, Y_0 = a\big); \tag{3.1.4}$$

siendo $\pi' = (d_1, d_2, d_3, \ldots)$.

Demostración. Sean $x \in X$ y $\pi \in \Pi^K$, calculamos

$$\begin{aligned} v(x,\pi) &= \mathbb{E}^\pi\left(\sum_{t=0}^{\infty} \gamma^t r(X_t, Y_t) \,|\, X_0 = x\right) \\ &= \sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \mathbb{P}^\pi(X_t = x_t, Y_t = a_t \,|\, X_0 = x) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \left[\sum_{a \in A(x)} \mathbb{P}^\pi(X_t = x_t, Y_t = a_t \,|\, X_0 = x, Y_0 = a) \pi_0(a \,|\, x) \right] \\
&= \sum_{a \in A(x)} \left[\sum_{t=0}^{\infty} \gamma^t r(x_t, a_t) \mathbb{P}^\pi(X_t = x_t, Y_t = a_t \,|\, X_0 = x, Y_0 = a) \right] \pi_0(a \,|\, x) \\
&= \sum_{a \in A(x)} \mathbb{E}^\pi \left(\sum_{t=0}^{\infty} \gamma^t r(X_t, Y_t) \,|\, X_0 = x, Y_0 = a \right) \pi_0(a \,|\, x) \\
&= \sum_{a \in A(x)} Q^\pi(x, a) \pi_0(a \,|\, x).
\end{aligned}
$$

Por otro lado

$$
\begin{aligned}
Q^\pi(x, a) &= r(x, a) + \mathbb{E}^\pi \left(\sum_{t=1}^{\infty} \gamma^t r(X_t, Y_t) \,|\, X_0 = x, Y_0 = a \right) \\
&= r(x, a) + \gamma \mathbb{E}^\pi \left(\mathbb{E}^{\pi'} \left(\sum_{t=1}^{\infty} \gamma^{t-1} r(X_t, Y_t) \,|\, X_1 = x_1 \right) \,|\, X_0 = x, Y_0 = a \right) \\
&= r(x, a) + \gamma \mathbb{E}^\pi (v(X_1, \pi') \,|\, X_0 = x, Y_0 = a).
\end{aligned}
$$

□

Se observa que si $\pi = (d, d, d, \ldots)$, con $d \in D^{MD}$, la ecuación (3.1.3) dice $v(x, \pi) = Q^\pi(x, d(x))$.

Por otro lado, la ecuación (3.1.4) sugiere una definición alternativa a las Q-funciones de valor. Definimos $Q : \mathbb{K} \times \Pi^K \to \mathbb{R}$ como:

$$
Q((x, a), \pi) \coloneqq r(x, a) + \gamma \mathbb{E}(v(X_1, \pi) \,|\, X_0 = x, Y_0 = a);
$$

siendo la esperanza $\mathbb{E}$ respecto a la medida de probabilidad $p(\cdot \,|\, x, a)$. Es decir, interpretamos a $Q^\pi(x, a)$ como la recompensa inmediata del estado-acción (x, a) más la esperanza condicionada de la función de valor siguiendo la política π.

Al igual que con la función de valor óptimo, se tiene el análogo con las Q-funciones.

Definición 3.1.2. La función $Q^* : \mathbb{K} \to \mathbb{R}$ definida como:

$$
Q^*(x, a) \coloneqq \sup_{\pi \in \Pi^{HR}} Q^\pi(x, a); \tag{3.1.5}
$$

recibe el nombre de *Q-función óptima.*

Por lo tanto, planteamos el problema de control óptimo diciendo que una política de decisión $\pi^* \in \Pi^{HR}$ es óptima (ε-óptima) cuando satisface:

$$Q^*(x,a) = Q^{\pi^*}(x,a), \quad \forall (x,a) \in \mathbb{K};$$
$$(Q^*(x,a) \leq Q^{\pi^*}(x,a) + \varepsilon, \quad \forall (x,a) \in \mathbb{K}).$$

Proposición 3.1.2. *Para todo $x \in X$ se cumple que*

$$v^*(x) = \max_{a \in A(x)} Q^*(x,a).$$

Demostración. Sea $x \in X$ y $\pi \in \Pi^{HR}$ arbitrarios. De la ecuación (3.1.3) tenemos que la función de valor es el promedio de las Q-funciones de valor, entonces debe de ser que

$$v(x,\pi) \leq \max_{a \in A(x)} Q^{\pi}(x,a) \leq \max_{a \in A(x)} Q^*(x,a);$$

y como π es arbitrario

$$v^*(x) \leq \max_{a \in A(x)} Q^*(x,a).$$

Sea $\pi^* = (d^*, \pi^{**}) \in \Pi^{HR}$ una política óptima, usando la ecuación (3.1.4) se tiene que

$$\begin{aligned} Q^{\pi^*}(x,a) &= \mathbb{E}^{\pi^*}(r(x,a) + \gamma v(X_1, \pi^{**}) \,|\, X_0 = x, Y_0 = a) \\ &\leq \mathbb{E}^{\pi^*}\big(r(x,a') + \gamma v(X_1, \pi^{**}) \,|\, X_0 = x\big) \\ &= v(x, \pi^*); \end{aligned}$$

por lo tanto

$$Q^*(x,a) \leq v^*(x);$$

y por ser $a \in A(x)$ arbitraria

$$\max_{a \in A(x)} Q^*(x,a) \leq v^*(x).$$

□

Observación 3.1.1. En la proposición anterior es posible cambiar máximo por supremo en caso de que se estén considerando conjuntos de estados y acciones numerables.

3.2. Operador de Bellman para Q-funciones

Representamos por $(\mathcal{Q}, \|\cdot\|)$ el espacio vectorial normado de funciones reales acotadas sobre $\mathbb{K}$ con la norma del supremo.

Definición 3.2.1 (Operador de Bellman para Q-funciones)**.** Considerando las Suposiciones 3.0.1, definimos el operador $T : \mathcal{Q} \to \mathcal{Q}$:

$$Tq(x,a) \coloneqq r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) \max_{a' \in A(j)} q(j,a'), \quad (x,a) \in \mathbb{K}. \tag{3.2.1}$$

Teorema 3.2.1. *Dadas las Suposiciones 3.0.1, se cumple que*

$$TQ^*(x,a) = Q^*(x,a), \quad \forall (x,a) \in \mathbb{K};$$

es decir, Q^* es solución al sistema de ecuaciones

$$q(x,a) = r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) \max_{a' \in A(j)} q(j,a'), \quad \forall \in (x,a) \in \mathbb{K}, q \in \mathcal{Q}. \tag{3.2.2}$$

Tal sistema se conoce como ecuaciones de Bellman para Q-funciones.

Demostración. Sean $(x,a) \in \mathbb{K}$ y $\pi = (d_0, d_1, d_2, d_3, \ldots) \in \Pi^{MR}$ arbitrarios. Calculamos

$$\begin{aligned} Q^\pi(x,a) &= r(x,a) + \gamma \mathbb{E}^\pi\big(v(X_1, \pi') \,|\, X_0 = x, Y_0 = a\big) \\ &= r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) v(j, \pi') \\ &\leq r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) v^*(j) \\ &= r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) \max_{a' \in A(j)} Q^*(j,a'); \end{aligned}$$

siendo $\pi' = (d_1, d_2, d_3, \ldots)$. Como π es arbitrario, tenemos que

$$Q^*(x,a) \leq r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) \max_{a' \in A(j)} Q^*(j,a').$$

Dado $(x,a) \in \mathbb{K}$, sea $\varepsilon > 0$. Construimos $\pi \in \Pi^{MR}$ de forma que

$$v(x,\pi) \geq v^*(x) - \varepsilon;$$

entonces

$$\begin{aligned} Q^\pi(x,a) &= r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) v(j,\pi') \\ &\geq r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a)(v^*(j) - \varepsilon) \\ &= r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) v^*(j) - \gamma\varepsilon \\ &= r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) v^*(j) - \gamma\varepsilon \\ &= r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) \max_{a' \in A(j)} Q^*(j,a') - \gamma\varepsilon; \end{aligned}$$

siendo $\pi' = (d_1, d_2, d_3, \ldots)$. Haciendo $\varepsilon \to 0$, tenemos

$$Q^\pi(x,a) \geq r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) \max_{a' \in A(j)} Q^*(j,a');$$

pero $Q^*(x,a) \geq Q^\pi(x,a)$, por lo tanto debe de ser que

$$Q^*(x,a) \geq r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) \max_{a' \in A(j)} Q^*(j,a').$$

□

Usando el resultado anterior y junto a la Proposición 3.1.2, se puede expresar a Q^* como:

$$Q^*(x,a) = r(x,a) + \gamma \sum_{j \in X} p(j \,|\, x,a) v^*(j), \quad (x,a) \in \mathbb{K}.$$

Proposición 3.2.2. *El operador de Bellman para Q-funciones es una contracción en* $\mathcal{Q}$.

Demostración. Dados $q, w \in \mathcal{Q}$, se tiene que

$$\begin{aligned}
\|Tq - Tw\| &= \max_{(x,a)\in\mathbb{K}} \left| \gamma \sum_{j\in X} p(j \mid x, a) \left(\max_{a'\in A(j)} q(j, a') - \max_{a'\in A(j)} w(j, a') \right) \right| \\
&\leq \max_{(x,a)\in\mathbb{K}} \gamma \sum_{j\in X} p(j \mid x, a) \max_{a'\in A(j)} \left| q(j, a') - w(j, a') \right| \\
&\leq \max_{(x,a)\in\mathbb{K}} \gamma \sum_{j\in X} p(j \mid x, a) \|q - w\| \\
&= \gamma \|q - w\|.
\end{aligned}$$

□

3.3. Existencia de Políticas Óptimas

Como consecuencia de los dos resultados de la sección anterior, se tiene el siguiente Teorema análogo a 2.3.1 y 2.4.1.

