Universidad Carlos III de Madrid
Escuela Politécnica Superior

Aprendizaje por Refuerzo en Espacios de Estados Continuos

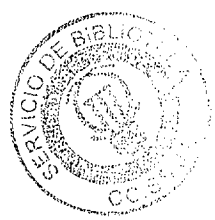
Tesis Doctoral

Fernando Fernández Rebollo
Leganés, 2002

Departamento de Informática

Escuela Politécnica Superior
Universidad Carlos III de Madrid

Aprendizaje por Refuerzo en Espacios de Estados Continuos



AUTOR: Fernando Fernández Rebollo
DIRECTOR: Daniel Borrajo Millán

D. FERNANDO FERNÁNDEZ REBOLLO, con D. N. I. 3.863.926 H

AUTORIZA:

A que su tesis doctoral con el título: **"Aprendizaje por refuerzo en espacios de estados continuos"** pueda ser utilizada para fines de investigación por parte de la Universidad Carlos III de Madrid.

Leganés, 1 de abril de 2003

Fdo.: Fernando Fernández Rebollo

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día ..1.... de ..FEBRERO..... de 2003.

Presidente: D. Arturo Ribagorda Garnacho

Vocal: D. Miguel Angel Salichs Sánchez-Caballero

Vocal: D. Dario Maravall Gómez-Allende

Vocal: D. Eva Onaindía de la Rivaherrera

Secretario: D. Pedro Isasi Viñuela

Realizado el acto de defensa y lectura de la Tesis el día ...1... de Abril..... de 2003 en ...Leganés.....

Calificación: Sobresaliente Cum Laude por unanimidad

EL PRESIDENTE

LOS VOCALES

EL SECRETARIO

A Ana

Agradecimientos

Me es muy difícil concretar cuándo comencé esta tesis doctoral. Y no es porque haya pasado tanto tiempo como para haberlo olvidado, sino porque no hubo un instante inicial o un día concreto en el que decidiera realizarla. Quizá fuese el día que creé un directorio denominado "tesis" en mi ordenador, o quizá antes, cuando mi director, Daniel Borrajo, me entregó una estupenda introducción al Aprendizaje por Refuerzo, titulada "*Reinforcement Learning: a Survey*", de L. P. Kaelbling, M. L. Littman y A. W. Moore, cuando todavía no había terminado mis estudios en Ingeniería Informática.

Y si no tengo claro cuándo comencé la tesis, tampoco estoy seguro de que esta memoria sea su fin, sino sólo una etapa más, puesto que la investigación que en ella desarrollo me proporciona además de trabajo, diversión. Y espero seguir divirtiéndome durante mucho tiempo más.

En el camino está toda la gente que me ha apoyado. Quisiera comenzar recordando a todos mis compañeros del grupo SCALAB, por toda la colaboración que me han ofrecido desde que comencé a trabajar con ellos. También a mis compañeros de doctorado, que además de ésto han sido compañeros de juego, de cafés, de tapas, y sobre todo, amigos.

Y dado que hay vida tras las puertas de la universidad, también hay gente fuera de ella a la que me gustaría agradecer el haberme dado su apoyo durante todo este tiempo. Amigos de Toledo, Calera y Villacañas se merecen que les incluya aquí, aunque sólo sea como disculpa por esas veces que he dejado de estar con ellos por poder escribir estas líneas.

Y cómo no incluir aquí a mis padres, que han vivido día a día todo este trabajo, animándome en los momentos que me han visto más agotado, reponiendo mis energías con cariño; y a mis hermanos, Joaquín y Virginia, que tanta alegría le dan a mi vida junto a Estíbaliz y a Luis; y a la que ya viene en camino, que simplemente con venir, ya me hace feliz.

Y para finalizar, quisiera agradecer a Ana tantas cosas, que quizá sería demasiado largo de escribir. En vez de ello, espero tenerla siempre a mi lado para poder recompensarla.

II

Resumen

El aprendizaje por refuerzo es un modelo de aprendizaje que permite implementar comportamientos inteligentes de forma automática, sin que el diseñador tenga que incorporar conocimiento o modelos del dominio en el que está trabajando. La mayor parte de la teoría del aprendizaje por refuerzo tiene su fundamento en la programación dinámica, y por tanto, en lo que se denominan funciones de valor. Estas funciones dan información sobre lo valioso que es para el desarrollo de una tarea, el encontrarse en una determinada situación o estado, e incluso lo valioso que es ejecutar una determinada acción, dando por hecho que el sistema se encuentra en un determinado estado. Estas funciones, generalmente implementadas como tablas, son utilizadas para obtener de forma directa la política de acción que debe guiar el comportamiento del sistema. Sin embargo, la implementación tradicional de estas funciones en forma tabular no es práctica cuando el espacio de estados es muy grande, o incluso infinito. Cuando se produce esta situación, se deben aplicar métodos de generalización que permitan extrapolar la experiencia adquirida para un conjunto limitado de estados, a la totalidad del espacio, de forma que se consigan comportamientos óptimos en cualquier situación, aunque ésta no hubiera sido explorada con anterioridad.

En la actualidad, existen dos aproximaciones básicas para resolver este problema. Por un lado, están aquellas técnicas que se basan en obtener una discretización adecuada del espacio de estados, de forma que el espacio de estados continuo es traducido a un espacio de estados finito y más reducido. Por otro lado, están los métodos basados en implementar las funciones de valor con algún método supervisado de aproximación de funciones, como, por ejemplo, una red de neuronas. En esta tesis doctoral se pretende desarrollar métodos de aprendizaje por refuerzo que sean aplicables en dominios con espacios de estados continuos, partiendo de las dos aproximaciones planteadas anteriormente. Para ello, se trata de fundir las ventajas de una y otra en un método eficaz y eficiente que permita que el aprendizaje sea un proceso totalmente automático en el que el diseñador tenga que incluir la menor cantidad posible de información sobre la tarea a desarrollar.

Abstract

Reinforcement Learning is a technique that allows to implement intelligent behaviours automatically without the need of introducing knowledge or models about the domain. Most of the reinforcement learning theory is based on dynamic programming, and hence, on value functions. These functions provide information about how good it is, in order to solve a defined task, to be in a given situation in the domain, typically named state, or even how good it is to execute a defined action if the system is in a given state. These functions, typically implemented using look-up tables, are used to represent the action policy that must guide the behaviour of the system. However, the traditional implementation of these functions as look-up tables is not practical when the state space is very large, or even infinite. When one of these situations appears, generalization methods must be applied in order to extrapolate the acquired experience for a limited set of states, to the whole space, so optimal behaviours can be achieved, even when the whole domain has not been explored.

Two main approaches can be found in the literature. On the one hand, there are methods based on learning an adequate state space discretization, so the continuous state space is mapped to a finite and reduced one. On the other hand, methods based on implementing the value functions with some supervised learning technique for function approximation, for instance, a neural network, can be found. This dissertation tries to develop reinforcement learning methods that can be applied in domains with a continuous state space. The start point is given by the two approaches above, and it tries to join the advantages of one and another in an efficient and effective method that allows the learning process be a fully automatic process where the designer has to introduce the less possible amount of information about the task to solve.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

El comportamiento inteligente es un elemento presente en muchos de los sistemas con los que estamos acostumbrados a tratar de forma cotidiana, desde aparatos aparentemente tan sencillos como un ascensor, hasta otros tan complejos como una central química. Dentro de este amplio abanico de sistemas se encuentran algunos donde la interacción con el entorno es muy activa y dinámica, como puede ser un robot que sirva de guía de un museo, o un coche que conduzca autónomamente. A la hora de proporcionar inteligencia a estos sistemas, pueden aparecer dos características que los define y agrupa en el conjunto de problemas que se tratarán en esta tesis. En primer lugar, el aprendizaje de una tarea por parte del sistema o agente se realiza mediante un proceso iterativo de prueba y error en el entorno con el que interactúa. En segundo lugar, la forma en que el entorno informa al agente sobre si está haciendo bien o mal la tarea que está aprendiendo se realiza a través de una señal de refuerzo, que puede recibirse retardada en el tiempo. El cómo realizar ese proceso de prueba y error y el cómo tratar esta señal de refuerzo para que el sistema aprenda de forma eficiente un comportamiento, deseablemente óptimo, han sido objeto de estudio desde los primeros trabajos de programación dinámica [Bellman, 1957], y se han unificado bajo el término de aprendizaje por refuerzo [Kaelbling *et al.*, 1996].

La aparición de nuevos campos como la robótica, dieron un auge importante a este tipo de técnicas, definiendo como objetivo la búsqueda de políticas de comportamiento óptimas para realizar determinadas tareas, introduciendo nuevos problemas a resolver [Mahadevan and Connell, 1992]. Uno de estos problemas consiste en cómo aplicar técnicas definidas para dominios discretos y probablemente de tamaño pequeño, como la programación dinámica, a dominios de gran tamaño o incluso infinitos [Santamaría *et al.*, 1998]. Las técnicas de generalización aparecieron con este objetivo, si bien la mayor parte de estas técnicas han derivado de otras disciplinas o han sido

adaptadas a este nuevo tipo de problemas.

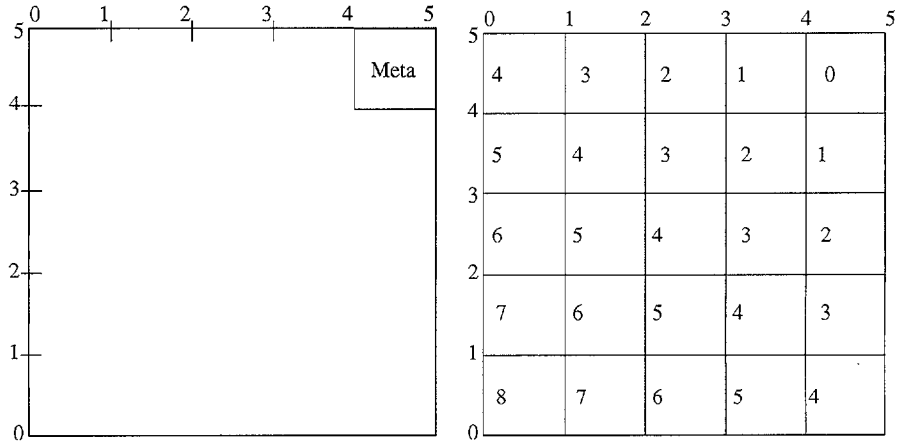
1.1. Planteamiento del Problema

Un primer paso para la realización de la tesis doctoral consiste en la revisión de las técnicas de aprendizaje por refuerzo existentes, haciendo especial hincapié en la definición de los principales conceptos heredados de la programación dinámica, pero teniendo en cuenta que el objetivo de la tesis doctoral se centra en sistemas donde no existe un modelo "a priori", y donde la adquisición y utilización de este modelo es suficientemente costosa como para preferir un sistema de aprendizaje directo, que no requiera el conocimiento de dicho modelo.

Una vez centrado el problema en métodos de aprendizaje por refuerzo directos, se plantea la necesidad de revisar la problemática que el uso de estos algoritmos de aprendizaje introducen en dominios con espacios de estados de gran tamaño. Quizá también por herencia de la programación dinámica, muchos de estos algoritmos plantean una representación tabular del comportamiento o política de acción. Esta representación tabular hace que cuando el espacio de estados o de acciones crece, el aprendizaje se haga impracticable, no sólo por requisitos de memoria, sino por el escaso aprovechamiento que se hace del conocimiento adquirido en el proceso de prueba y error. Además, puede ocurrir que el espacio de estados y acciones sea continuo, impidiendo estas representaciones tabulares. Esto plantea la necesidad de utilizar técnicas que permitan, a partir de una experiencia reducida sobre un conjunto limitado de estados o acciones, generalizar a un conjunto mucho más extenso. De esta forma, se plantea la definición de un marco que integre las técnicas de generalización no como un elemento adicional a una de las técnicas de aprendizaje por refuerzo existentes, sino como parte de ella.

Un ejemplo sencillo del problema de la generalización en espacios de estados continuos es el introducido en la Figura 1.1(a). En ella se plantea el problema de un robot que debe aprender a llegar a una zona de meta fija en un entorno, desde cualquier posición inicial de dicho entorno. En todo momento, el robot dispone de sus coordenadas x e y, que pueden tomar valores continuos en el rango $[0, 5]$. Para alcanzar la meta, el robot sólo puede ejecutar 4 acciones, correspondientes a movimientos de tamaño 1 en dirección Norte, Sur, Este y Oeste. En la Figura 1.1(b) se muestra una división del entorno en regiones que definen claramente el número de acciones que necesita ejecutar el robot para llegar a la meta. Así, la región correspondiente con la región meta tiene un valor 0, mientras que las regiones situadas al oeste y al sur de la meta, desde las que con una única acción (ir al este e ir al norte,

respectivamente) el número de acciones a ejecutar para alcanzar la meta es 1.



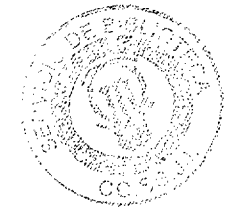
(a) Dominio de Robot (b) Discretización 5×5

Figura 1.1: Ejemplo de problema de Aprendizaje por Refuerzo.

Obviamente, esta división (o discretización) del espacio de estados en regiones resulta muy provechosa, ya que todos los estados que se encuentran dentro de cada una de esas regiones, pueden tratarse por igual. Por tanto, el conocimiento que se obtenga para cualquier estado de cualquier región, puede generalizarse a cualquier estado de su misma región. No obstante, y como se planteará a lo largo de esta tesis doctoral, encontrar un método de generalización adecuado, no siempre es una tarea sencilla.

1.2. Soluciones Actuales

El conjunto de técnicas de generalización existentes puede dividirse en dos grandes grupos, dependiendo de la forma en que se representen las políticas que guían el comportamiento del agente en cuestión. El primero de ellos plantea romper con la representación tabular definida anteriormente para las políticas, introduciendo un problema de aproximación de funciones [Bertsekas and Tsitsiklis, 1996] que puede ser lineal o no. Dado que la políticas pueden representarse mediante una función de utilidad de estados y acciones, una buena aproximación es estimar esta función con cualquier método de aprendizaje supervisado en paralelo al proceso de prueba y error. No obstante, y como se mostrará posteriormente, esta aproximación introduce grandes problemas de convergencia, dado que al realizar el aprendizaje a partir de



ejemplos de entrenamiento limitados, los pequeños errores de clasificación que se producen al inicio del aprendizaje en determinadas zonas del espacio de estados son propagados durante el proceso de prueba y error al resto del espacio, impidiendo que la función de utilidad sea aproximada correctamente.

Por otro lado, hay técnicas que mantienen la representación tabular mediante la agrupación del espacio de estados en conjuntos de estados reducidos, de forma que se puede hacer una traducción de cualquier estado del conjunto inicial a un estado del conjunto reducido. Se trata también, por tanto, de una aproximación de funciones, pero en este caso, lineal [Sutton and Barto, 1998]. Los principales problemas que estas técnicas introducen son, por un lado, definir cuál es el número de estados adecuado para la nueva representación, y por otro lado, definir los límites entre esos estados. Para resolver estos problemas se suelen plantear dos métodos básicos. En el primero se plantea probar con discretizaciones uniformes con distinto número de estados, hasta que se obtiene uno que produce unos resultados, al menos, cercanos a los esperados [Fernández and Borrajo, 2000].

En el ejemplo de la Figura 1.1(a), el conjunto de regiones definido en la Figura 1.1(b) muestra una discretización del espacio de estados en 25 regiones, que permiten definir perfectamente el número de acciones necesarias para alcanzar la meta. Sin embargo, otras discretizaciones distintas podrían no ser capaces de recoger esa información, lo que como se mostrará posteriormente, es muy importante para realizar un buen aprendizaje. Este método sólo es válido cuando el espacio de estados no es muy elevado, y cuando la dimensión de este espacio es también pequeño (dos o tres dimensiones).

Esto es debido principalmente a que los métodos uniformes discretizan por igual todo el espacio de estados, sin prestar más atención a unas zonas u otras en función de lo críticas que son para realizar un buen aprendizaje. Esto se intenta resolver utilizando métodos de discretización variable, que definen una discretización inicial del entorno, y posteriormente, aumentan la resolución de la discretización en aquellas zonas que se considera necesario [Moore, 1991]. Sin embargo, la mayor parte de la experimentación que se encuentra en la bibliografía no suele incluir espacios de estados de más de 2 ó 3 dimensiones. Estas técnicas también presentan la necesidad de incluir un parámetro que define la resolución que se debe alcanzar durante el aprendizaje, por lo que también requiere de un proceso de refinamiento de este parámetro. Por último, la discretización gruesa del espacio de estados suele producir la pérdida de la propiedad de Markov [Moore and Atkeson, 1995], en la que se basa toda la teoría del aprendizaje por refuerzo [Puterman, 1994]. La pérdida de esta propiedad, tal y como se mostrará a lo largo de esta tesis, produce efectos muy indeseables en el aprendizaje y en los resultados que se obtienen.

1.3. Objetivos

Los objetivos de esta tesis doctoral (que se detallan en la sección 3) se centran en la obtención de métodos de aprendizaje por refuerzo libre de modelo que sean capaces de aprender en dominios continuos y de gran tamaño, y que contemplen toda la problemática resumida anteriormente, tanto de los métodos de aproximación de funciones supervisada, como de los métodos basados en la obtención de nuevos espacios de estados reducidos. Para ello, se pretende revisar las características fundamentales de ambos métodos, la problemática que introducen, y cuáles son sus principales aportaciones, para obtener una aproximación que reúna las ventajas de ambos, pero sin incluir sus inconvenientes. Para ello, se parte de que la capacidad de convergencia de los métodos basados en la discretización del espacio de estados está mucho más asegurada que en los métodos de aproximación de funciones supervisada. Sin embargo, también hay que tener en cuenta que el aprendizaje de un nuevo espacio de estados reducido debe estar supervisado por la política que se está aprendiendo, y que, de una forma u otra, los errores introducidos por este método supervisado deben ser depurados.

1.4. Organización de la Memoria

Esta tesis doctoral se ha organizado del siguiente modo. En el segundo capítulo se repasan el aprendizaje por refuerzo y las principales técnicas de generalización que se han aplicado hasta la actualidad. Con este capítulo se pretende enmarcar el trabajo desarrollado en esta tesis doctoral, introduciendo poco a poco su motivación.

El capítulo 3 expone los objetivos de esta tesis doctoral, haciendo especial hincapié en la definición de las principales características que se desean obtener con el método a desarrollar. Para verificar el cumplimiento de estos objetivos, se definen una serie de variables que permitirán realizar una evaluación objetiva de los resultados obtenidos.

En el capítulo 4 se expone el modelo de aprendizaje por refuerzo VQQL, (*Vector Quantization for Q-Learning*). Dicho modelo plantea el uso de métodos de cuantificación vectorial para discretizar el espacio de estados de forma no supervisada, y posteriormente aplicar *Q-Learning* con el fin de aprender la política de acción sobre la discretización obtenida. En ese capítulo se plantea el modelo, así como una buena parte de la experimentación realizada con él, y las principales conclusiones obtenidas.

El capítulo 5 describe un método de clasificación supervisada basada en el vecino (o prototipo) más cercano denominado ENNC (*Evolutionary Nearest*

Neighbour Classifier). Este algoritmo de clasificación surge por la necesidad de supervisar la discretización del espacio de estados y con el fin de cumplir ciertos requisitos derivados de la experiencia obtenida con el modelo VQQL. En este capítulo se realiza una descripción detallada del método, así como parte de la experimentación realizada para validar el método como algoritmo de clasificación.

En el capítulo 6 se muestra la aplicación del algoritmo *ENNC* como método para, por un lado, aproximar la función de valor o utilidad y, por otro lado, discretizar el espacio de estados. Esta aplicación, integrada en lo que se ha denominado ENNC-QL, representa la consecución de los objetivos planteados al inicio de la tesis doctoral, y es validada mediante comparativas con varios métodos de aprendizaje en distintos dominios. Todas estas comparativas se encuentran detalladas en el capítulo 7.

Por último, en el capítulo 8 se muestran las principales conclusiones derivadas de la tesis doctoral, un resumen de sus principales aportaciones. y las líneas de investigación y trabajos que plantea para el futuro.

Capítulo 2

Estado del Arte

El objetivo de este capítulo es describir el aprendizaje por refuerzo como un conjunto de problemas con unas características comunes. Estas características se centran principalmente en dos puntos. Primero, en el aprendizaje por refuerzo siempre se plantea el problema de un agente que debe realizar un aprendizaje por prueba y error en un determinado entorno. Segundo, este proceso de prueba y error que realiza el agente está guiado por ciertas señales de refuerzo o ganancia que se reciben desde el entorno, y que se pretende maximizar siguiendo alguna medida de ganancia o utilidad.

Para realizar esta descripción, se introduce primero el concepto de aprendizaje, y la relación del aprendizaje por refuerzo con otros métodos de aprendizaje. Posteriormente, en la sección 2.2 se definen los principales conceptos del aprendizaje por refuerzo y cómo se relacionan, a la vez que se presenta un marco matemático basado en los procesos de decisión de Markov (MDPs). En la sección 2.3 se presentan las principales técnicas de aprendizaje por refuerzo, desde aquéllas que presuponían un conocimiento matemático completo del problema que se trataba hasta aquéllas que "a priori" tienen un conocimiento nulo del dominio. El grado de conocimiento que se tiene de este modelo es un criterio de clasificación típicamente aplicado, ya que este conocimiento define en gran medida el tipo de técnica a utilizar, por lo que este trabajo también sigue ese criterio. La sección 2.4 describe más extensamente la necesidad de la generalización en la mayoría de los dominios de aplicación, así como las técnicas más utilizadas, siguiendo como principal criterio de clasificación el introducido en la sección 1.2. La sección 2.5 plantea un marco unificado de las técnicas de generalización con las distintas técnicas de aprendizaje por refuerzo definidas, tratando el problema de la generalización como un problema de conocimiento y representación del dominio. La sección 2.6 plantea algunos problemas generales a tener en cuenta en el aprendizaje por refuerzo y, para finalizar este capítulo, la sección 2.7 muestra algunos dominios de

experimentación utilizados en la bibliografía.

2.1. Aprendizaje: Definición y Clasificación

El aprendizaje automático trata la cuestión de cómo construir programas de ordenador que sean capaces de mejorar con la experiencia [Mitchell, 1997]. Cuando se utiliza el término mejorar, se refiere a la capacidad de desarrollar una tarea cada vez mejor, teniendo en cuenta alguna medida de rendimiento o satisfacción, que indica lo bien o mal que se realiza dicha tarea.

Uno de los puntos esenciales en cualquier proceso de aprendizaje automático consiste en qué tipo de experiencia se posee para realizar ese proceso de aprendizaje, y que normalmente aparece en forma de ejemplos de entrenamiento. Por un lado, se puede disponer de ejemplos de entrenamiento directos. Con dichos ejemplos, se supone que se dispone "a priori" de un entrenador que, para cada decisión o acción ejecutada por el sistema, indica cuál debería haber sido la decisión correcta. En función de si la decisión coincide o no con la correcta, el sistema recibe una realimentación orientada a mantener sus decisiones, si es que eran acertadas, o a modificarlas, en caso contrario. Aquellos problemas de aprendizaje en los que se dispone de esta información se denominan de aprendizaje supervisado, ya que se dispone de un tutor que indica (o supervisa) cuál debería haber sido la acción elegida. Uno de los problemas más típicos que corresponden a esta descripción son los problemas de clasificación, en la que se dispone de una experiencia de ejemplos correctamente clasificados, y el programa de ordenador debe aprender a clasificar correctamente no sólo esos ejemplos, sino otros ejemplos distintos. Este problema de clasificación se describe a continuación:

- Dado un conjunto de etiquetas (clases), $S = \{s_1, \ldots, s_L\}$, y
- Dado un conjunto de ejemplos etiquetados, $V = \{v_1, \ldots, v_M\}$, y
- Dado un nuevo ejemplo sin etiquetar, x,
- Encontrar la etiqueta correcta para x.