Teorema 3.3.1. *Bajo las suposiciones 3.0.1 se tiene lo siguiente.*

a) Existe un único elemento $q^* \in \mathcal{Q}$ *el cual satisface* $Tq^* = q^*$. *Además,* $q^* = Q^*$.

b) Una política de decisión $\pi^* \in \Pi^{HR}$ *es óptima si y sólo si* Q^{π^*} *es solución a las ecuaciones Bellman* (3.2.2).

Demostración. Se prueba de forma similar a como se hizo en los Teoremas 2.3.1 y 2.4.1. □

Los resultados de la sección 2.4 son de ayuda para asegurar la existencia de políticas óptimas obtenidas por medio de Q-funciones.

Si $q \in \mathcal{Q}$ y $d \in D^{MD}$, definimos a $q_d \in \mathcal{V}$ como $q_d(x) \coloneqq q(x, d(x))$.

Definición 3.3.1. Dado $q \in \mathcal{Q}$. Una regla de decisión $d_q \in D^{MD}$ es *q-avariciosa* (*q-greedy* en inglés) si:

$$q_{d_q} = \max_{d\in D^{MD}} q_d.$$

Escrita en términos de componentes, d_q es q-avariciosa si es tal que:

$$d_q(x) = \arg\max_{a \in A(x)} q(x, a), \quad \forall x \in X.$$

Corolario 3.3.2. *Si $d^* \in D^{MD}$ es una regla de decisión Q^*-avariciosa, entonces la política de decisión $\pi^* = (d^*, d^*, d^*, \ldots)$ es óptima.*

Demostración. Dado que

$$\begin{aligned} v^*(x) &= \max_{a \in A(x)} Q^*(x, a) \\ &= Q^*(x, d^*(x)) \\ &= r(x, d^*(x)) + \gamma \sum_{j \in X} p(j \mid x, d^*(x)) v^*(j) \\ &= r_{d^*}(x) + \gamma [P_{d^*} v^*](x); \end{aligned}$$

y como $Lv^* = v^*$, concluimos que d^* también es una regla de decisión conservadora. Por lo tanto, el Teorema 2.4.2(b) nos dice que π^* es óptima. □

Dado que los conjuntos X y A son finitos, siempre se puede construir una regla de decisión Q^*-avariciosa, y por lo tanto siempre existe al menos una política de decisión óptima.

3.4. Q-learning

Con los resultados presentados hasta ahora, es posible formular una versión de iteración de valores para las Q-funciones, del cual se puede obtener políticas de decisión óptimas. Tal algoritmo puede encontrarse en la literatura como *Q-iteration* [6, 11]; allí mismo también se presenta una versión de iteración de políticas para las Q-funciones. Estos algoritmos siguen formando parte de PD, por lo que continúan con las mismas desventajas ya mencionadas en la sección 2.7.

A fin de prescindir del modelo completo correspondiente al PDM, Watkins [32, 33] desarrolló el algoritmo *Q-learning*, basado en los algoritmos tipo Robbins–Monro[1] y la

[1] Ver Teorema A.2.1

teoría de *aproximación estocástica*.

Algoritmo 3.4.1 (Q-learning)**.**

1. Fijamos $Q^0 \in \mathcal{Q}$; $N > 0$; $n = 0$; y (α_n) una sucesión que cumple con $\sum_n \alpha_n(x,a) = \infty$, $\sum_n \alpha_n^2(x,a) < \infty$, $\forall (x,a) \in \mathbb{K}$.

2. Seleccionamos un estado $X_n = x_n \in X$ de manera arbitraria. Después escogemos $Y_n = a_n \in A(x_n)$ siguiendo algún procedimiento.

3. Observamos los resultados de realizar la acción a_n en el sistema. Es decir, obtenemos $r(x_n, a_n)$ y el nuevo estado del sistema $X_{n+1} = s$.

4. Para todo $(x,a) \in \mathbb{K}$, calculamos $Q^{n+1}(x,a)$ como:

$$Q^{n+1}(x,a) = \begin{cases} (1-\alpha_n(x,a))Q^n(x,a) & \\ \quad + \alpha_n(x,a)(r(x,a) + \gamma W_n(s)), & \text{si } (x,a) = (x_n, a_n); \\ Q^n(x,a), & \text{si } (x,a) \neq (x_n, a_n); \end{cases} \tag{3.4.1}$$

   siendo $W_n(x) \coloneqq \max_{a' \in A(x)} Q^n(x,a')$.

5. Si $n+1 = N$, ir al paso 6. En caso contrario incrementar n en una unidad y regresar al paso 2 con $X_n = s$.

6. Para cada $x \in X$, escogemos

$$d_N(x) = \arg\max_{a \in A(x)} Q^{n+1}(x,a); \tag{3.4.2}$$

   y el algoritmo concluye.

Una libertad que se tiene en el paso 2 es la acción a tomar. Para tal elección aparecen dos opciones naturales por las que uno puede optar; la primera es «explotar» el conocimiento obtenido de las iteraciones realizando acciones codiciosas: $Y_n = \arg\max_{a \in A(x_n)} Q^n(x_n, a)$; la segunda opción natural, es escoger Y_n de manera aleatoria con probabilidad $1/|A(x_n)|$, de esta forma el algoritmo tiene más oportunidad de «explorar» el rendimiento de otras acciones.

Ambas opciones pueden combinarse de la siguiente manera: dado $\varepsilon \in [0,1]$ escogemos la acción en la iteración n como:

$$Y_n = \begin{cases} \arg\max_{a \in A(x_n)} Q^n(x_n, a), & \text{con probabilidad } 1-\varepsilon; \\ a_n \sim \text{Uniforme}(A(x_n)), & \text{con probabilidad } \varepsilon. \end{cases}$$

El número ε recibe el nombre de *probabilidad de exploración*, y las acciones elegidas de esa forma reciben el nombre de ε-*codiciosas* (ε-*greedy*), éstas permiten un balance entre la explotación y exploración de las Q-funciones. Incluso puede considerarse una sucesión (ε_n) de probabilidades de transición, que para iteraciones muy avanzadas, favorezca más la elección de acciones codiciosas (ver el algoritmo de Q-learning de la referencia [6, página 30]).

El balance entre explotar el conocimiento y explorar el sistema (*exploration - exploitation trade-off* [35]) es algo que tener en cuenta en los algoritmos de AR. En el caso del apostador que se mencionó en la sección 2.8.4, éste juega con el fin de maximizar sus ganancias de acuerdo (explotando) a su experiencia que tiene con las distintas tragamonedas; sin embargo, jugar (explorar) aleatoriamente otras tragamonedas le puede llevar a nuevas estrategias que mejoren a largo plazo su ganancia.

Respecto a la sucesión (α_n), sus términos reciben el nombre de *tamaño de paso* o *tasa de aprendizaje* (*step size*, *learning rate* [11]). Como puede apreciarse en el proceso de actualización del algoritmo, su función es asignar un peso a los términos $Q^n(x,a)$ y $r(x,a) + \gamma W_n(s)$ con los que la nueva función se actualiza. Algunos ejemplos son (otros pueden consultarse en [11, capítulo 7]):

$$\alpha_n = \frac{1}{n}; \qquad\qquad \alpha_n = \frac{\log(n)}{n}. \tag{3.4.3}$$

Finalmente, mencionamos la falta de necesidad de las probabilidades de transición en el paso 3, únicamente se requiere de la realización del siguiente estado. Tal resultado puede ser obtenido mediante simulación o una interacción en tiempo real con el sistema. Algoritmos que requieren el uso de un simulador se dice que son *implementaciones fuera de línea* (*off-line*), mientras que los algoritmos que utilizan la observación en tiempo real del verdadero sistema, se denominan *implementaciones en línea* (*on-line*) [11, 35].

El siguiente teorema respecto la convergencia del algoritmo de Q-learning fue probado por Watkins & Dayan en [33], sin embargo las pruebas dadas por Jaakkola et al. [15, 16], siguen más las ideas de la aproximación estocástica.

Teorema 3.4.1. *Sea $Q^0 \in \mathcal{Q}$ y (Q^n) generada mediante el Algoritmo 3.4.1 para $n \geq 1$. Bajo las Suposiciones 3.0.1 se tiene que $\|Q^n - Q^*\| \to 0$ cuando $n \to \infty$.*

Demostración. La demostración utiliza el resultado de convergencia estocástica presentado en el Teorema A.2.1. Dado $(x, a) \in \mathbb{K}$ y suponiendo $X_{n+1} = s$, al proceso de actualización restamos de ambos lados de la igualdad el término $Q^*(x, a)$, y al lado derecho sumamos y restamos $\alpha_n(x, a)Q^*(x, a)$; de forma que se tiene

$$\begin{aligned} Q^{n+1}(x,a) - Q^*(x,a) = (1 - \alpha_n(x,a))(Q^n(x,a) - Q^*(x,a)) \\ + \alpha_n(x,a)(r(x,a) + \gamma W_n(s) - Q^*(x,a)); \end{aligned}$$

definiendo

$$\begin{aligned} \beta_n &\coloneqq \alpha_n; \\ \Delta_n(x,a) &\coloneqq Q^n(x,a) - Q^*(x,a); \\ F_n(x,a) &\coloneqq r(x,a) + \gamma W_n(s) - Q^*(x,a); \end{aligned}$$

observamos que el procedimiento de actualización tiene la forma del proceso en la ecuación (A.2.1). Ahora, la esperanza condicional de F_n respecto a la historia del proceso es

$$\begin{aligned} \mathbb{E}(r(x,a) + \gamma W_n(X_{n+1}) - Q^*(x,a) \,|\, \mathcal{F}_n) &= \sum_{s \in X} p(s \,|\, x, a) \Big[r(x,a) \\ &\quad + \gamma \max_{a' \in A(s)} Q^n(s,a') - Q^*(x,a) \Big] \\ &= TQ^n(x,a) - Q^*(x,a) \\ &= TQ^n(x,a) - TQ^*(x,a); \end{aligned}$$

entonces

$$\begin{aligned}\|\mathbb{E}(F_n(x,a)\,|\,\mathcal{F}_n)\| &= \|TQ^n(x,a) - TQ^*(x,a)\| \\ &\leq \gamma\|Q^n(x,a) - Q^*(x,a)\| \\ &= \gamma\|\Delta_n\|.\end{aligned}$$

Por otro lado, la varianza

$$\begin{aligned}\mathrm{Var}(F_n(x,a)\,|\,\mathcal{F}_n) &= \mathbb{E}((r(x,a) + \gamma W_n(X_{n+1}) - Q^*(x,a) \\ &\qquad - TQ^n(x,a) + TQ^*(x,a))^2\,|\,\mathcal{F}_n) \\ &= \mathbb{E}((r(x,a) + \gamma W_n(X_{n+1}) - TQ^n(x,a))^2\,|\,\mathcal{F}_n) \\ &= \mathrm{Var}(r(x,a) + \gamma W_n(X_{n+1})\,|\,\mathcal{F}_n) \\ &= \mathrm{Var}(r(x,a)\,|\,\mathcal{F}_n) + \gamma^2\mathrm{Var}(W_n(X_{n+1})\,|\,\mathcal{F}_n) \\ &\qquad + \gamma\mathrm{Cov}(r(x,a), W_n(X_{n+1})) \\ &\leq \mathrm{Var}(r(x,a)\,|\,\mathcal{F}_n) + \gamma^2\mathrm{Var}(W_n(X_{n+1})\,|\,\mathcal{F}_n) + \gamma \\ &\leq R + \gamma + \gamma^2\Big(\mathbb{E}(W_n^2(X_{n+1})\,|\,\mathcal{F}_n) - \mathbb{E}^2(W_n(X_{n+1})\,|\,\mathcal{F}_n)\Big) \\ &\leq R + \gamma + \gamma^2\mathbb{E}(W_n^2(X_{n+1})\,|\,\mathcal{F}_n) \\ &\leq R + \gamma + \gamma^2\|Q^n\|^2 \\ &= R + \gamma + \gamma^2\|Q^* + \Delta_n\|^2 \\ &\leq R + \gamma + \gamma^2(\|Q^*\| + \|\Delta_n\|)^2 \\ &\leq R + \gamma + \gamma^2\Big(\frac{M}{1-\gamma} + \|\Delta_n\|\Big)^2 \\ &= C(1 + \|\Delta_n\|)^2;\end{aligned}$$

siendo $M = \max_{(x,a)\in\mathbb{K}} r(x,a)$, y $C = \Big(R + \gamma + \gamma^2\Big(\frac{M}{1-\gamma} + \|\Delta_n\|\Big)^2\Big)/(1 + \|\Delta_n\|)^2$.

Por lo tanto todas las condiciones del Teorema se cumplen y concluimos que $\Delta_n \to 0$ c.s. □

Observación 3.4.1. La convergencia también sucede cuando $\gamma = 1$. Tal caso es discutido en las referencias [15, 16].

3.4.1. Sistemas Episódicos

Puede ser que la dinámica de un sistema contenga un estado final o bien que sea un sistema de horizonte finito. En ambos casos, nos referimos a una configuración del sistema y su respectiva evolución hasta su estado final u horizonte finito como un *episodio*. Con los debidos ajustes al Algoritmo 3.4.1, es posible resolver sistemas episódicos.

En el caso de un sistema con estado terminal $x_T \in X$ (o varios), una modificación al quinto paso del algoritmo es:

5. Si $s = x_T$, ir al paso 6. En caso contrario incrementar n en una unidad y regresar al paso 2 con $X_n = s$.

Junto con una función recompensa adecuada, el algoritmo aprendería a controlar el sistema, con el fin de alcanzar o evitar tal estado terminal. Un problema surge cuando tales estados terminales son accedidos en épocas de decisión muy tempranas; pues el algoritmo no tendría el suficiente tiempo para lograr una buena aproximación (lo mismo puede suceder para un sistema con horizonte finito muy corto). Tal problema es solventado repitiendo todos los pasos completos del algoritmo una cantidad fija de episodios, empezando cada episodio con función inicial la última Q-función aprendida. Es precisamente esta versión del algoritmo Q-learning presentado en las referencias [30, 35]

3.5. Error en las Q-funciones de Valor Aproximadas

En la misma referencia [28], Singh & Yee, presentan un resultado similar al Teorema 2.6.1 para Q-learning.

Teorema 3.5.1. *Dados los conjuntos de estados y acciones, X y A finitos, y $\gamma \in [0, 1)$. Si $q \in \mathcal{Q}$ es tal que $\|Q^* - q\| \leq \varepsilon$; y $\pi_{d_q} = (d_q, d_q, d_q, \ldots)$, con d_q una regla de decisión q-avariciosa. Entonces*

$$Q^*(x, \pi^*(x)) - Q^{\pi_{d_q}}(x, d_q(x)) \leq \frac{2\varepsilon}{1-\gamma}; \tag{3.5.1}$$

siendo $\pi^* = (d^*, d^*, d^*, \ldots)$, $d^* \in D^{MD}$ *una política de decisión óptima.*

Demostración. Sea $z \in X$ tal que

$$Q^*(x, \pi^*(x)) - Q^{\pi_{d_q}}(x, d_q(x)) \leq Q^*(z, \pi^*(z)) - Q^{\pi_{d_q}}(z, d_q(z)), \quad \forall x \in X.$$

Escribimos $d^*(z) = a$ y $d_q(z) = b$. Dado que d_q es q-avariciosa, entonces debe de ser que

$$q(z, a) \leq q(z, b);$$

y utilizando la suposición

$$Q^*(z, a) - \varepsilon \leq Q^*(z, b) + \varepsilon;$$

reescribiendo

$$r(z, a) - r(z, b) \leq 2\varepsilon + \gamma \sum_{j \in X} p(j \mid z, b) \max_{a' \in A(j)} Q^*(j, a') - \gamma \sum_{j \in X} p(j \mid z, a) \max_{a' \in A(j)} Q^*(j, a').$$

Por otro lado

$$\begin{aligned}
Q^*(z, a) - Q^{\pi_{d_q}}(z, b) &= r(z, a) + \gamma \sum_{j \in X} p(j \mid z, a) \max_{a' \in A(j)} Q^*(j, a') \\
&\quad - r(z, b) - \gamma \sum_{j \in X} p(j \mid z, b) v(j, \pi_{d_q}) \\
&\leq 2\varepsilon + \gamma \sum_{j \in X} p(j \mid z, b) \left(\max_{a' \in A(j)} Q^*(j, a') - v(j, \pi_{d_q}) \right) \\
&= 2\varepsilon + \gamma \sum_{j \in X} p(j \mid z, b) \left(\max_{a' \in A(j)} Q^*(j, a') - Q^{\pi_{d_q}}(j, d_q(j)) \right) \\
&\leq 2\varepsilon + \gamma \sum_{j \in X} p(j \mid z, b) (Q^*(z, a) - Q^{\pi_{d_q}}(z, b));
\end{aligned}$$

por lo tanto

$$Q^*(z, a) - Q^{\pi_{d_q}}(z, b) \leq \frac{2\varepsilon}{1 - \gamma}.$$

□

Observación 3.5.1. El resultado original (ecuación (3.5.1)) presentado en la referencia

[28] es el siguiente:

$$Q^*(x, \pi^*(x)) - q(x, d_q(x)) \leq \frac{2\varepsilon}{1-\gamma}.$$

Suponemos se trata de un error de escritura, y el resultado correcto es el presentado en este trabajo.

El resultado anterior proporciona el error de aproximación para una política de decisión obtenida mediante Q-learning. Si (Q^n) es generada mediante el Algoritmo 3.4.1; dado $\varepsilon > 0$, existe $N \geq 0$ tal que $\left\|Q^N - Q^*\right\| \leq (1-\gamma)\varepsilon/2$. Entonces la regla de decisión $d_N \in D^{MD}$ construida de forma que sea Q^N-avariciosa, cumple que:

$$Q^*(x, a) - Q^{\pi_N}(x, a) \leq \varepsilon, \quad \forall (x, a) \in \mathbb{K};$$

siendo $\pi_N = (d_N, d_N, d_N, \ldots)$. Es decir, π_N es una política ε-óptima.

Si bien se puede conocer el error de aproximación, el resultado no deja de ser teórico, pues es necesario conocer por completo la función Q^*, lo cual en la práctica no siempre es posible.

3.6. Desventajas de Q-learning

No es difícil observar que la teoría referente a Q-learning sigue las ideas de PD. Hay relaciones y recursividad entre las Q-funciones y las funciones de valor; se designa una Q-función óptima y un operador de programación dinámica que resulta ser una contracción en el espacio de estados-acciones admisibles y cuyo punto fijo es esta función óptima. La relevancia de este último resultado se aprecia en la prueba respecto a la convergencia del algoritmo.

Los dos siguientes puntos son las principales diferencias entre el algoritmo de iteración de valores y Q-learning:

- Q-learning no requiere del modelo completo del MDM a diferencia de iteración de valores.
- En iteración de valores, durante el proceso de una actualización, se genera una nueva aproximación para todos los estados del sistema. En cambio, para el caso

de una iteración de Q-learning, únicamente se genera una nueva aproximación para el elemento del espacio de estados-acciones correspondiente a esa iteración.

El último punto es debido a la naturaleza estocástica del algoritmo. El aprendizaje de la Q-función óptima sucede conforme van ocurriendo las parejas de estados-acciones en las observaciones que se hacen al sistema. Como es comentado en la referencia [33, página 281], en las suposiciones respecto a la convergencia del Algoritmo 3.4.1, viene implícito la necesidad de que todos los pares de estados-acciones sean visitados un numero infinito de veces.