Una de las técnicas de clasificación más usadas es la basada en el vecino o prototipo más cercano, que utiliza el concepto de cercanía o lejanía en alguna medida de distancia definida por el programador [Aha, 1997, Bezdek and Kuncheva, 2001]. La clasificación basada en el vecino más cercano puede definirse como:

- Dado un conjunto de etiquetas, $S = \{s_1, \ldots, s_L\}$, y

- Dado un conjunto de ejemplos etiquetados, $V = \{v_1, \ldots, v_M\}$, y
- Dado un conjunto de ejemplos etiquetados, $V' = \{v'_1, \ldots, v'_N\}$, generado a partir de V, con $N << M$, y
- Dado un nuevo ejemplo sin etiquetar, x,
- Asignar al nuevo ejemplo, x, la etiqueta del ejemplo en V' más cercano.

Encontrar un conjunto V' adecuado sería el objetivo indirecto del proceso de aprendizaje. En el lado opuesto a los algoritmos supervisados se encuentran problemas donde esa supervisión no existe, y es el diseñador el que debe definir una medida de satisfacción, más o menos artificial. A estos problemas se les suele denominar de aprendizaje no supervisado. Esta medida de satisfacción suele estar relacionada con la agrupación de ejemplos que tienen características similares. Este concepto de similitud debe ser introducido por el diseñador, en función de una medida de distancia parecida a la del caso supervisado. El problema de agrupación no supervisada se describe como:

- Dado un conjunto de ejemplos, $V = \{v_1, \ldots, v_M\}$, y
- Dado un nuevo ejemplo , x,
- Asociar el nuevo ejemplo a un subconjunto de ejemplos de V, teniendo en cuenta algún criterio de similitud.

En este caso, también puede realizarse una agrupación basada en el vecino o prototipo más cercano, tal y como sigue:

- Dado un conjunto de ejemplos, $V = \{v_1, \ldots, v_M\}$, y
- Dado un conjunto de ejemplos , $V' = \{v'_1, \ldots, v'_N\}$, generado a partir de V, con $N << M$, que representan respectivamente a N subconjuntos de ejemplos de V, y
- Dado un ejemplo, x,
- Asociar el nuevo ejemplo al elemento de V', v'_i, más cercano. Como v'_i, a su vez, representa un subconjunto de ejemplos de V, se asocia x a ese subconjunto de V.

Nuevamente, el principal inconveniente es encontrar el conjunto V' adecuado. Por último, existen problemas donde se dispone de información sobre lo bien o mal que se resuelve una tarea en su conjunto, pero no de lo bueno o

malos que fueron los pasos concretos que se llevaron a cabo para solucionarla. En este caso, se dice que se dispone de ejemplos de entrenamientos indirectos, donde es difícil dar un valor positivo o negativo a las acciones a partir del resultado global. Se plantea, por tanto, un problema de asignación de crédito desde el resultado global a las decisiones particulares. Este tipo de problemas suele enmarcarse en lo que se denomina aprendizaje por refuerzo, y tienen como objetivo generar políticas de comportamiento que son evaluadas no a corto plazo, sino a lo largo del tiempo. Estos problemas pueden formalizarse como se muestra a continuación:

- Dado un agente que debe desarrollar una tarea en un determinado entorno, y
- Dado un conjunto de experiencias $V = \{v_1, \ldots, v_M\}$ que relacionan situaciones en las que se encuentra el agente, s, acciones que puede llevar a cabo, a, y refuerzos inmediatos que recibe del entorno, r, lo que forma las ternas $(< s, a, r >)$,
- Asociar el nuevo estado con la acción que maximice el refuerzo que se espera recibir a lo largo del tiempo.

En este caso se observa que, para maximizar los refuerzos a lo largo del tiempo, no basta con buscar para cada nueva situación s, ejemplos similares en el conjunto V, ya que eso sólo maximiza el refuerzo inmediato en ese instante, y no el refuerzo a lo largo del tiempo, que es lo que asegura la ejecución satisfactoria de la tarea que se desea resolver. El tema central de esta tesis doctoral se centra en este tipo de aprendizaje. No obstante, el aprendizaje por refuerzo requiere en muchas ocasiones de técnicas y/o métodos de aprendizaje, tanto supervisado, como no supervisado, tal y como se mostrará a lo largo de esta memoria.

2.2. El Aprendizaje por Refuerzo

El aprendizaje por refuerzo consiste en aprender a decidir, ante una situación determinada, qué acción es la más adecuada para lograr un objetivo. Una de las características fundamentales de este aprendizaje es que es un aprendizaje basado en prueba y error. Al igual que ocurre en gran medida en el aprendizaje en el mundo animal, el aprendizaje por prueba y error se puede resumir en dos aspectos definidos en la Ley del Efecto (*Law of Effect*) [Thorndike, 1911]. El primero de ellos es que el aprendizaje por prueba y error es *selectivo*, dando a entender que involucra la selección de varias alternativas,

y la selección, de entre ellas, de la mejor en función de sus consecuencias. El segundo es que es *asociativo*, en el sentido de que las alternativas encontradas se asocian a las situaciones particulares en que se tomaron.

Como se mostrará a lo largo de este trabajo, estos dos aspectos están presentes en la aplicación computacional del aprendizaje por refuerzo, si bien la teoría matemática que se aplica en su desarrollo tiene su fundamento en los trabajos de programación dinámica de Bellman y toda la teoría de control óptimo que se comenzó a desarrollar a mediados de los 50 [Bellman, 1957, Bellman and Kalaba, 1965, Kumar, 1986]. Ambas ramas, aprendizaje por prueba y error, y control óptimo, se unen en el aprendizaje por refuerzo, permitiendo el desarrollo de toda una teoría que se repasa en este capítulo.

Algunos libros que cubren de forma extensa y detallada el aprendizaje por refuerzo son [Sutton and Barto, 1998] y [Bertsekas and Tsitsiklis, 1996], así como dos ediciones especiales de la revista *Machine Learning*, [Sutton, 1992, Kaelbling, 1996] en la que se encuentran algunos de los trabajos más importantes de los últimos años. [Kaelbling *et al.*, 1996] es un estupendo resumen.

2.2.1. Definición

El modelo de aprendizaje por refuerzo se muestra en la Figura 2.1.

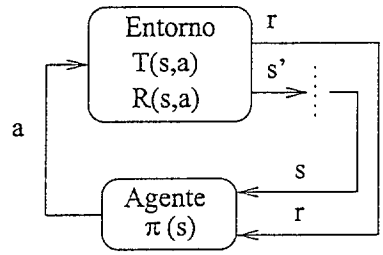


Figura 2.1: Modelo de Aprendizaje por Refuerzo.

Un agente hardware o software está conectado a su entorno vía percepción y acción. En cada instante el agente recibe desde el entorno el estado en el que se encuentra, s; entonces el agente decide ejecutar una acción, a, que genera como salida. Esta salida cambia el estado del entorno a s', que es transmitido al agente junto a una señal de refuerzo r, tal y como se muestra en la figura.

El comportamiento o política, π, del agente debe ser tal que elija acciones que incrementen la suma de todas las señales de refuerzo recibidas a lo largo del tiempo. Formalmente, el modelo consta de:

1. Un conjunto discreto de estados, $\mathcal{S}$;
2. Un conjunto discreto de acciones del agente, $\mathcal{A}$;
3. Un conjunto de señales de refuerzo escalares, $\mathcal{R}$.

La figura también incluye varias funciones que, en principio, son desconocidas: una función que realiza las transiciones de estado, T, y otra, R, que calcula el refuerzo que recibe el agente en cada momento. Con estas funciones, se termina la descripción de los denominados *Procesos de Decisión de Markov* (MDPs) [Puterman, 1994], que es el modelo más utilizado para definir los problemas de refuerzo retardado, y que define las funciones T y R como:

- $T : \mathcal{S} \times \mathcal{A} \rightarrow P(\mathcal{S})$, donde un miembro de $P(s)$ es una distribución de probabilidad sobre el conjunto $\mathcal{S}$ (es decir, transforma estados en probabilidades). Decimos que $T(s, a, s')$ es la probabilidad de que se realice una transición desde s hasta s' ejecutando la acción a.
- $R : \mathcal{S}x\mathcal{A} \rightarrow \mathcal{R}$, que para cada par estado acción proporciona su refuerzo.

El trabajo del agente es encontrar una política, π, que para cada estado $s \in S$, decida cuál es la acción, $a \in A$, que debe ser ejecutada, de forma que se maximice alguna medida de refuerzo a largo plazo. Dicha medida viene definida por el criterio de horizonte infinito descontado, definido en la ecuación 2.1, y que utiliza un parámetro de descuento, γ, que reduce la influencia del refuerzo, r_k, recibido en el futuro.

$$\sum_{k=0}^{\infty} \gamma^k r_k \tag{2.1}$$

Existen otros criterios de optimalidad, como el de horizonte finito, en el que el sumatorio se limita a un número de acciones predefinido, o incluso modelos que no incluyen el factor de descuento, asignando la misma importancia a todos los refuerzos recibidos a través del tiempo. Sin embargo, estos dos últimos modelos tienen inconvenientes, como el de tener que definir una longitud fija de las secuencias de acciones, en el método de horizonte finito, o el de no saber distinguir cuándo se ejecutaron las acciones verdaderamente útiles, en aquellos modelos que no introducen factor de descuento [Kaelbling *et al.*, 1996]. Por estas razones, en el resto de este trabajo se supondrá un modelo de optimización de horizonte infinito descontado, si bien no hay que olvidar los otros dos modelos, que en ocasiones podrían resultar muy útiles.

2.2.2. Procesos de Decisión de Markov

En los problemas de aprendizaje por refuerzo, la toma de decisión de los agentes es una función del denominado "estado" del entorno. La definición de la información que se introduce en este estado es fundamental a la hora de afrontar cualquier problema de aprendizaje por refuerzo. Esto está íntimamente relacionado con el trabajo que se plantea: la obtención de una representación eficiente del espacio de dichos estados. Sin embargo, son problemas que en principio podrían tratarse por separado. Por un lado está la decisión de qué información es útil para afrontar un problema, y por otro, cómo representar esa información eficientemente. En esta sección, nos centramos en el primero de los problemas.

Según el modelo introducido anteriormente, dado que la acción del agente depende sólo del estado actual del entorno, y no de acciones pasadas, es deseable una señal de estado que resuma adecuadamente ese pasado ocurrido. Es decir, no importa qué acciones se hayan llevado a cabo para alcanzar el estado actual, pero el estado actual es suficiente para decidir cuáles deben ser las acciones futuras. A este tipo de señales de estado que resumen toda la información relevante pasada se le denominan *Markovianas*, o que poseen la propiedad de Markov [Bellman, 1957, Howard, 1960, Puterman, 1994]. Dicha propiedad se define matemáticamente mediante la ecuación 2.2.

$$\begin{aligned} &Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} = \\ &\quad Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0, \} \end{aligned} \tag{2.2}$$

Según dicha ecuación, la probabilidad de que el agente se encuentre en un estado y que reciba un determinado refuerzo sólo depende del estado en el que se encontraba y la acción ejecutada en el instante inmediatamente anterior.

La importancia de la propiedad de Markov viene dada por el hecho de que en los problemas de aprendizaje por refuerzo se asume que cada decisión se realiza solamente en función del estado actual, y no del camino que se ha trazado para llegar a ese estado.

Un problema de aprendizaje por refuerzo se denomina Proceso de Decisión de Markov (MDP) si satisface la propiedad de Markov. Si el número de estados y de acciones definidas en el modelo son finitos, se denomina proceso de decisión de Markov finito. Un MDP finito viene definido por sus conjuntos de estados y de acciones y por la dinámica del entorno. Dicha dinámica se resume en las probabilidades de las transiciones de estado y de las funciones de refuerzo, que se definen en las ecuaciones 2.3 y 2.4 respectivamente.

$$T(s, a, s') = P^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{2.3}$$

donde $Pr\{\}$ define una probabilidad.

$$R(s,a) = R_s^a = E\{r_{t+1}|s_t = s, a_t = a\} \tag{2.4}$$

donde $E\{\}$ define una esperanza sobre el valor a recibir.

Para concluir, se puede decir que se tiene un conocimiento completo del modelo de un MDP finito si se conocen los conjuntos de estados y acciones y las probabilidades que definen su dinámica.

2.2.3. Funciones de Valor

Casi todos los algoritmos de aprendizaje por refuerzo están basados en la aproximación de las funciones de valor (*"value functions"*) que estiman cómo de bueno es para un agente estar en un determinado estado, o incluso cómo de bueno es ejecutar una determinada acción desde un determinado estado. El valor de un estado, s, bajo una política π, que se denota con $V^\pi(s)$ es el refuerzo que se espera obtener si comenzamos a guiarnos por la política de acción π desde el estado s hasta el infinito. Esta definición queda formalizada para MDP's en la ecuación 2.5.

$$V^\pi(s) = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\} \tag{2.5}$$

donde $E_\pi\{\}$ denota el valor esperado dado que el agente sigue la política π. A V^π se le suele denominar función de valor-estado (*"state-value function"*).

De la misma forma, se puede definir $Q^\pi(s,a)$ como el valor de ejecutar una determinada acción a desde un estado s siguiendo una política π; es decir, como el refuerzo que se espera obtener si comenzamos a guiarnos por la política de acción π tras ejecutar la acción a desde el estado s, tal y como muestra la ecuación 2.6.

$$Q^\pi(s,a) = E_\pi\{R_t|s_t = s, a_t = a\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s, a_t = a\} \tag{2.6}$$

A Q^π se le suele denominar función de valor-acción (*"action-value function"*). En ambas ecuaciones se ha introducido el parámetro γ como factor de descuento de las futuras acciones, siguiendo un criterio de optimalidad de horizonte infinito descontado en el tiempo definido anteriormente.

La utilidad de estas funciones de valor es que se pueden usar para definir un orden de calidad entre políticas. Así, se puede decir que una política π es mejor o igual que otra política π' si su ganancia esperada es mayor o igual

que la de la segunda para todos los estados, tal y como muestra la ecuación 2.7.

$$\pi \geq \pi' \text{ si y sólo si } V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in \mathcal{S} \tag{2.7}$$

Además, siempre hay una o más políticas que son mejores o iguales que el resto de políticas, que se definen como políticas óptimas y que se denotan por π^*. Estas políticas óptimas comparten una función de valor-estado, $V^*(s)$, y una función de valor-acción $Q^*(s,a)$ que sí son únicas y se definen en las ecuaciones 2.8 y 2.9.

$$V^*(s) = \max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S} \tag{2.8}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2.9}$$

2.3. Métodos de Resolución Tradicionales

Los problemas de aprendizaje por refuerzo suelen dividirse en dos clases fundamentales que vienen dadas por el conocimiento que se tiene sobre el modelo o el dominio a tratar [Kaelbling *et al.*, 1996]. Si se tiene un conocimiento completo del MDP, es decir, los conjuntos de estados y acciones y la dinámica del entorno representada anteriormente con las probabilidades $P^a_{ss'}$ y R^a_s, se pueden aplicar directamente técnicas de programación dinámica [Bellman, 1957], muy desarrolladas matemáticamente y ampliamente utilizadas en problemas de optimización en los que se dispone de toda la información necesaria.

Por otro lado, cuando no se dispone de ese conocimiento sobre el modelo a tratar, se suelen seguir dos aproximaciones. En la primera, la solución se encamina a aprender primero el modelo, y de esa forma poder aplicar las técnicas de programación dinámica definidas anteriormente. Estas aproximaciones se engloban en los denominados métodos basados en el modelo [Kumar, 1986, Sutton, 1990]. En la segunda, se buscan técnicas alternativas que puedan aplicarse sin un conocimiento "a priori" del modelo, es decir, métodos libres de modelo [Watkins, 1989].

Aunque existen otras clasificaciones de los algoritmos de aprendizaje por refuerzo se utiliza ésta, ya que es la más cercana al problema de representación que se trata en este trabajo. Además es una clasificación utilizada en otros problemas de decisión. Por ejemplo, toda la teoría de detección y reconocimiento de patrones [Duda and Hart, 1973] se puede resumir en tres aproximaciones. En primer lugar, aquélla en la que se dispone de un conocimiento completo de la física del problema y en la que se pueden utilizar los modelos matemáticos existentes como la teoría bayesiana. En segundo lugar,

si no se conoce esa física, se dispone de dos posibilidades: o aprenderla para así poder aplicar los modelos matemáticos existentes, o utilizar otro tipo de técnicas que permitan hacer el estudio sin conocer la física del problema.

En esta sección se repasan brevemente todas estas técnicas, haciendo especial hincapié en el conocimiento que se tiene del dominio en cada momento.

2.3.1. Programación Dinámica

Las técnicas de programación dinámica se refieren a aquellos algoritmos que pueden ser utilizados para obtener políticas óptimas dado un conocimiento completo de un modelo perfecto del entorno como un Proceso de Decisión de Markov. Al igual que en el resto de técnicas de aprendizaje por refuerzo, la programación dinámica se basa en el mantenimiento de las funciones de valor de cada estado ($V(s)$ y/o $Q(s,a)$) para estructurar el conocimiento. Obteniendo los valores óptimos de ambas funciones se obtiene una representación óptima de la política a seguir, tal y como se mostró en la sección anterior. Dichas funciones de valor óptimas cumplen las ecuaciones de optimalidad de Bellman [Bellman, 1957] (ecuaciones 2.10 y 2.11), y que formulan matemáticamente el principio de optimalidad, descrito a continuación:

Principio de Optimalidad. *Una política óptima tiene la propiedad de que cualesquiera que sean el estado inicial y la primera decisión tomada; el resto de decisiones deben constituir una política óptima respecto al estado resultante de la primera decisión.*

$$\begin{aligned} V^*(s) &= \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\} = \\ &= \max_a \sum_{s'} P^a_{ss'}[R^a_s + \gamma V^*(s')] \end{aligned} \tag{2.10}$$

$$\begin{aligned} Q^*(s,a) &= E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\} = \\ &= \sum_{s'} P^a_{ss'}[R^a_s + \gamma \max_{a'} Q^*(s', a')] \end{aligned} \tag{2.11}$$

para todo $s \in S$, $a \in A(s)$ y $s' \in S$.

La mayoría de las técnicas de programación dinámica se derivan de estas ecuaciones. Por ejemplo, el algoritmo de Iteración de Política (*Policy Iteration*) consiste en una repetición iterativa de dos pasos fundamentales. Por un lado, la evaluación de la política de que se dispone en un determinado momento, es decir, el cálculo de su función de valor-estado. Por otro lado, la mejora de la política mediante dicha función. Este algoritmo se resume en la Figura 2.2.

Iteración de la Política

- Inicialización
  $V(s) \in R$ y $\pi(s) \in A(s)$ arbitrarios para todo $s \in S$
- Repetir
  1. Evaluación de la Política
     - Repetir
       - $\Delta \leftarrow 0$
       - Para cada $s \in S$
         - $v \leftarrow V(s)$
         - $V(s) \leftarrow \sum_{s'} P_{ss'}^{\pi(s)}[R_{ss'}^{\pi(s)} + \gamma V(s')]$
         - $\Delta \leftarrow max(\Delta, |v - V(s)|)$
     - Hasta que $\Delta < \theta$ (un número positivo entero)
  2. Mejora de la Política
     - *política_estable* $\leftarrow$ *cierto*
     - Para cada $s \in S$
       - $b \leftarrow \pi(s)$
       - $\pi(s) \leftarrow \arg\max_a \sum_{s'} P_{ss'}^{a}[R_s^a + \gamma V(s')]$
       - Si $b \neq \pi(s)$, entonces *política_estable* $\leftarrow falso$

  Hasta que *política_estable* = *cierto*

Figura 2.2: Algoritmo de Iteración de Política.

Para evitar tener que recalcular la función de valor de la política en cada iteración del algoritmo, se puede utilizar el algoritmo de Iteración de Valor (*Value Iteration*), que se muestra en la Figura 2.3.

Uno de los principales inconvenientes que presentan estos algoritmos es que todas las operaciones involucradas deben realizarse sobre todo el conjunto de estados. Así, para dominios con un conjunto de estados muy grande, el problema podría hacerse intratable. Los algoritmos de programación dinámica asíncrona [Barto *et al.*, 1995] tratan de resolver este problema mediante la selección en cada momento de qué estados deben actualizarse, de forma que unos estados pueden recibir más actualizaciones que otros. Además, estos métodos permiten unir el cálculo con una interacción en tiempo real del

Iteración del Valor

- Inicializar $V(s)$ arbitrariamente. Por ejemplo, $V(s) = 0, \forall s \in S$
- Repetir
  - $\Delta \leftarrow 0$
  - Para todo $s \in S$
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow \text{máx}_a \sum_{s'} P_{ss'}^{\pi(s)}[R_{ss'}^{\pi(s)} + \gamma V(s')]$
    - $\Delta \leftarrow max(\Delta, |v - V(s)|)$
- Hasta que $\Delta < \theta$ (un número positivo entero)
- Dar como salida una política π tal que $\pi(s) = \arg\text{máx}_a \sum_{s'} P_{ss'}^{a}[R_s^a + \gamma V(s')]$

Figura 2.3: Algoritmo de Iteración de Valor.

agente con el entorno, es decir, se puede ejecutar el algoritmo de programación dinámica a la vez que el agente está experimentando con el MDP. Esta experiencia del agente puede ser utilizada para determinar sobre qué estados del MDP realizar actualizaciones. A su vez, la información actual de la política y las funciones de valor pueden determinar qué acciones puede realizar el agente en su aprendizaje, pudiendo prestar más atención a partes del entorno que se consideran más relevantes. Este esquema es muy general en todas las propuestas de aprendizaje por refuerzo, tal y como se comprobará posteriormente.

2.3.2. Métodos Libres de Modelo

En los métodos libres de modelo, al igual que los métodos basados en el modelo, se asume un desconocimiento completo de la dinámica del entorno. Es decir, se conocen los conjuntos de estados y de acciones del agente, pero no cuáles son sus efectos en el entorno. La diferencia entre los métodos libres de modelo y los basados en el modelo, es que estos últimos tratan de aprender esa dinámica para posteriormente aplicar las técnicas de programación dinámica sobre ellos, convirtiéndose, por tanto, en problemas de modelado.

Sin embargo, los métodos libres de modelo se basan sólo en la experiencia

que obtienen de interactuar con el entorno. Esta interacción puede entenderse de varias formas. Por un lado, se pueden extraer secuencias completas de comportamiento desde una situación inicial a una final, de forma que el aprendizaje de las políticas se basa en estas secuencias. Por otro lado, la experiencia puede utilizarse paso a paso, es decir, para cada acción realizada por el agente. En el termino medio, estarían los sistemas que hacen las actualizaciones en función de trozos de esas secuencias. Los primeros, que se suelen agrupar en los denominados métodos Monte Carlo [Michie and Chambers, 1968], se aseguran el conocimiento final de una secuencia o intento de resolución del problema para realizar la asignación correcta del crédito o ganancia a cada una de las acciones que se han ido realizando. En el otro extremo, los denominados métodos de diferencia temporal (*TD methods*) [Sutton, 1988, Watkins, 1989], no requieren finalizar la secuencia con lo que utilizan la experiencia paso a paso. En el medio se sitúan los métodos TD(λ) [Watkins, 1989, Dayan, 1992, Jaakkola *et al.*, 1994], en los que la longitud de las secuencias de acciones que se utilizan para realizar las actualizaciones viene dado por el parámetro λ.

En cualquier caso, todos estos métodos se basan en las funciones de valor definidas anteriormente para representar las políticas de acción, y en la forma en que se realiza la interacción con el entorno. En este sentido, se suelen diferenciar también dos métodos. Los métodos basados en la política (*on-policy methods*) utilizan la política de acción que están aprendiendo para decidir la interacción que se realiza con el entorno. En los métodos fuera de política (*off-policy methods*) la política de interacción con el entorno es independiente de la política que se está aprendiendo en ese momento [Sutton and Barto, 1998]. Las ventajas de uno y otro dependen del problema que se trate en cada momento.