Q-learning pertenece a la familia de algoritmos tipo «*diferencia temporal*» (*temporal difference learning* -TD-). El proceso de actualización de la ecuación (3.4.1) puede reescribirse como

$$Q^{n+1}(x,a) = Q^n(x,a) + \alpha_n(x,a)\delta_n(x,a);$$

siendo $\delta^n(x,a) = r(x,a) + \gamma W_n(s) - Q^n(x,a)$ el error (TD error) entre la aproximación $Q^n(x,a)$ y la nueva estimación $r(x,a) + \gamma W_n(s)$. La similitud entre los algoritmos tipo TD y los de PD, es que utilizan estimaciones pasadas para las siguientes aproximaciones a realizar; con diferencia que los algoritmos tipo TD realizan un bootstrap a las estimaciones ya conocidas [30]. Otros algoritmos de esta familia se distinguen principalmente por su TD error [30]:

- TD(0):

$$\delta^n(x) = r(s, a_{n+1}) + \gamma v^n(s) - v^n(x);$$

- SARSA:

$$\delta^n(x,a) = r(x,a) + \gamma Q^n(s, a_{n+1}) - Q^n(x,a);$$

- Double Q-learning [13]:

$$\delta_A^n(x,a) = r(x,a) + \gamma Q_B^n(s,a') - Q_A^n(x,a),$$
$$\delta_B^n(x,a) = r(x,a) + \gamma Q_A^n(s,b') - Q_B^n(x,a);$$

donde $a' = \arg\max_{a\in A(s)} Q_A^n(s,a)$, $b' = \arg\max_{a\in A(s)} Q_B^n(s,a)$; y δ_A^n ó δ_B^n son escogidos aleatoriamente.

La naturaleza estocástica de esta familia de algoritmos solventan «la maldición del modelo» que PD presenta.

3.6.1. Dimensión del MDM

Q-learning es un método tabular. Sin embargo, el hecho de que en cada iteración se actualice únicamente un valor de una pareja de estado-acción, permite que en la práctica sea necesario reservar memoria para solo una Q-función. Aún así, para conjuntos de estados y acciones muy grandes, tal cosa puede llegar a ser imposible.

Por otro lado, cuando la dimensión es demasiado grande, el proceso de aprendizaje es muy lento. Esto debido a que deben pasar muchas iteraciones para que el algoritmo haya recorrido todas las parejas de estados-acciones las suficientes veces como para tener una aproximación adecuada.

Entonces, Q-learning y el resto de algoritmos basados en la aproximación estocástica, siguen presentando la maldición de dimensionalidad.

3.7. Ejemplos

Los siguientes son ejemplos de sistemas cuyas probabilidades de transición son difíciles de conocer y por lo tanto el algoritmo de Q-learning se presta para resolverlos. Tales ejemplos son simuladores de un taxi y del juego de blackjack; los cuales están implementados en la librería Gymnasium[2]. Dicha librería contiene implementaciones de diversos sistemas aptos para probar algoritmos de AR; la ventaja que se obtiene es la consistencia que se tiene para acceder a los elementos correspondientes de los MDM, así como para realizar evoluciones de los sistemas; por lo que se vuelve muy fácil reutilizar y adaptar códigos ya realizados.

Debido a la naturaleza episódica de los sistemas, se utilizaron las adaptaciones al algoritmo de Q-learning mencionadas en la subsección 3.4.1. Para ambos sistemas se realizaron simulaciones siguiendo las políticas obtenidas con las aproximaciones realizadas. Los resultados obtenidos fueron los cálculos de la recompensa promedio y la tasa de éxito del sistema, los cuales dan una idea del rendimiento de la aproximación realizada.

[2] `https://gymnasium.farama.org`

3.7.1. Ejemplo: Taxi

El ejemplo del taxi es un sistema de rejilla con la tarea extra de recoger a un pasajero y llevarlo a su destino. La implementación que se encuentra en la librería Gymnasium[3] consiste en una rejilla de tamaño 5×5; la posición inicial del taxi es en un punto aleatorio de la rejilla; mientras que el pasajero inicia aleatoriamente en una de las casillas roja, verde, amarilla, o azul; con alguna de las casillas antes mencionadas como destino (ver Figura 3.7.3a). Nos referimos a un episodio del sistema desde que comienza en el estado inicial hasta que el sistema termina, lo cual ocurre cuando el pasajero es llevado correctamente a su destino, o bien, pasan doscientas épocas de decisión. Intentamos dar su descripción como un MDM de la siguiente manera:

- Aparte de las veinticinco casillas accesibles para el taxi, se debe de tomar en cuenta la ubicación del pasajero y de su destino. Se asignan los valores de 0 a la casilla roja, 1 a la verde, 2 amarilla, 3 azul, y 4 a si el pasajero se encuentra en el taxi. Por otro lado, si (i, j) es la posición del taxi en la rejilla, entonces, un elemento x del espacio de estados es codificado de la siguiente manera:

$$x = ((i * 5 + j) * 5 + \text{ubicación del pasajero}) * 4 + \text{destino del pasajero};$$

  entonces $X = \{0, 1, 2, 3, \ldots, 497, 498, 499\}$; y $|X| = 500$, pues hay $5 \times 5 \times 5 \times 4$ posibles estados.

- Al igual que en la sección 2.8.1, el conjunto de acciones corresponde a los cuatro movimientos que el taxi puede realizar en la rejilla, con el extra de que ahora hay acciones correspondientes para recoger y bajar al pasajero:

$$A = \{\downarrow, \uparrow, \rightarrow, \leftarrow, \text{“Recoger pasajero”}, \text{“Bajar pasajero”}\}.$$

- Respecto a los conjuntos de acciones admisibles, estos coinciden con el de acciones: $A(x) = A$, $\forall x \in X$. De esta forma $|\mathbb{K}| = 3000$. Realizar acciones que no tienen sentido para algún estado del sistema se verá reflejado en la función recompensa.

[3] `https://gymnasium.farama.org/environments/toy_text/taxi/`

- Las transiciones de estado son deterministas, es decir, la acción ↑ desplaza al taxi una casilla arriba. Realizar acciones inadmisibles, como mover el taxi fuera de la rejilla, simplemente no cambian el estado del sistema.

- La función de recompensa retorna lo siguiente: -1 por cada acción realizada en cada época de decisión; a menos que el pasajero sea llevado a su destino, para lo cual proporciona el valor de 20; finalmente realizar la acción de recoger o bajar pasajero donde no corresponde, devuelve -10.

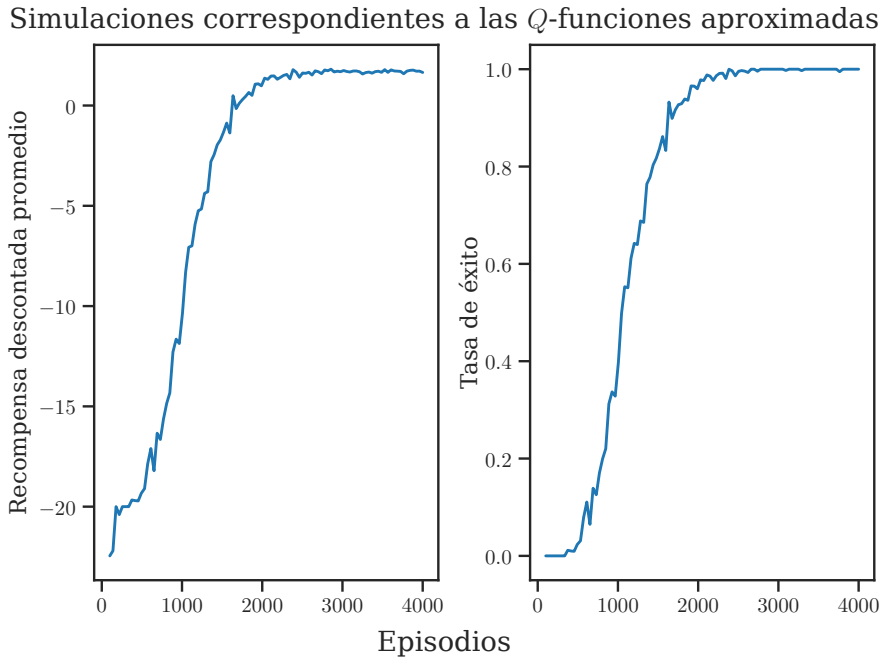


Figura 3.7.1. Para el proceso de aprendizaje se fijaron los valores de $\gamma = 0.95$, $\varepsilon = 0.60$ (para la búsqueda ε-codiciosa); y como tasa de aprendizaje $\alpha_n = \log(n)/n$.

Se midió el rendimiento de Q-learning en función del número de episodios que se hayan utilizado para el entrenamiento. Con cada Q-función y (regla de decisión derivada) a probar, se simuló el sistema cinco mil veces, a fin de obtener la recompensa descontada promedio y la tasa de éxito (si el pasajero fue llevado a su destino correctamente). Como se aprecia en la Figura 3.7.1, a partir de dos mil episodios, la recompensa promedio se estabiliza y las aproximaciones superan el 95 % de tasa de éxito.

Se realizó una aproximación a la Q-función con dos mil episodios. Tal aproximación la podemos observar en la Figura 3.7.2. Realizando cinco mil simulaciones con la regla

de decisión derivada; ésta obtiene 0.975 unidades de recompensa descontada promedio, así como una tasa de éxito de 95.7 %. Sin embargo, por la naturaleza estocástica del algoritmo, estos valores pueden cambiar si se vuelve a aproximar otra Q-función. Una realización del sistema siguiendo la regla de decisión obtenida aprecia en la Figura 3.7.3.