Métodos Monte Carlo

Como se ha introducido anteriormente, los métodos Monte Carlo [Michie and Chambers, 1968] se basan en la actualización de las funciones de valor mediante secuencias o intentos completos de resolución del problema por parte del agente en una interacción directa con el entorno. Dado que en principio el modelo del problema no es conocido, es necesario aprender la función de valor-acción en lugar de la función de valor-estado, dado que se debe estimar explícitamente el valor de cada acción con el fin de sugerir una política. Por tanto, el objetivo de los métodos Monte Carlo es estimar Q^*.

Uno de los principales algoritmos Monte Carlo es denominado Monte Carlo con arranque exploratorio o Monte Carlo ES (*"Monte Carlo with Exploring Starts"*) [Sutton and Barto, 1998] y se muestra en la Figura 2.4.

Monte Carlo ES

- Inicializar, para todo $s \in S$, $a \in A(s)$:
  - $Q(s,a) \leftarrow$valor arbitrario
  - $\pi(s) \leftarrow$ valor arbitrario
  - $ganancias(s,a) \leftarrow$ lista vacía
- Repetir para siempre
  1. Generar un episodio usando arranque exploratorio y π
  2. Para cada par (s,a) que aparece en el episodio
     - $R \leftarrow$ ganancia obtenida tras la primer ocurrencia del par (s,a)
     - Añadir R a $ganancias(s,a)$
     - $Q(s,a) \leftarrow promedio(ganancias(s,a))$
  3. Para cada s en el episodio
     $\pi(s) \leftarrow \arg\max_a Q(s,a)$

Figura 2.4: Algoritmo Monte Carlo ES.

En dicho algoritmo, se utiliza la lista $ganancias(s,a)$ para almacenar la lista de ganancias que se han ido obteniendo al ejecutar la acción a desde el estado s. Posteriormente, se utiliza el valor promedio de esta lista para actualizar la función Q. Esta lista es la figura clave de los métodos Monte Carlo, si bien existen métodos derivados que van calculando el valor promedio de las ganancias obtenidas sin necesidad de almacenar la lista completa [Sutton and Barto, 1998]. Como se puede observar, este algoritmo mantiene los dos pasos fundamentales de evaluación de la política: la actualización de la función Q y de mejora de la política; y la actualización de π presentada en el algoritmo de iteración de la política que se muestra en la Figura 2.2.

El arranque exploratorio significa que el primer par estado-acción de la secuencia se elige aleatoriamente, lo que asegura que todos los pares estado-acción son visitados infinitas veces, asegurando así la convergencia del algoritmo.

Métodos TD

Una de las principales aportaciones del aprendizaje por refuerzo es el aprendizaje por diferencia temporal (*Temporal-Difference Learning*). Estos métodos toman ideas tanto de la programación dinámica como de Monte Carlo: como en Monte Carlo, los métodos TD aprenden directamente de la experiencia, sin un modelo completo de la dinámica del entorno. Como en la programación dinámica, los métodos TD hacen estimaciones en función de otras estimaciones previamente aprendidas. El método TD más sencillo, TD(0) [Sutton, 1988], se muestra en la Figura 2.5, y se basa en el aprendizaje de la función de valor-estado. Al igual que ocurría con Monte Carlo, dado

Algoritmo TD(0)

- Inicializar $V(s)$ arbitrariamente
- Repetir (para cada episodio)
  - $a \leftarrow$ acción dada por π y s
  - Ejecutar la acción a y observar el refuerzo recibido, r, y el estado siguiente, s'
  - $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
  - $s \leftarrow s'$

  Hasta que s sea un estado terminal

Figura 2.5: Algoritmo TD(0).

que no se conoce la dinámica del modelo, es preferible aprender la función de valor-acción en vez de la función de valor-estado. Uno de los principales algoritmos desarrollados en esta línea es el algoritmo *Q-Learning* [Watkins, 1989, Watkins and Dayan, 1992], y que se muestra en la Figura 2.6.

En ambos algoritmos, tanto TD(0) como *Q-Learning*, el aprendizaje se realiza utilizando un parámetro de enfriamiento, α, que define la importancia que se da a las nuevas actualizaciones.

La Perspectiva Unificada: Métodos TD(λ) y Trazas de Eligibilidad

Entre los métodos TD, en los que la actualización de las funciones de valor se realiza en cada ejecución de la acción, y los métodos Monte Carlo,

Algoritmo Q-learning

- Inicializar $Q(s, a)$ arbitrariamente
- Repetir (para cada episodio)
  - Inicializar s
  - Repetir (para cada paso del episodio)
    - Seleccionar una acción a a partir de s usando una política derivada de Q
    - Ejecutar la acción a observando el refuerzo inmediato recibido, r, y el siguiente estado, s'
    - Actualizar la entrada de la tabla, $Q(s, a)$ con la ecuación:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.12}$$

    - $s \leftarrow s'$

    Hasta que s sea un estado final

Figura 2.6: Algoritmo *Q-learning*.

en los que esta actualización no se realiza hasta que no se termina una secuencia, existe un amplio rango de posibilidades, tal y como se muestra en la Figura 2.7, obtenida de [Sutton and Barto, 1998].

En la figura se muestra cómo los métodos TD tradicionales son métodos de un único paso, en el sentido de que cada vez que se realiza una acción y se llega a un estado nuevo, se realiza una actualización de las funciones de valor. En el lado opuesto, se encuentran los métodos Monte Carlo, donde las actualizaciones se realizan al final de un intento completo de solucionar el problema. En medio se encuentran los métodos TD de n pasos (*"n-step TD methods"*), en los que las actualizaciones se realizan en función de los resultados de ejecutar n acciones. Es decir, el refuerzo con el que se actualiza la función de valor estado en un instante de tiempo t, siguiendo un método TD de n pasos es tal y como se define en la ecuación 2.13:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}) \tag{2.13}$$

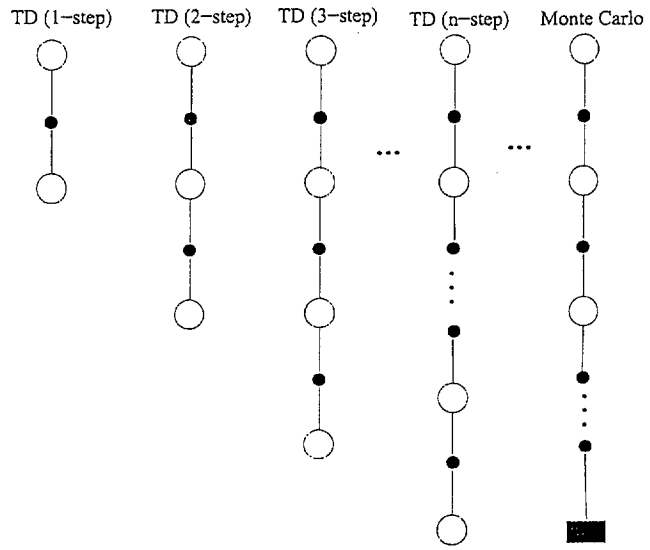


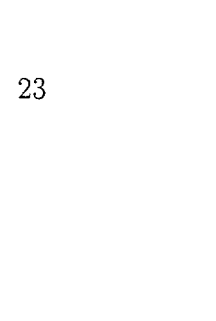
Figura 2.7: Espectro de posibilidades desde los TD tradicionales hasta Monte Carlo.

Cuando n es igual a 1, la función de actualización se convierte en la función típica de los métodos TD vistos anteriormente, mientras que si n es igual a la longitud de la secuencia completa, se convierte en la función de actualización de los métodos Monte Carlo.

Los métodos TD(λ) [Sutton, 1988] utilizan estas ideas, pero en vez de sumar directamente los refuerzos obtenidos en el futuro, los promedian con un factor λ^{n-1}, donde $0 \leq \lambda \leq 1$, tal y como se muestra en la Figura 2.8. El factor de normalización $(1 - \lambda)$ hace que el sumatorio completo sea 1.

La forma más sencilla de implementar esta aproximación se basa en las trazas de elegibilidad ("*eligibility traces*"). La traza de elegibilidad para cada estado, s, en cada instante, t, se denota con $e_t(s)$, y en cada paso, esta traza disminuye en un factor $\gamma\lambda$ para todos los estados, excepto para el que se acaba de visitar, que se incrementa en 1. De esta forma, la traza de elegibilidad actúa como una especie de memoria de los estados que han sido visitados recientemente, y de esa forma definen, para cada estado, cuánto es elegible para su actualización. El algoritmo TD(λ) es descrito en la Figura 2.9. Se observa que, si $\lambda = 0$, el algoritmo se convierte en un método TD, mientras que, si $\lambda = 1$, sería un método Monte Carlo.

Estas ideas planteadas para predecir la función de valor pueden ser adaptadas fácilmente a problemas de control, sin más que adaptarlo para aprender la función de valor estado-acción, $Q(s, a)$ en vez de la función de valor esta-



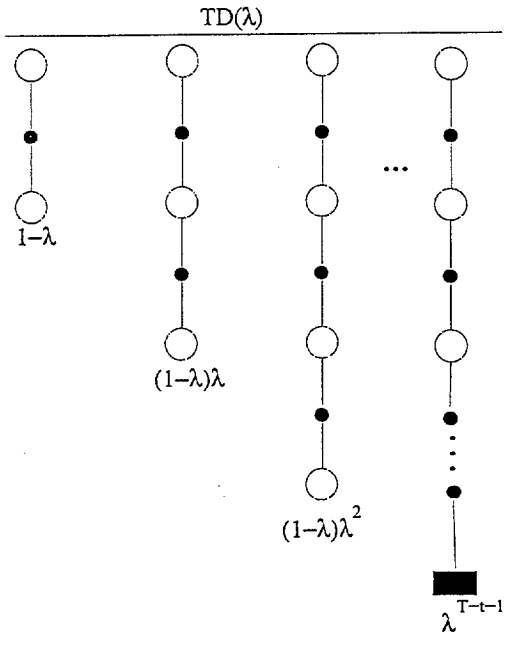


Figura 2.8: Diagrama de actualizaciones para TD(λ).

do, $V(s)$. Así existen los algoritmos *Sarsa(λ)* [Rummery and Niranjan, 1994, Rummery, 1995], o Q(λ) [Watkins, 1989], que adaptan los algoritmos Sarsa y Q-Learning para incluir el concepto de las trazas de elegibilidad.

2.3.3. Métodos Basados en el Modelo

En el apartado anterior se ha mostrado una alternativa a cómo resolver un problema de aprendizaje por refuerzo cuando no se conoce la dinámica del problema. La idea de esas técnicas era aprender las funciones de valor del dominio mediante la interacción del agente con el entorno, aprendiendo implícitamente esa dinámica desconocida. Estas técnicas suelen dar muy buenos resultados, aunque, a veces, esos resultados se obtienen a costa de una gran cantidad de experiencia. Por tanto, en aplicaciones donde la obtención de esta experiencia es extremadamente costosa, podrían no ser aconsejables.

En lugar de esta alternativa, existe la posibilidad de aprender la dinámica del problema mediante exploración, para así poder aplicar las técnicas de programación dinámica desarrolladas anteriormente. En este sentido surgen los métodos basados en el modelo, que se desglosan a continuación.

El método de Equivalencia de Certeza (*Certainty Equivalence Methods*) [Ku-

Algoritmo TD(λ)

- Inicializar V(s) arbitrariamente y e(s)=0, , para todo $s \in S$:
- Repetir para cada episodio
  1. $a \leftarrow$ una acción dada por la política π y el estado actual, s.
  2. Ejecutar la acción, a, observar el refuerzo, r, y el nuevo estado, s'.
  3. $\delta \leftarrow r + \gamma V(s') - V(s)$
  4. $e(s) \leftarrow e(s) + 1$
  5. Para todo s:
     - $V(s) \leftarrow V(s) + \alpha\delta e(s)$
     - $e(s) \leftarrow \gamma\lambda e(s)$
  6. $s \leftarrow s'$

  Hasta que s sea un estado terminal

Figura 2.9: Algoritmo TD(λ).

mar, 1986] es uno de los métodos más directos, consistente en aprender directamente la dinámica del problema (concretamente, las funciones T y R) mediante exploración, y calcular la política óptima usando uno de los métodos de programación dinámica. Sin embargo, este método tiene varias objeciones. Por un lado destaca el cómo realizar la exploración inicial [Whitehead, 1991]; por otro, cómo afrontar posibles cambios en el entorno.

El algoritmo *Dyna* [Sutton, 1990, Sutton, 1991], utiliza la experiencia simultáneamente para construir el modelo y para ajustar la política, además de utilizar el modelo para ajustar también la política. El ciclo de ejecución se basa, dada una tupla de experiencia completa, (s, a, s', r), en los siguientes pasos.

- Actualizar las estadísticas del modelo en función de la nueva tupla.
- Actualizar la política para el estado s, modificando la función de valor-acción de dicho estado y dicha acción, $Q(s, a)$.
- Realizar k actualizaciones más de dicha función de forma arbitraria,

basándose en el conocimiento del modelo.

- Elegir una nueva acción a ejecutar siguiendo una determinada política de exploración.

Existen muchos otros algoritmos que se basan en estas ideas, como el algoritmo *Prioritized Sweeping* de Moore [Moore and Atkeson, 1993], *Queue-Dyna* [Peng and Williams, 1993], *Real Time Dynamic Programming* [Barto *et al.*, 1995], etc. Estos algoritmos, aunque suelen realizar un buen aprovechamiento de la experiencia, suelen introducir inconvenientes, como un alto coste computacional.

2.4. Generalización en Aprendizaje por Refuerzo

En toda la discusión previa se ha asumido que es posible enumerar los espacios de estados y acciones y por tanto definir las tablas de valor asociadas a ellos. En otras palabras, se han asumido procesos de Markov finitos. En la Figura 2.10 se muestra un esquema sencillo de cómo a partir de un estado s, se puede obtener la acción a ejecutar, a partir de la tabla $Q(s,a)$, es decir, la política de acción, cuando el número de acciones posibles es L.

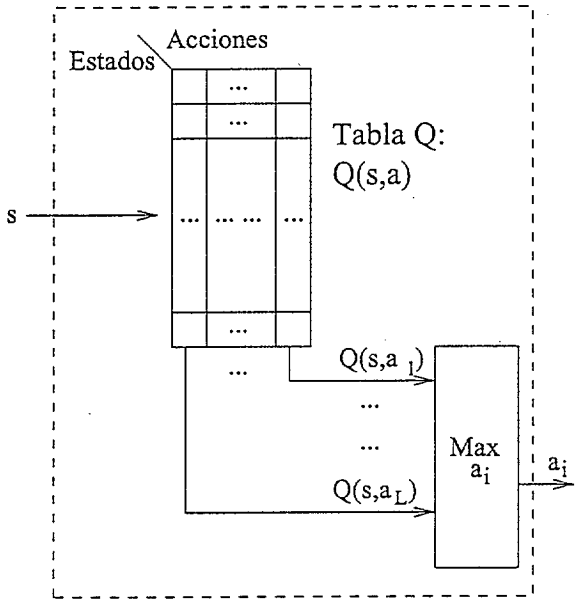


Figura 2.10: Representación tabular de la función $Q(s,a)$.

Sin embargo, en la mayoría de dominios esta suposición no es posible debido a dos causas fundamentales. Por un lado, en la mayoría de los dominios existen valores, atributos o parámetros de acciones continuos, lo que impide una enumeración de los estados o acciones que lo componen. Por otro lado, aunque los valores de estos atributos sean discretos, muchas veces el espacio es tan grande que es intratable en la práctica. Esta intratabilidad viene dada no tanto por los requisitos de memoria que se puedan derivar del almacenamiento de las tablas de valor, sino del uso ineficiente de la experiencia que se deriva del uso de dominios tan extensos.

Las técnicas de generalización engloban todos aquellos métodos que sirven para resolver todos los problemas previamente mencionados, y agrupan diversos conjuntos de técnicas que se repasarán brevemente en esta sección. Afortunadamente, la generalización desde ejemplos es un tema que ha sido ampliamente estudiado en otros dominios fuera del aprendizaje por refuerzo, por lo que muchas de las técnicas que se pueden aplicar no son más que adaptaciones de técnicas de aprendizaje supervisado. Este tipo de generalización que se requiere es a menudo denominada aproximación de funciones, dado que toma ejemplos de una función que se desea aprender (por ejemplo, una función de valor-estado) e intenta generalizar esos ejemplos para construir una aproximación completa de dicha función. La relación entre la generalización y la aproximación de funciones se amplia en la sección 2.4.1.

Uno de los métodos más habituales para realizar esta estimación son los métodos no lineales de descenso de gradiente, que suelen reflejarse en el uso de redes de neuronas como aproximadores, y que incluso, para algunos autores, merecen la redefinición del aprendizaje por refuerzo como programación dinámica neuronal [Bertsekas and Tsitsiklis, 1996]. En la sección 2.4.2 se introducen algunas de estas aproximaciones, junto con la problemática que introducen.

Por otro lado, otros autores han preferido basarse en otro tipo de aproximación, como la obtención eficiente de conjuntos reducidos de estados que permitan agrupar en regiones distintas grupos de estados. Estos métodos, que tratan de forma lineal la aproximación de la función de valor-acción, son tratados en las secciones 2.4.3 y 2.4.4.

2.4.1. Generalización y Aproximación de Funciones

En las técnicas vistas anteriormente, se solía comenzar con el problema de estimar la función de valor-estado, V^π, a partir de la experiencia generada utilizando una política π. Este es el paso de evaluación de la política del algoritmo de Iteración de la Política. La novedad que se introduce ahora es que la aproximación que se tiene en un instante t, $\hat{V}_t$, se representa no con una

tabla, sino como una función de un conjunto de parámetros, representados por un vector $\vec{\theta_t}$. Esto significa que el valor de la función $\hat{V}_t$ depende sólo del vector $\vec{\theta_t}$, variando en cada instante de tiempo en función de cómo varíe $\vec{\theta_t}$. Así, $\hat{V}_t$ podría ser una función implementada, por ejemplo, mediante una red neuronal, de forma que los parámetros $\vec{\theta_t}$ no serían más que los pesos de la red. Típicamente, la dimensión del vector de parámetros $\vec{\theta_t}$ es mucho menor que el número de estados, de forma que, cambiando sólo uno de los valores del vector de parámetros, se cambian los valores de la función para muchos estados. Consecuentemente, cuando se realiza una actualización para un estado, esa actualización es generalizada para muchos estados distintos.

Estas actualizaciones pueden denotarse por $s \rightarrow v$, refiriéndose al estado que se actualiza y el nuevo valor de la función V que se ha estimado. Mirando estas actualizaciones como ejemplos de entrenamiento, los problemas de aprendizaje por refuerzo se pueden reducir a típicos problemas de aproximación de funciones en aprendizaje supervisado, y que, por tanto, pueden ser tratados con cualquiera de las técnicas que existen, como redes neuronales, árboles de decisión, etc. Sin embargo, no todas las técnicas son válidas, dado que la mayoría de ellas presuponen un conjunto de entrenamiento estático sobre el que se realizan una o varias pasadas, mientras que aquí se dispone de información generada de forma interactiva con un entorno, que puede ser dinámico. Por tanto, se requieren métodos que funcionen bien con información adquirida de forma incremental.

Otra característica importante es que, normalmente, lo que se intenta aproximar no es la función de valor-estado, $V(s)$, sino la función de valor-acción, $Q(s,a)$. Ésta función incluye un parámetro más, que es la acción. En este punto existen dos alternativas. La primera es mantener la acción como un parámetro de la función, de forma que el tipo de actualizaciones que se producen ahora son del tipo $s, a \rightarrow q$. Esto produce, por un lado, que se incremente la dimensión de los datos de entrada, complicando, por tanto, la correcta aproximación de la función de valor. Sin embargo, el principal inconveniente es cómo a partir de ese aproximador se puede obtener correctamente la política de acción, ya que eso supondría consultar el valor de Q para todos los posibles valores de las acciones. Debido a estos inconvenientes, si se dispone de un conjunto reducido de acciones, por ejemplo, L, se suelen generar tantos aproximadores como posibles acciones haya, es decir $\hat{Q}_{a_i}(s), i = 1, \ldots, L$. De esta forma, la obtención de la política a partir de los distintos aproximadores es más sencilla. En la Figura 2.11 se muestra un esquema de este tipo, en el que se supone que se ha utilizado una red neuronal para aproximar cada $\hat{Q}_{a_i}(s)$.

Existen muchos trabajos en esta línea, entre los que destacan CMAC para *Q-Learning* [Watkins, 1989], redes de retro-propagación ("*backpropagation*")

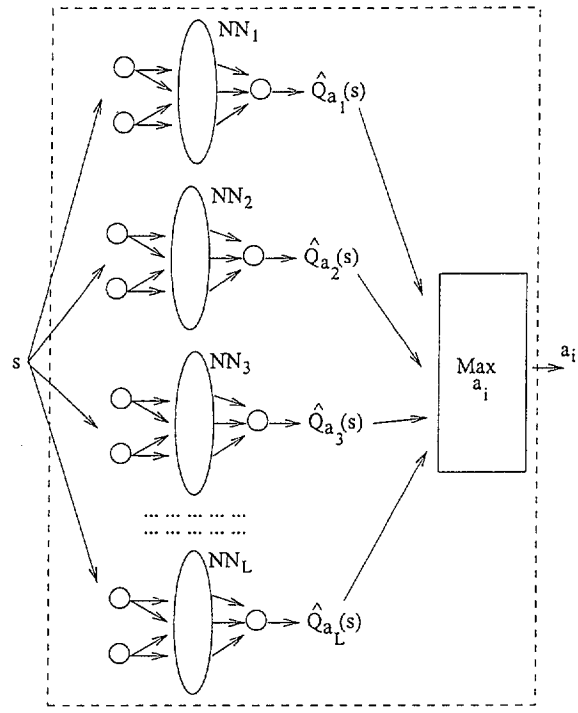


Figura 2.11: Aproximación con redes neuronales de la función $Q(s, a)$.

para *Q-Learning* en [Lin, 1991], redes de retro-propagación para aprender la función de valor en *backgammon* [Tesauro, 1992], etc.

2.4.2. Métodos No Lineales de Estimación

Cuando se establece un método de aproximación de funciones, puede ser necesario plantear cuál es la medida que nos permite decidir si la aproximación obtenida es buena o mala. La mayoría de los métodos supervisados utilizan para esto el error cuadrático medio (ECM) sobre una distribución cualquiera, P, de las entradas. En el problema de aproximación funcional en aprendizaje por refuerzo, esas entradas consisten en los estados, mientras que la función objetivo que se quiere aproximar puede ser $V^{\pi}(s)$. Por tanto, el ECM para una aproximación de la función de valor-estado en el instante t, $\hat{V}_t(s)$, utilizando los parámetros $\vec{\theta_t}$, se define como muestra la ecuación 2.14.

$$ECM(\vec{\theta_t}) = \sum P(s)[V^{\pi}(s) - \hat{V}_t(s)]^2 \qquad (2.14)$$

El objetivo de cualquier aproximación sería, por tanto, encontrar el conjunto de parámetros $\vec{\theta_t}$ que minimicen el ECM. O dicho de otra forma, buscar

el punto mínimo de la función $ECM(\vec{\theta_t})$. El objetivo de los métodos de descenso de gradiente es, precisamente, buscar un $\vec{\theta_t}$ que minimice esa función, realizando, ante cualquier nuevo ejemplo, s_t, pequeñas modificaciones en el vector de parámetros, $\vec{\theta_t}$, en el sentido que reduzca más el error para ese ejemplo, tal y como muestra la ecuación 2.15.

$$\vec{\theta_{t+1}} = \vec{\theta_t} + \alpha[V^{\pi}(s_t) - \hat{V}_t(s_t)]\nabla_{\vec{\theta_t}}\hat{V}_t(s_t) \tag{2.15}$$

donde $\nabla_{\vec{\theta_t}}\hat{V}_t(s_t)$ es el gradiente de $\hat{V}_t(s_t)$ con respecto a $\vec{\theta_t}$. Estos métodos se denominan de descenso de gradiente porque las modificaciones realizadas sobre el vector de parámetros, $\vec{\theta_t}$, son proporcionales al gradiente negativo del error cuadrático para ese ejemplo, que es el camino más rápido para minimizar el error.