Describimos brevemente el código utilizado: como siempre, se inicia cargando las librerías necesarias

```
import numpy as np
import gymnasium as gym
from gymnasium.wrappers import TimeLimit
```

después, con fin de poder entrenar varios sistemas con configuraciones distintas, es útil crear una clase que contenga lo necesario para entrenar cada uno de ellos:

```
class TaxiEnv:
        def __init__(self, episodes):
                self.episodes = episodes # k episodios para
                    entrenar
                self.rng = np.random.default_rng()
                self.env = gym.make("Taxi-v3", render_mode="
                    rgb_array").env #sistema de taxi
                self.env = TimeLimit(self.env, max_episode_steps
                    =200) #timpo maximo de iteraciones
                self.states_n = self.env.observation_space.n
                self.actions_n = self.env.action_space.n
                self.epsilon = 0.60 # para la busqueda codiciosa
                self.gamma = 0.95 #gamma
                self.q_func = np.zeros((self.states_n, self.
                    actions_n)) #iniciar Q-funcion como cero

        def _eg_action(self, x): #seleccionar accion epislon-
            codiciosa
                y = self.rng.random()
                if y >= self.epsilon:
                        a = self.env.action_space.sample() #
                            exploracion
                else:
```

```
                    a = self.q_func[x].argmax() #codiciosa
            return a

    def q_learn(self): #procedimiento de Q-learning
            for k in range(self.episodes): #los k episodes
            n = 0
            xn, info = self.env.reset() #inicializa el
                sistema
            episode_finished = False
            episode_truncation = False
            while episode_finished == False:
                    if episode_truncation == True:
                            break #detenerse si se alcanza
                                maximo de iteraciones
                    else:
                            an = self._eg_action(xn)
                            s, reward, episode_finished,
                                episode_truncation, info =
                                self.env.step(an) #aplica
                                accion y observa transicion de
                                 sistema
                            n += 1
                            #calculo y actualizacion de
                                valores
                            alphan = (np.log(n))/(n) #
                                alpha_n(xn,an)
                            Qn = self.q_func[xn,an] # Qn(xn,
                                an)
                            Wn = self.q_func[s].max() # Wn(s)
                            Qit = (1 - alphan)*Qn + alphan*(
                                reward + self.gamma*Wn) #Q_n+1
                            self.q_func[xn,an] = Qit
                            xn = s

    def max_qfunc(self):
```

```
                return self.q_func.max(axis = 1)


        def d_rule(self):
                return self.q_func.argmax(axis = 1)
```

con la clase definida, solamente es cuestión de crear los objetos

```
taxi_env = TaxiEnv(2000) #crear objeto
taxi_env.q_learn() #entrenarlo
#guardar resultados
q_func = taxi_env.q_func
d_rule = taxi_env.d_rule()
```

Es fácil ver como la librería Gymnasium facilita poder reutilizar código en otros sistemas u otro tipo de algoritmo.

3.7.2. Ejemplo: Blackjack

El siguiente ejemplo es planteado en la referencia [30, capítulo 5, página 93], y es analizado mediante métodos tipo Monte Carlo; sin embargo, aquí es resuelto mediante Q-learning. El objetivo del blackjack es obtener cartas cuya suma se acerque los más posible sin sobrepasar el valor de 21. El jugador compite contra el dealer; al inicio del juego ambos reciben dos cartas, una carta del dealer se mantiene boca abajo, mientras que las dos del jugador son visibles; en base a lo que el jugador observa, escoge pedir más cartas al dealer (hits), o bien mantenerse con las que ya tiene (sticks). Si el jugador pide más cartas y su suma sobrepasa 21, pierde y el juego termina; cuando decide mantenerse, finaliza su turno y comienza el del dealer, el cual debe sacar cartas hasta que su suma sea de 17 ó más; si la suma del dealer excede 21, la victoria es para el jugador, en caso contrario, la victoria (o empate) depende de cual suma es más cercana a 21. Los valores de las cartas son los siguientes: los J, Q y K valen 10; el as, su valor es de 1 ó 11 a conveniencia del jugador (cuando el valor es de 11 sin que haga perder la partida al jugador, se dice que es un as usable); para el resto de cartas su valor corresponde a su valor numérico. Puede suceder que al inicio del juego, el jugador obtenga 21 inmediatamente, en tal caso, es victoria para el, a menos que el dealer también haya obtenido inmediatamente 21 (empate).

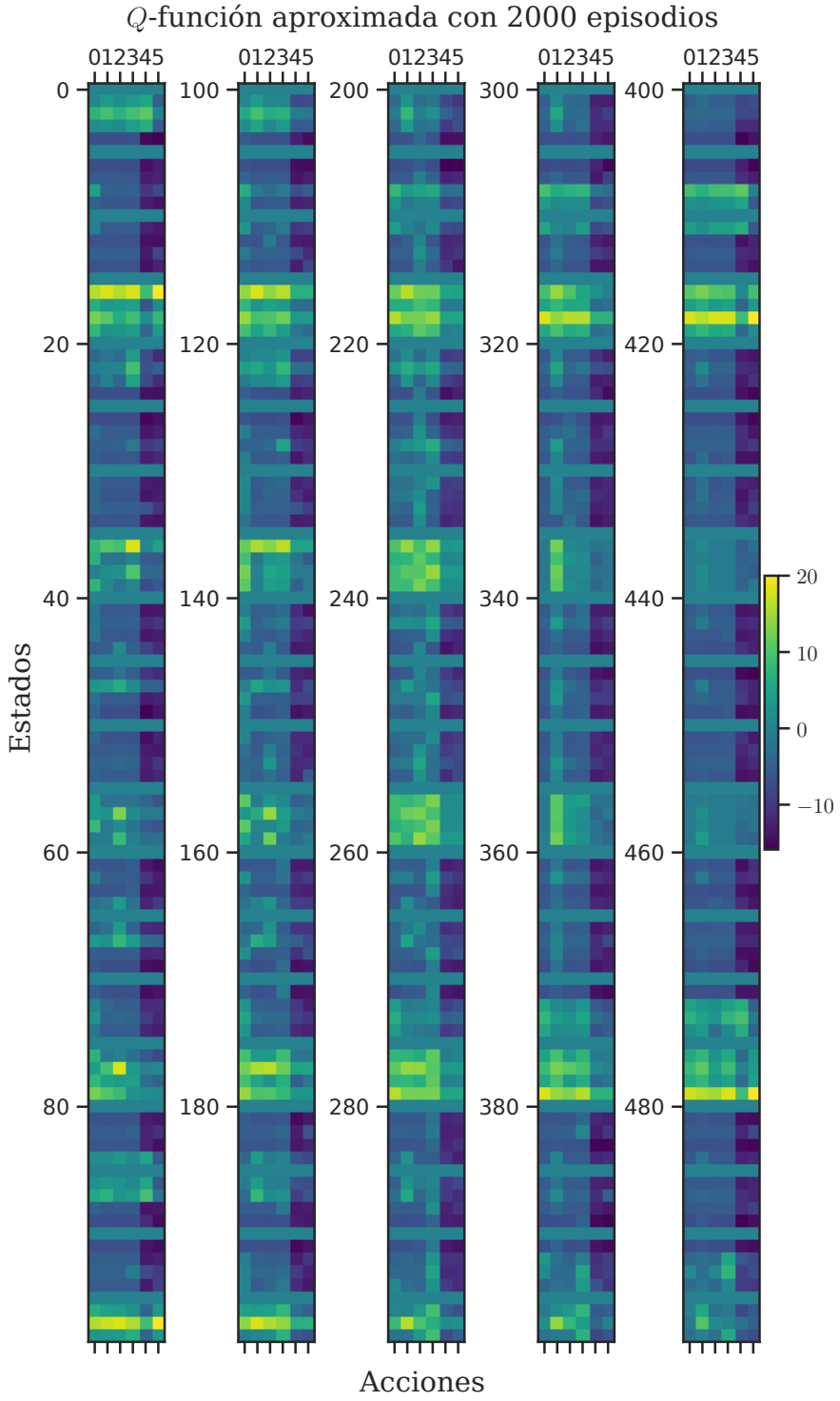
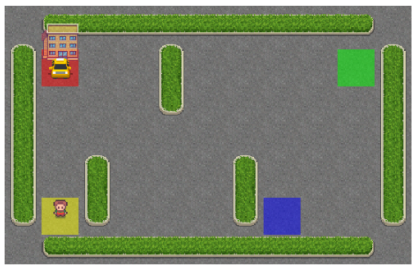
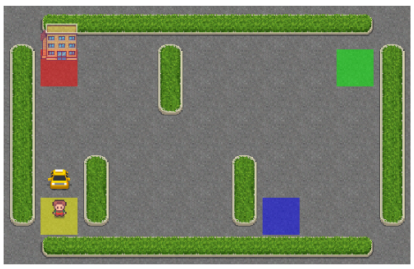


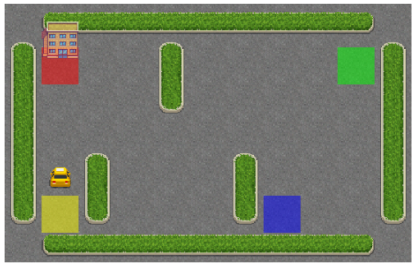
Figura 3.7.2. El orden de las acciones coincide con la dada en la descripción como MDM.



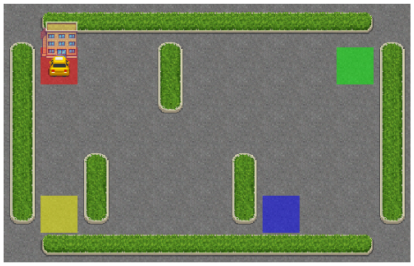
(a) Una posible configuración inicial del sistema.



(b)



(c)



(d) La duración del trayecto fue de diez épocas de decisión.

Figura 3.7.3. En el repositorio donde se encuentran los códigos, se puede apreciar una animación similar a esta en formato de video.