El descenso de gradiente tiene, sin embargo, algunos inconvenientes. El primero de ellos es cómo definir el vector de parámetros $\vec{\theta_t}$. En una implementación de redes de neuronas, la definición de estos parámetros está ligado a definir la arquitectura de la red, e incluso el método de aprendizaje de los pesos [Yao, 1993]. Esta definición es muy complicada en la mayoría de los sistemas a modelar, y más en un entorno tan dinámico como el del aprendizaje por refuerzo.

El segundo gran inconveniente, es que en la ecuación 2.15 se presupone el conocimiento "a priori" de $V^{\pi}(s_t)$, es decir, presupone el conocimiento de la función de valor que se está aproximando para el estado s_t. Esto, que en los problemas de aprendizaje supervisado se cumple siempre, no se cumple para el aprendizaje por refuerzo. Esto lleva en muchas ocasiones a que no se consiga la convergencia del algoritmo, como se describe en [Boyan and Moore, 1995]. En este artículo se presenta el algoritmo de *Smooth Value Iteration* en el que se plantea una versión del *Value Iteration*, pero en el que se sustituye la tabla de todos los estados por un aproximador suave de la función de valor, $\hat{V}$, tal y como muestra la Figura 2.12. El aproximador es aprendido de forma iterativa, generado tantas aproximaciones $\hat{V}^{iter}$ como iteraciones de el algoritmo. En cada una de estas iteraciones, el aproximador $\hat{V}^{iter}$ es recalculado utilizando como datos de entrenamiento los estados de ejemplo, $x_i \in \hat{S}$, cada uno de ellos etiquetado con el nuevo coste, $v^{iter}[i]$, calculado a partir de la aproximación de la función de valor aprendida en la iteración anterior.

Como se observa, este algoritmo se planteó como una aproximación de coste a meta, donde no se intenta maximizar la suma de los refuerzos alcanzados, sino minimizar el coste en alcanzar la meta. Se observa que el algoritmo va aprendiendo la función de valor propagando los costes desde la meta (coste 0) hacia el resto del entorno, aprendiendo en cada iteración el

Smooth Value Iteration

- Dados
  - Una colección de estados $\hat{S} = \{x_1, \ldots, x_N\}$, obtenidos del espacio de estados total S y una zona de meta $G \subset S$
  - Un conjunto de acciones finito, A
  - Una función de transición de estados determinista, $T : S \times A \rightarrow S$
  - Una función de coste $Cost : S \times A \rightarrow R$
  - Un aproximador de la función de valor: $\hat{V}^0 : S \rightarrow R$
- Hacer $iter = 0$
- $v^0[i] = 0$, $\forall i = 1, \ldots, N$
- Repetir
  - Entrenar $\hat{V}^{iter}$ para aproximar el conjunto de entrenamiento formado por: $< x_1, v^{iter}[1] >, \ldots, < x_N, v^{iter}[N] >$
  - Hacer $iter = iter + 1$
  - Para i=1 hasta N hacer

$$v^{iter}[i] = \begin{cases} 0 & \text{si } x_i \in G \\ \text{mín}_{a \in A}\{Cost(x_i, a) + \hat{V}^{iter-1}(T(x_i, a))\} & \text{en otro caso} \end{cases}$$

  Hasta que el vector v no cambie
- Devolver $\hat{V}^{iter-1}$

Figura 2.12: Algoritmo de *Smooth Value Iteration*.

coste a meta de los estados situados a una distancia de una acción de los estados para los cuales se había aprendido el coste en la iteración anterior.

En [Boyan and Moore, 1995] se planteaba que dependiendo del aproximador utilizado, el algoritmo convergía a la función de valor y política óptimas, otras veces sólo a la política óptima, otras convergía pero a políticas y funciones de valor no óptimas, y otras veces no convergía. Para evitar estos problemas, se planteaba el algoritmo de *Grow-Support*, que sólo mantenía

en el vector v aquellos estados para los cuales se aseguraba que siguiendo la política avariciosa aprendida hasta el momento, se podía llegar a la meta con un coste menor que infinito.

De este algoritmo se puede obtener fácilmente una versión libre de modelo sin más que aproximar, en vez de la función de valor V, la función $Q(s,a)$, o si suponemos que existen L acciones posibles $a_1,\ldots,a_L$, L aproximadores $\hat{Q}_{a_i}(s)$, para $i = 1\ldots,L$. Además, dado que es una versión libre de modelo, en lugar de partir de una colección de estados, se debe partir de una colección de tuplas de experiencia, y utilizar una función de actualización de los aproximadores $\hat{Q}_{a_i}(s)$ adecuada, por ejemplo, *Q-Learning*.

El algoritmo queda detallado en la Figura 2.13. La principal diferencia con la anterior versión es que, dado que se requieren L aproximadores $\hat{Q}_{a_i}(s)$, se requieren L conjuntos de entrenamiento en cada iteración, cada uno de ellos con las tuplas de experiencia que contienen la acción correspondiente.

El problema de generalización es un problema que crece según aumenta el número de variables que definen cada estado. Concretamente, el tamaño del espacio de estados crece de forma exponencial con el número de atributos que definen dichos estados. Esto hace que pueda ser necesario buscar representaciones más compactas de dichos estados, de forma que aunque el número de atributos que definen cada estado crezca de forma exponencial, no lo haga de igual forma el tamaño del espacio de características. Esta aproximación tiene como uno de sus principales métodos a los de discretización del espacio de estados que se plantean en la siguiente sección. No obstante, la extracción de características es especialmente útil cuando la aproximación de la función de valor a partir de los estados originales es no lineal, pero al obtener un espacio de características, estas características sí permiten una aproximación lineal [Bertsekas and Tsitsiklis, 1996].

Existen otros métodos típicos del aprendizaje supervisado que pueden ser utilizados para aproximar la función de valor. Por ejemplo, el aprendizaje basado en instancias o aprendizaje perezoso [Aha, 1997] tiene como principal característica el mantener o memorizar toda o parte de la experiencia que se obtiene, de forma que sólo es utilizada o tratada a la hora de realizar una consulta o decisión. Las aproximaciones basadas en instancias siguen la aproximación del vecino más cercano, almacenando todo o parte del conocimiento, y reutilizándolo cada vez que debe tomar una decisión. En [Parker and Touzet, 2000] se plantea un método de actualización de los valores de la función Q denominado *Pessimistic Lazy Q-Learning*. Con este método la actualización del valor de cada estado o situación en la memoria se realiza teniendo en cuenta una medida de similitud. De esta forma, para una nueva situación, se actualizan un conjunto fijo de situaciones similares obtenidas del total de la memoria, realizando siempre la actualización más pesimista

Iterative Smooth Q-Learning

- Dados
  - Un espacio de estados S y una zona de meta $G \subset S$
  - Un conjunto de acciones finito, A, de tamaño L, $A = \{a_1, \ldots, a_L\}$
  - Una colección de tuplas de experiencia del tipo $< s, a_i, s' >$, donde $s \in S$ es un estado desde el que se ejecuta la acción a_i, y s' es el estado al que se llega
  - Una función de coste $Cost : S \times A \rightarrow R$
  - L aproximadores de la función de valor-acción: $\hat{Q}_{a_i} : S \rightarrow R$
- Hacer $iter = 0$
- $q^0_{a_i}[j] = 0, \forall j = 1, \ldots, N,\ a_i \in A$
- Repetir
  - Entrenar $\hat{Q}^{iter}_{a_i}$ para aproximar el conjunto de entrenamiento formado por: $< s_1, q^{iter}_{a_i}[1] >, \ldots, < s_N, q^{iter}_{a_i}[N] >$
  - Hacer $iter = iter + 1$
  - Para j=1 hasta N, y para todo $a_i \in A$ hacer

$$q^{iter}_{a_i}[j] = \begin{cases} 0 & \text{si } s'_j \in G \\ \text{mín}_{a_r \in A}(Cost(s_j, a_j) + \hat{Q}^{iter-1}_{a_r}(s'_j)) & \text{en otro caso} \end{cases}$$

  Hasta que el vector q no cambie
- Devolver $\hat{Q}^{iter-1}_{a_i}$, para $i = 1, \ldots, L$

Figura 2.13: Algoritmo de *Iterative Smooth Q-Learning*.

de entre todas las situaciones similares.

En [Smart and Kaelbling, 2000, Smart, 2002], también se realiza una aproximación basada en instancias. Sin embargo, como aproximador se utiliza un método de regresión lineal denominado regresión ponderada localmente (*Locally Weighted Regression*) o LWR, que pondera la importancia de las instancias almacenadas para hacer la regresión en función de lo cercanas que

están a la consulta que se está realizando. Para optimizar el acceso a los datos más cercanos, organiza los ejemplos de entrenamiento en árboles *kd*.

2.4.3. Modelos de Representación del Espacio de Estados

Una de las aproximaciones típicas para representar el espacio de estados viene dada por la transformación de los estados en características, es decir, atributos derivados que se consideran relevantes para la tarea que se está llevando a cabo. Eligiendo estas características apropiadas para la tarea es una forma de introducir conocimiento "a priori" en el sistema de aprendizaje. Además, en general, se suelen necesitar características derivadas de otras que se reciben más directamente del entorno [Bertsekas and Tsitsiklis, 1996]. Esta representación en características deriva en cómo transformar espacios de estados continuos e infinitos en conjuntos finitos y de un tamaño mucho más limitado. Es decir, el vector de parámetros $\vec{\theta_t}$ se convierte en un vector de parámetros que define las regiones del espacio de estados original. La Figura 2.14 muestra cómo, a partir de una discretización del espacio de estados, se puede utilizar una tabla para aproximar la función de valor-acción, Q.

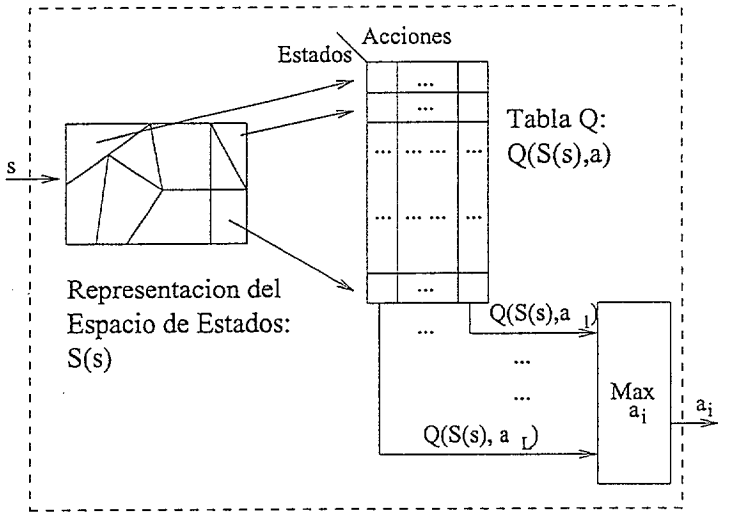


Figura 2.14: Representación de la función $Q(s, a)$ basada en discretización.

A continuación se repasan algunos de los modelos de discretización de los espacios de estados más extendidos.

Codificación Gruesa (*"Coarse Coding"*)

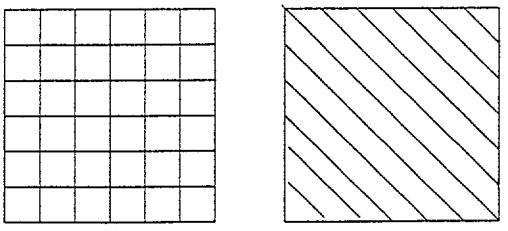
Uno de los métodos que se pueden utilizar es la denominada *codificación gruesa* (*coarse coding*) [Hinton, 1984]. Supongamos un ejemplo de tarea en el que el conjunto de estados es continuo y bidimensional. En este caso, un tipo de característica podría definirse como la pertenencia o no a un círculo, de entre un conjunto de N círculos que pueden o no estar superpuestos. Si el estado está dentro de un círculo, entonces la característica asociada a ese círculo toma valor 1, y si no, toma el valor 0. Son, por tanto, características binarias. De esta forma, a partir del vector de características se puede dar una localización *gruesa* de dónde está el estado al que representa. A aquellas técnicas que representan cada estado con características que se pueden superponer de esta forma, aunque las características no se basen en círculos ni sean binarias, se le denomina *codificación gruesa.* En este caso, y siguiendo la técnica de aproximación de funciones por descenso de gradiente, se pasa de un conjunto continuo bidimensional a un problema de aproximación de funciones, donde el vector $\vec{\theta}_t$ tiene N componentes que son afectados por el aprendizaje. Así, si estamos entrenando para un punto $\vec{x}$, entonces los parámetros de todos los círculos que contienen ese punto serán afectados, influyendo más en aquellas regiones que contienen al punto y en las que intersectan varios círculos. Por tanto, el tamaño, la densidad y la forma de los círculos influyen en cómo generaliza el problema.