A continuación damos su descripción como un MDM:

- El jugador en todo momento tiene conocimiento de lo siguiente: la suma de sus cartas, la carta visible del dealer, y si el jugador cuenta con algún as. Entonces representamos un elemento x del espacio de estados como la tripleta

$$x = (\text{“suma del jugador”}, \text{“carta dealer”}, \text{“as usable”});$$

  con “suma del jugador” tomando los valores $\{4, 5, 6, \ldots, 19, 20, 21\}$. “As usable” los valores {0,1}; donde 1 representa si se cuenta con el as usable y 0 el caso contrario.

- El espacio de acciones es $A = \{0, 1\}$; donde 0 corresponde a stick y 1 a hit.

- En cualquier estado del sistema es posible realizar ambas acciones, por lo que $A(x) = A, \forall x \in X$.

- En esta versión del juego consideramos que el dealer cuenta con un mazo de cartas infinito, es decir, siempre cuenta con un mazo de cartas completo. Por lo que en cada acción hit, el dealer reparte una carta con distribución Uniforme(52), luego el sistema se actualiza a las correspondientes sumas.

- La función de recompensa devuelve el valor de 1 en caso de victoria del jugador, -1 en caso de derrota y 0 de cualquier otra forma. Dado que la recompensa se obtiene al final del juego, se utiliza factor de descuento $\gamma = 1$, pues no tiene sentido penalizar la recompensa.

La librería Gymnasium[4] tiene implementado este sistema. A fin de intentarlo resolver con Q-learning, se calcularon las aproximaciones mediante Q-learning en un rango de cien a quinientos mil episodios. Con cada aproximación obtenida, se simularon diez mil juegos de blackjack siguiendo las reglas de decisión derivadas, a fin de calcular sus tasas de éxito. Una primera exploración fue probar el rendimientos de las aproximaciones con las tasas de aprendizaje (3.4.3), y distintos valores de epsilon: $\varepsilon = 0.35$, $\varepsilon = 0.50$, $\varepsilon = 0.65$. Los resultados se observan en la primera fila de la Figura 3.7.4; lo primero que

[4] `https://gymnasium.farama.org/environments/toy_text/blackjack/`

se observa es que a partir de cien mil episodios, ya no hay mejora en la tasa de éxito; lo segundo en observarse es que el valor de ε no mejora la tasa en ninguno de los dos casos; sin embargo, la función $\alpha_n = \log(n)/n$, obtiene en general mayor rendimiento.

Para tratar de obtener una mayor tasa, se intentó con sucesiones de la probabilidad de exploración (ε_n): $\varepsilon_n = 1 - (1/n)$, $\varepsilon_n = 1/n$, $\varepsilon_n = 1/(1+n)$. Los resultados corresponden a la segunda fila de la Figura 3.7.4; donde se aprecia una ligera mejora en la tasa de éxito con $\varepsilon_n = 1/n$.

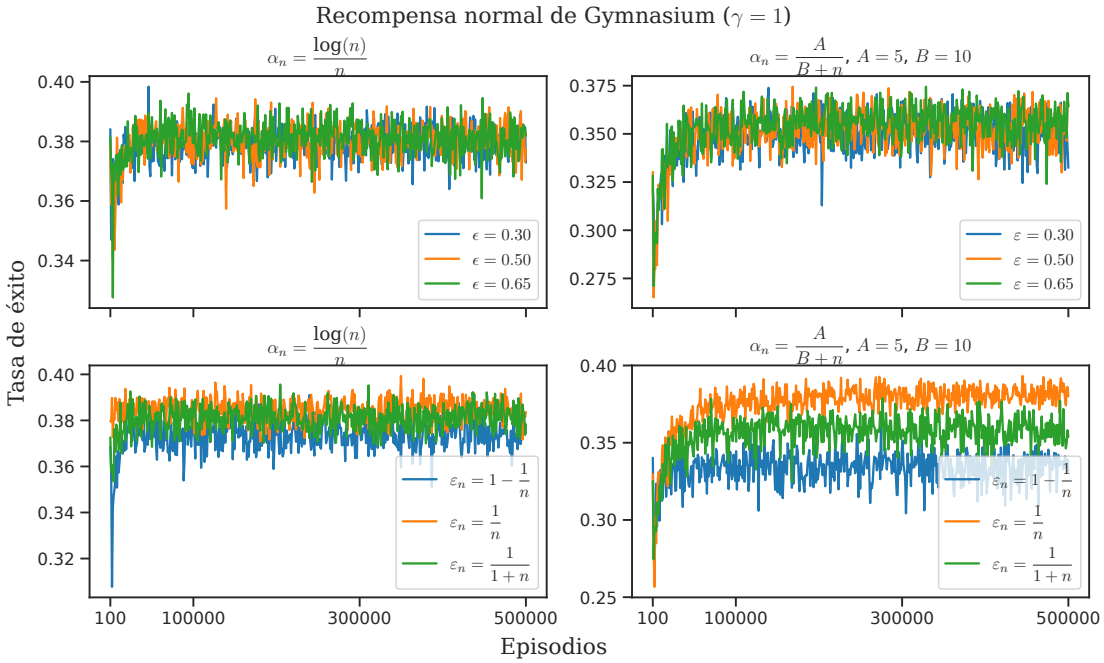


Figura 3.7.4

A fin de intentar mejorar el rendimiento de 38 %, se modificó la función recompensa, asignando el valor de 0.5 a las acciones hit que no terminen el juego. Esto con el fin de motivar al algoritmo a jugar un poco más arriesgado. Los resultados se aprecian en la Figura 3.7.5. Aun así, se obtienen resultados similares a los anteriores con la recompensa sin modificar.

La Figura 3.7.6 es un ejemplo de las políticas de decisión que el algoritmo aprende. Para la mayor parte de estados decide realizar acciones stick, sin ningún patrón aparente para las acciones hit.

Simulando diez mil juegos, donde se realizaban las acciones stick y hit de manera aleatoria, se obtuvo que la tasa de éxito fue 28.15 %; mientras que realizando acciones

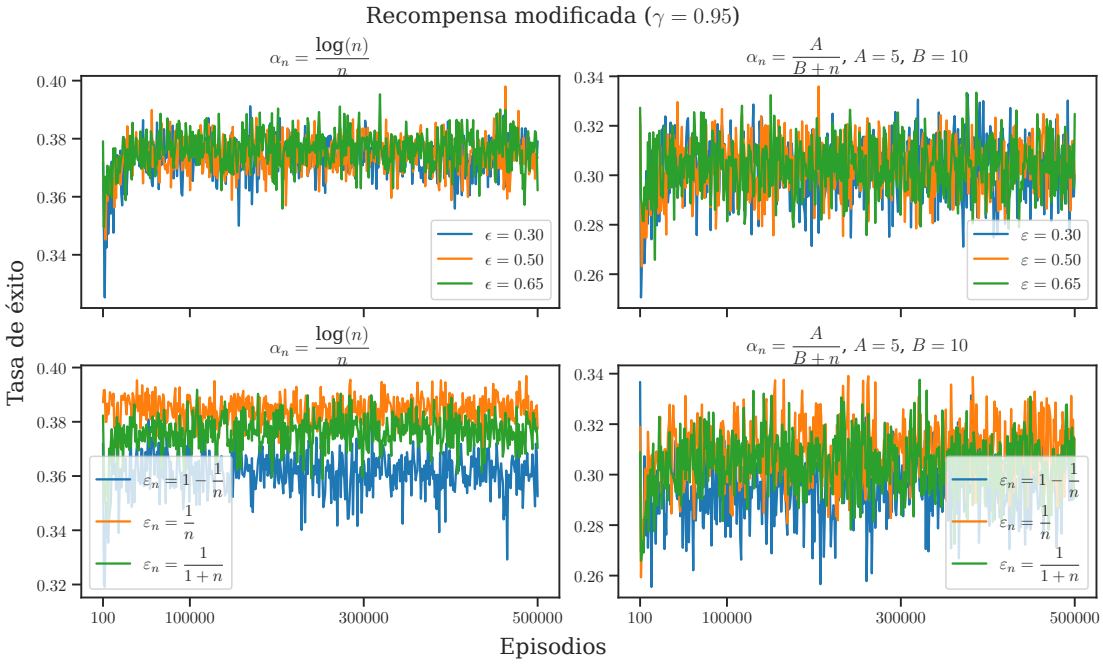


Figura 3.7.5. Dado que ahora hay recompensas durante el transcurso del juego, se utiliza el factor de descuento.

stick a partir de una suma de dieciocho y hit para sumas menores, la tasa es de 40.31 %; por otro lado, siempre realizar acciones stick resulta en 38.36 % de tasa de éxito. El gran número de posibles resultados que tiene el juego, le dificulta al algoritmo descubrir algún patrón de juego.

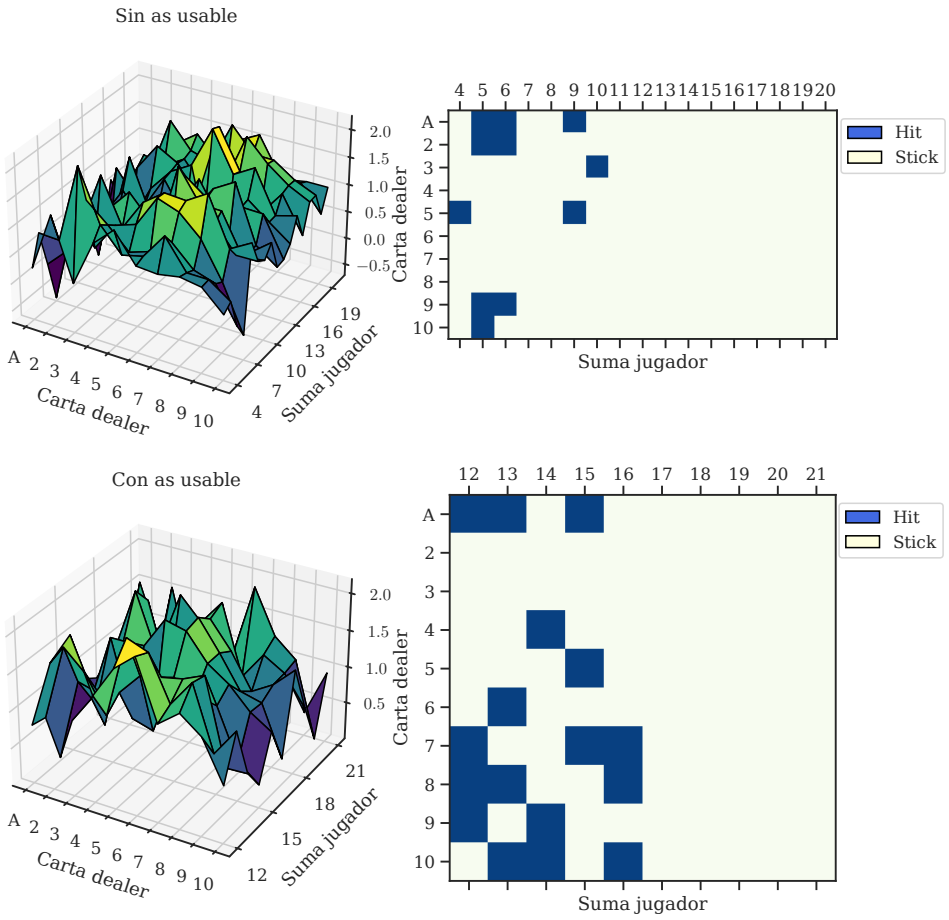


Figura 3.7.6. Los gráficos de la izquierda corresponden a $\max_{a \in A(x)} Q(x, a)$; y los de la derecha a la regla de decisión. La tasa de éxito lograda por esta aproximación fue de 37.41 %.

Resumen y Conclusiones

Se abordó la teoría de los Procesos de Decisión de Markov, junto con el problema del control óptimo en el caso descontado. Este criterio de recompensa y la teoría de los espacios lineales, juega un papel importante en la Programación Dinámica.

Otro concepto relevante son las reglas de decisión llamadas conservadoras y avariciosas. En primer lugar se demostró que solo es necesario el estudio de reglas markovianas, para después, probar que las políticas estacionarias que siguen las reglas conservadoras, sus funciones de valor coinciden con las función de valor óptimo, y por lo tanto, resuelven el problema de control óptimo. Sin restar importancia, los resultados que aseguran la existencia de políticas conservadoras (y por lo tanto de reglas de decisión óptimas), son relevantes para delimitar los sistemas que pueden ser resueltos. Se presentó el algoritmo de Iteración de Valores, como el método para encontrar aproximaciones a reglas de decisión óptimas que sirvan como solución a tales sistemas.

Bajo esos conceptos fue planteado en términos matemáticos el paradigma de Aprendizaje por Refuerzo. Sin embargo, se presentan dos problemas comunes en la mayor parte de sistemas. Estos dos problemas son la «maldición del modelo y dimensionalidad»; ambos presentes en la mayoría de problemas que surgen en la vida real.

Tanto el criterio descontando como la Programación Dinámica continuaron siendo elementos importantes en la teoría de las Q-funciones. Así mismo, la relación entre las reglas de decisión codiciosas y conservadoras sirven como puente para la existencia de políticas de decisión óptimas. La teoría de la aproximación estocástica es una primera solución a la «maldición del modelo». Mediante el algoritmo de Q-learning, se pueden obtener aproximaciones a la Q-función óptima realizando simulaciones del sistema, y de donde se construye la regla de decisión que resuelve el sistema. Sin embargo, la «maldición de dimensionalidad» sigue estando presente.

Finalmente, los ejemplos desarrollados, cumplen el objetivo de representar los conceptos expuestos. Así como los códigos realizados enseñan a implementar estos algoritmos.

Continuación y Trabajos Futuros

Los siguientes son temas que pueden seguirse estudiando e investigando a modo de continuación de este trabajo.

Tasas de Convergencia

Sea una sucesión (u^n) con límite u^* en un espacio $\mathcal{U}$. Se dice que (u^n) converge en orden al menos α $(\alpha > 0)$, si existe una constante $K > 0$ tal que

$$\left\|u^{n+1} - u^*\right\| \leq K\left\|u^{n+1} - u^*\right\|^{\alpha};$$

para $n = 1, 2, 3, \ldots$ Mientras más pequeña sea la constante K, más rápido converge (u^n) a u^*.

Puterman [26, capítulo 6, página 159] desarrolla estos conceptos; mientras que [26, Teorema 6.3.3], provee resultados respecto a la tasa de convergencia de iteración de valores.

Un complemento interesante a este trabajo sería respecto a la tasa de convergencia del algoritmo Q-learning.

Restricción a Políticas de Decisión Clase MR

En la sección 1.5 se demostró que el problema de control óptimo se puede restringir a la búsqueda de políticas Markovianas. Más adelante, en la sección 3.1, el problema de control óptimo fue presentado en términos de las Q-funciones, por lo que se intentó proponer un resultado análogo al Teorema 1.5.2:

Corolario. *Supongamos* $\pi \in \Pi^{HR}$. *Entonces para cada* $(x, a) \in \mathbb{K}$ *existe* $\pi' \in \Pi^{MR}$ *(la*

cual depende de (x, a)*) tal que*

$$Q^{\pi}(x, a) = Q^{\pi'}(x, a).$$

Sin embargo, no fue posible establecer una demostración satisfactoria. De ser verdadero este resultado, sería un buen complemento para la teoría de las Q-funciones.

Q-learning y error en las Políticas de Decisión Aproximadas

Como se mencionó en la sección 3.5, conociendo la Q-función de valor óptimo, es posible conocer el nivel de optimalidad de la política de decisión obtenida mediante el algoritmo de Q-learning. Pero un resultado más interesante sería como al que se tiene en iteración de valores: conocer el nivel de optimalidad de la política en base a la cercanía de las dos últimas iteraciones de Q-learning; o bien, al menos que el nivel de optimalidad sea en términos del número de iteraciones realizadas.

Aproximación de Funciones

Es posible aproximar una Q-función mediante un mapeo $F : \mathbb{R}^n \to \mathcal{Q}$, de forma que un parámetro $\theta \in \mathbb{R}^n$ representa

$$q(x, a) = [F(\theta)](x, a), \quad \forall (x, a) \in \mathbb{K}.$$

De esta forma, solamente es necesario guardar n parámetros, en vez de un valor para todos los elemento de $\mathbb{K}$. Este esquema de aproximación permite resolver problemas de AR donde los espacios de estados y acciones son de cardinalidades muy grandes, o incluso continuos.

El uso de aproximaciones paramétricas lineales, lleva a una versión modificada del algoritmo Q-iteration, el cual utiliza regresión por mínimos cuadrados como método para estimar los parámetros[6]. Otros algoritmos son basados en técnicas de descenso por gradiente [30]. En general, la aproximación puede ser de tipo no lineal, para lo que típicamente se utilizan redes neuronales como función aproximadora [4]. En años más recientes, con el uso de redes neuronales profundas como aproximadores, fueron

desarrollados los algoritmos *deep Q-learning* [24] y *double deep Q-learning* [14]. Algunos resultados teóricos respecto a convergencias se han realizado [10, 31], sin embargo, sigue siendo una línea de investigación activa.

Apéndice A

Resultados Auxiliares

A continuación se presentan los dos resultados principales utilizados para probar la convergencia de los Algoritmos 2.5.1 y 3.4.1.

A.1. Teorema de Punto Fijo de Banach

La teoría respecto a espacios lineales normados es de relevancia para el análisis de algoritmos de PD. Los siguientes conceptos son abordados por Puterman [26, apéndice C], cuyos resultados son empleados en algunos teoremas (ej. Teorema 2.1.3).

Sea $(\mathcal{U}, \|\cdot\|)$ un espacio de Banach.

Definición A.1.1. Un operador $T : \mathcal{U} \to \mathcal{U}$ se dice que es una *contracción* en $\mathcal{U}$ si existe $\lambda \in [0, 1)$ tal que para todo $w, u \in \mathcal{U}$:

$$\|Tw - Tu\| \leq \lambda \|w - u\|.$$

Teorema A.1.1. *Sea $T : \mathcal{U} \to \mathcal{U}$ una contracción en $\mathcal{U}$. Entonces*

a) existe un único $u^* \in \mathcal{U}$ tal que $Tu^* = u^*$; y

b) para un $u^0 \in \mathcal{U}$ arbitrario, la sucesión (u^n) definida como:

$$u^{n+1} = Tu^n = T^{n+1}u^0; \tag{A.1.1}$$

converge a u^*.

Demostración. Primero se prueba que la sucesión definida en (A.1.1) es de Cauchy lo cual asegura la existencia del límite, posteriormente se prueba su unicidad.

Sea (u^n) definida como en (A.1.1). Para un $m \geq 1$, se tiene por la desigualdad triangular que

$$\begin{aligned}\left\|u^{n+m} - u^n\right\| &\leq \sum_{k=0}^{m-1} \left\|u^{n+k+1} - u^{n+k}\right\| \\ &= \sum_{k=0}^{m-1} \left\|T^{n+k}u^1 - T^{n+k}u^0\right\| \\ &\leq \sum_{k=0}^{m-1} \lambda^{n+k}\left\|u^1 - u^0\right\| \\ &= \frac{\lambda^n(1-\lambda^m)}{(1-\lambda)}\left\|u^1 - u^0\right\|;\end{aligned}$$

dado que $\lambda \in [0, 1)$, entonces el lado derecha de la desigualdad se puede hacer arbitrariamente pequeño para un n lo suficientemente grande. Esto demuestra que la sucesión (u^n) es de Cauchy y por la completes de $\mathcal{U}$, esta tiene límite $u^* \in \mathcal{U}$. Ahora,

$$\begin{aligned}0 \leq \|Tu^* - u^*\| &\leq \|Tu^* - u^n\| + \|u^n - u^*\| \\ &= \left\|Tu^* - Tu^{n-1}\right\| + \|u^n - u^*\| \\ &\leq \lambda\left\|u^* - u^{n-1}\right\| + \|u^n - u^*\|;\end{aligned}$$

tomando el límite cuando $n \to \infty$ se tiene que $\|Tu^* - u^*\| = 0$, por lo tanto debe de ser que $Tu^* = u^*$. Para demostrar la unicidad, supongamos que $w^* \in \mathcal{U}$ es otro punto fijo de T, luego

$$\|Tw^* - Tu^*\| = \|w^* - u^*\| \leq \lambda\|w^* - u^*\|;$$

lo cual es una contradicción al hecho de que T es una contracción. Por lo tanto u^* es el único punto fijo de T. □

A.2. Aproximación Estocástica

El aumento en poder de computo ha inspirado a resolver problemas de optimización mediante procesos de simulación. El siguiente resultado es encontrado en la literatura como algoritmo tipo Robbins–Monro. Su implicación es utilizada para las pruebas de los algoritmos tipo TD mencionados en la sección 3.6.

Teorema A.2.1. *El proceso estocástico actualizado de forma iterativa como*

$$\Delta_{n+1}(x) = (1 - \alpha_n(x))\Delta_n(x) + \beta_n(x)F_n(x); \tag{A.2.1}$$

converge a cero c.s. bajo las siguientes suposiciones:

1. *El espacio de estados es finito.*
2. $\sum_n \alpha_n(x) = \infty$; $\sum_n \alpha_n^2(x) < \infty$; $\sum_n \beta_n(x) = \infty$; $\sum_n \beta_n^2(x) < \infty$; *y*

$$\mathbb{E}(\beta_n(x) \,|\, \mathcal{F}_n) \leq \mathbb{E}(\alpha_n(x) \,|\, \mathcal{F}_n), \quad \textit{uniformemente c.s.}$$

3. $\|\mathbb{E}(F_n(x) \,|\, \mathcal{F}_n)\|_W \leq \gamma \|\Delta_n\|_W$, *con* $\gamma \in (0, 1)$.
4. $\mathrm{Var}(F_n(x) \,|\, \mathcal{F}_n) \leq C(1 + \|\Delta_n\|_W)^2$, *con* C *alguna constante.*

Siendo $\mathcal{F}_n = \{\Delta_n, \Delta_{n-1}, \ldots, F_{n-1}, \ldots, \alpha_{n-1}, \ldots, \beta_{n-1}, \ldots\}$ *el pasado del proceso hasta el tiempo* n, *y* $\|\cdot\|_W$ *es alguna norma del supremo ponderada.*

Demostración. Ver referencia [16]. □

Más bibliografía referente a la aproximación estocástica son las referencias [1, 6, 11, 17].

Referencias

[1] A. L. Almudevar, *Approximate Iterative Algorithms*, 1.ª edición. CRC Press, 2014, ISBN: 9780203503416.

[2] R. B. Ash y C. A. Doleans-Dade, *Probability and Measure Theory*, 2.ª edición. Academic Press, 1999, ISBN: 978-0120652020.

[3] R. E. Bellman, *Dynamic Programming*, 1.ª edición. Princeton University Press, 1957, ISBN: 0-691-07951-X.

[4] D. P. Bertsekas y J. N. Tsitsiklis, *Neuro-Dynamic Programming* (Optimization and Neural Computation Series), 1.ª edición. Athena Scientific, 1996, ISBN: 1-886529-10-8.

[5] D. Bouneffouf e I. Rish. "A Survey on Practical Applications of Multi-Armed and Contextual Bandits." arXiv: `1904.10040`. (2019).

[6] L. Busoniu, R. Babuska, B. D. Schutter y D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators* (Automation and Control Engineering Series), 1.ª edición. CRC Press, 2010, ISBN: 9781439821091.

[7] J. Chakravorty y A. Mahajan, "Multi-Armed Bandits, Gittins Index, and its Calculation," *Methods and Applications of Statistics in Clinical Trials*, páginas 416-435, 2014.

[8] J. Edwards. "Practical Calculation of Gittins Indices for Multi-armed Bandits." arXiv: `1909.05075`. (2019).

[9] A. N. Elmachtoub, R. McNellis, S. Oh y M. Petrik. "A Practical Method for Solving Contextual Bandit Problems Using Decision Trees." arXiv: 1706.04687. (2017).

[10] J. Fan, Z. Wang, Y. Xie y Z. Yang. "A Theoretical Analysis of Deep Q-Learning." arXiv: 1901.00137. (2019).

[11] A. Gosavi, *Simulation-Based Optimization* (Operations Research/Computer Science Interfaces Series), 2.ª edición. Springer US, 2015, volumen 55, ISBN: 978-1-4899-7490-7.

[12] R. Gross, *Psychology The Science of Mind and Behaviour*, 6.ª edición. Hodder Education, 2010, ISBN: 9781444108316.

[13] H. Hasselt, "Double Q-learning," *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel y A. Culotta, edición, páginas 2613-2621, 2010.

[14] H. V. Hasselt, A. Guez y D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, volumen 30, número 1, 2016, ISSN: 2374-3468.

[15] T. Jaakkola, M. I. Jordan y S. P. Singh, "Convergence of stochastic iterative dynamic programming algorithms," *Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS 1993)*, páginas 703-710, 1993.

[16] T. Jaakkola, M. I. Jordan y S. P. Singh, "On the Convergence of Stochastic Iterative Dynamic Programming Algorithms," *Neural Computation*, volumen 6, número 6, páginas 1185-1201, 1994, ISSN: 0899-7667.

[17] H. J. Kushner y D. S. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, 1.ª edición. Springer New York, 1978, volumen 26, ISBN: 978-0-387-90341-5.

[18] J. Langford y T. Zhang, "The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information," *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, J. Platt, D. Koller, Y. Singer y S. Roweis, edición, páginas 817-824, 2007.

[19] T. Lattimore y C. Szepesvári, *Bandit Algorithms*, 1.ª edición. Cambridge University Press, 2020, ISBN: 9781108571401.

[20] L. Li, W. Chu, J. Langford y R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," *Proceedings of the 19th international conference on World wide web*, páginas 661-670, 2010.

[21] T. Lu, D. Pal y M. Pal, "Contextual Multi-Armed Bandits," *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh y M. Titterington, edición, páginas 485-492, 2010.

[22] S. Mahadevan, "Average reward reinforcement learning: Foundations, algorithms, and empirical results," *Machine Learning*, volumen 22, número 1-3, páginas 159-195, 1996, ISSN: 0885-6125.

[23] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran y R. Hadsell. "Learning to Navigate in Complex Environments." arXiv: `1611.03673`. (2016).

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg y D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, volumen 518, número 7540, páginas 529-533, 2015, ISSN: 0028-0836.

[25] A. I. Panov, K. S. Yakovlev y R. Suvorov, "Grid Path Planning with Deep Reinforcement Learning: Preliminary Results," *Procedia Computer Science*, volumen 123, páginas 347-353, 2018, ISSN: 18770509.

[26] M. L. Puterman, *Markov Decision Processes*, 1.ª edición. Wiley, 1994, ISBN: 0-471-72782-2.

[27] A. Rafiq, T. A. A. Kadir y S. N. Ihsan, "Pathfinding Algorithms in Game Development," *IOP Conference Series: Materials Science and Engineering*, volumen 769, número 1, página 012 021, 2020, ISSN: 1757-8981.

[28] S. P. Singh y R. C. Yee, "An Upper Bound on the Loss from Approximate Optimal-Value Functions," *Machine Learning*, volumen 16, número 3, páginas 227-233, 1994, ISSN: 08856125.

[29] Y. SunWoo y W. Lee, "Comparison of deep reinforcement learning algorithms: Path Search in Grid World," *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, páginas 1-3, 2021.

[30] R. S. Sutton y A. G. Barto, *Reinforcement learning: an introduction* (Adaptive Computation and Machine Learning Series), 2.ª edición. The MIT Press, 2018, ISBN: 9780262039246.

[31] Z. T. Wang y M. Ueda. "Convergent and Efficient Deep Q Network Algorithm." arXiv: `2106.15419`. (2021).

[32] C. J. C. H. Watkins, "Learning from Delayed Reward," Tesis doctoral, Cambridge University, 1989.

[33] C. J. C. H. Watkins y P. Dayan, "Q-learning," *Machine Learning*, volumen 8, número 3-4, páginas 279-292, 1992, ISSN: 0885-6125.

[34] D. J. White, *Markov Decision Processes*, 1.ª edición. Wiley, 1993, ISBN: 0-471-93627-8.

[35] M. Wiering y M. van Otterlo, edición, *Reinforcement Learning* (Adaptation, Learning, and Optimization), 1.ª edición. Springer Berlin Heidelberg, 2012, volumen 12, ISBN: 978-3-642-27644-6.

[36] P. Yap, "Grid-Based Path-Finding," *Advances in Artificial Intelligence*, volumen vol. 2338, R. Cohen y B. Spencer, edición, páginas 44-55, 2002.

[37] M. Zhao, H. Lu, S. Yang y F. Guo, "The Experience-Memory Q-Learning Algorithm for Robot Path Planning in Unknown Environment," *IEEE Access*, volumen 8, páginas 47 824-47 844, 2020, ISSN: 2169-3536.

[38] C. Zhou, B. Huang y P. Fränti, "A review of motion planning algorithms for intelligent robots," *Journal of Intelligent Manufacturing*, volumen 33, número 2, páginas 387-424, 2022, ISSN: 0956-5515.