Codificación en Teja (*"Tile Coding"*)

Otra codificación similar viene dada por la codificación en teja (*tile coding*) [Albus, 1971, Albus, 1981]. En esta codificación, el espacio de características se divide en particiones o *tejas.* En la Figura 2.15 se muestran varios ejemplos de particiones del espacio de características, desde una totalmente uniforme, a particiones irregulares, diagonales, etc. Nuevamente, la forma, tamaño y densidad de estas regiones determina la capacidad de generalización.

Funciones de Base Radial

Las funciones de base radial (RBFs) [Broomhead and Lowe, 1988] son la generalización natural de la codificación gruesa a características con valores continuos. Así, en lugar de que cada característica tome el valor 0 ó 1, puede tomar cualquier valor en el intervalo [0, 1], reflejando grados de pertenencia de la característica. Una función de base radial típica es la gaussiana, de forma que la posesión o no de una característica, i, en un estado, s, es una función, $\phi_s(i)$, del centro de la gaussiana, c_i, y de su varianza, σ_i, tal y como



(a) Regular (b) Diagonal

Figura 2.15: Ejemplos de codificación en teja.

muestra la ecuación 2.16:

$$\phi_s(i) = e^{-\frac{\|s-c_i^2\|}{2\sigma_i^2}} \tag{2.16}$$

Las redes de base radial son funciones de aproximación lineal que utilizan funciones de base radial para definir sus características. Usando los métodos de descenso de gradiente definidos anteriormente, pueden usarse para aprender la función de valor de cualquier problema.

El Vecino más Cercano

La regla del vecino más cercano [Duda and Hart, 1973, Gersho and Gray, 1992] proporciona una forma sencilla de dividir un espacio de estados en regiones, tal y como muestra la figura 2.16. Al igual que con las funciones de base radial, cada punto del espacio de estados continuo puede aproximarse mediante una medida de distancia a uno de entre un conjunto de n prototipos, definiendo de forma sencilla las distintas regiones en las que se divide el espacio de estados. Se observa en la figura que el resultado es un método sencillo de representación de una codificación en teja irregular.

Basado en Árboles de Decisión

Los árboles de decisión [Quinlan, 1986] también han sido utilizado ampliamente en la literatura de aprendizaje por refuerzo. No obstante, los árboles pueden ser construidos y/o utilizados de dos formas distintas. Una primera aproximación consiste en utilizarlos como aproximadores de funciones puros, incluidos como tales en algoritmos como *Smooth Value Iteration*, descrito anteriormente, utilizando métodos generales para la construcción del árbol, como C4.5 [Quinlan, 1993]. Otra posibilidad es que la construcción del árbol esté adaptada al problema de aproximar la función de valor [Chapman and

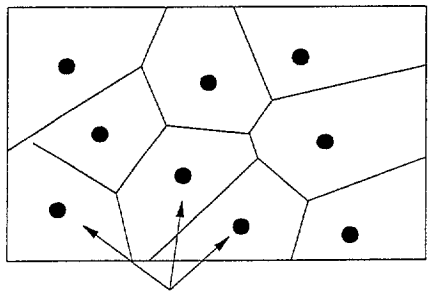


Figura 2.16: Ejemplos de codificación del vecino más cercano.

Kaelbling, 1991, Moore and Atkeson, 1995, Reynolds, 2000], tal y como se mostrará posteriormente en los métodos de discretización de resolución variable. En cualquier caso, los árboles de decisión pueden ser entendidos como una nueva forma de discretizar el espacio de estados, ya que cada hoja puede representar una de estas regiones.

Codificación *Kanerva*

Cuando la dimensión del espacio de estados se hace muy grande, del número de cientos de dimensiones, las aproximaciones vistas anteriormente se hacen impracticables. En esos casos, el tamaño del problema crece de forma exponencial con el número de dimensiones, imposibilitando su resolución. Sin embargo, parece lógico pensar que aunque el tamaño del problema crezca de forma exponencial con el número de dimensiones, su dificultad no tiene por qué incrementarse también de forma exponencial. Por ejemplo, el aumento en las dimensiones puede ser debido a la inclusión de atributos poco relevantes y que por tanto, no introducen complejidad. Por tanto, se requieren métodos cuya complejidad no se vea afectada por el incremento de la dimensión *per se*, sino métodos que estén limitados sólo por el problema que están aproximando. En este sentido se introduce lo que en [Sutton and Barto, 1998] se denomina codificación *Kanerva* (*Kanerva coding*) [Kanerva, 1988], que elige características binarias que corresponden con la cercanía o lejanía de los ejemplos de entrenamiento a determinados prototipos, del mismo estilo que los definidos en los modelos del vecino más cercano. Por tanto, se considera una representación del espacio de características binario, permitiendo utilizar la distancia de *hamming* (número de bits en los que dos vectores de características difieren) como medida de distancia entre estos vectores de características. La ventaja es, por tanto, que la complejidad de las funciones

que pueden ser aprendidas dependen del número de características, que no depende estrictamente del número de dimensiones de la tarea.

2.4.4. Aprendizaje de la Representación del Espacio de Estados

Anteriormente se han visto diversos modelos que permiten representar de una forma reducida el espacio de estados con el objetivo de realizar una generalización desde esa representación reducida a la original. Estas representaciones, sin embargo, introducen nuevos problemas. Por ejemplo, en *codificación gruesa* se introducía que la forma, tamaño y densidad de los círculos influía en cómo generalizaba. De igual forma, en *codificación en teja*, el tamaño y la forma de las tejas también influía en la capacidad de generalización. Cuando se utilizan redes de base radial, el problema es definir la arquitectura de la red, es decir, cuántas neuronas se utilizan, y cuáles son los parámetros (centros y varianzas) de las funciones de base radial. Por último, si se sigue el método del vecino más cercano, hay que definir eficientemente el número de prototipos, así como sus posiciones. En este apartado se definen algunos métodos que se han aplicado hasta ahora para resolver el problema planteado.

Modelos de Discretización de Resolución Variable

Los modelos de discretización de resolución variable tratan de atacar el problema de la densidad (o resolución) de los modelos de generalización. Por ejemplo, el algoritmo *Partigame* [Moore and Atkeson, 1995] divide el espacio de estados en regiones o tejas. En cada región, las acciones disponibles consisten en cómo llegar a regiones vecinas. A partir de ahí se puede construir un grafo de transiciones de estado, que puede ser resuelto por caminos mínimos. Además, se utiliza un criterio *minimax* para determinar cuándo un grupo de celdas es demasiado grueso como para prevenir movimientos entre obstáculos o evitar ciclos. Estos grupos de celdas pueden ser divididos o refinados, obteniendo una mayor resolución de esa zona del espacio de estados.

En [Munos and Moore, 2002] se definen distintos mecanismos para decidir cuáles son las regiones que se deben dividir, mientras que en [Reynolds, 2000] se aplican todas estas ideas en métodos de aprendizaje por refuerzo libres de modelo. En la mayoría de estos métodos, se utilizan *árboles-kd* para representar el espacio de estados.

Modelos de Representación Dinámica

Otra posibilidad, muy relacionada con la anterior, consiste en que las regiones que se definen pueden aumentar o disminuir de tamaño, así como desplazarse, con el fin de obtener una representación óptima del espacio de estados. Uno de los métodos más sencillos para hacer esto se basa en los métodos del vecino más cercano, tal y como se definió en la sección 2.4.3. En [Claussen *et al.*, 1999], se utiliza el algoritmo de aprendizaje supervisado LVQ [Kohonen, 1984] para dividir el espacio de estados en regiones definidas por prototipos, y el algoritmo *Q-Learning* para calcular la función de valor-acción. Cada estado o región viene representado por una neurona del mapa, de forma que tanto el mapa como la función de valor se van calculando de forma incremental. A medida que se va aprendiendo el mapa, las regiones van cambiando su posición, forma y tamaño, de forma que al final dichas regiones representen de forma eficiente el espacio de estados. En esta línea, se encuentran otros trabajos como [Touzet, 1997], donde los mapas auto-organizativos de Kohonen [Kohonen, 1995] se utilizan como memoria asociativa para aprender tuplas de estado, acción y función de valor.

Métodos Jerárquicos

Otro método para afrontar espacios de estados de gran tamaño es tratarlos como una jerarquía de problemas. Por ejemplo *Feudal Q-Learning* [Dayan and Hinton, 1993] utiliza una jerarquía de módulos de aprendizaje. En el caso más simple, existe un maestro a alto nivel, y varios esclavos a bajo nivel. El maestro recibe los refuerzos desde el entorno, y sus acciones consisten en enviar comandos a sus esclavos. El maestro está encargado de reforzar los comportamientos de los esclavos dependiendo de si las acciones que toman satisfacen el comando que les envió. El maestro, entonces, aprende una aplicación de estados a comandos, mientras que los esclavos aprenden una aplicación de comandos a acciones en el entorno.

En [Dietterich, 2000] se presenta otro método basado en la descomposición del MDP objetivo en varios MDPs más pequeños, de forma que la función de valor objetivo también puede dividirse en otras funciones de valor, también más pequeñas.

2.4.5. Generalización de Acciones

Todos los modelos vistos anteriormente generalizan sobre el espacio de estados. Sin embargo, hay veces que el conjunto de acciones es demasiado grande, e incluso continuo, planteándose la misma problemática que con los

estados. Por ejemplo, si se utiliza una red neuronal para estimar Q, se puede utilizar una red para cada acción, o una red con distintas salidas, una para cada acción. Con espacios de acciones continuos, ambas aproximaciones son imposibles. Una alternativa sencilla es usar una red con el estado y la acción como entrada, y el valor de Q como salida. El entrenamiento, en principio, puede no ser muy complicado. Sí parece más complicado utilizar la red para encontrar la política óptima.

En muchos casos, sin embargo, se opta por limitar los conjuntos de acciones a subconjuntos representativos, o incluso a conjuntos de macro-acciones más elaboradas, eliminando de forma sencilla este problema. No obstante, quedaría evaluar cómo afectan estas representaciones a la resolución del problema. Algunos ejemplos de generalización sobre el espacio de acciones son [Gullapalli, 1992, Baird and Klopf, 1993].

2.4.6. Generalización en Robótica

El aprendizaje por refuerzo no sólo es directamente aplicable a la mayoría de las tareas de control de robots, sino que también es donde más se ha utilizado. Sus aplicaciones van desde el aprendizaje de las habilidades individuales más simples, hasta las más complejas, cooperativas, y que además pueden estar separadas en capas de comportamiento [Stone, 2000].

Los dominios de robots suelen incluir serias limitaciones a los métodos de aprendizaje que se han desarrollado en todo este trabajo. Dentro de los temas que se han tratado, la primera característica a tener en cuenta es que habitualmente, el modelo de comportamiento del robot sobre el entorno es desconocido, lo que elimina directamente el uso de técnicas de programación dinámica por sí solas. Otra característica es que normalmente la interacción con el entorno es un proceso costoso, tanto en tiempo, como en el daño que pueden sufrir los robots durante el aprendizaje. La primera opción ante este problema es el uso de simuladores, pero la obtención de simuladores que imiten con cierta precisión a sistemas de robots no es una tarea fácil. Esto obliga a que las técnicas a utilizar requieran el tiempo mínimo. Como hemos visto, las técnicas basadas en el modelo requieren una exploración inicial muy fuerte, por lo que, aunque sin descartarlas, no parecen las más adecuadas. Esto limita el conjunto de soluciones a los métodos libres de modelo. Además, estos dominios suelen ser muy ruidosos, introduciendo un alto grado de indeterminismo, tanto en la información que se recibe del entorno a través de los sensores, como en las acciones ejecutadas por los actuadores. Por último, en general son dominios continuos, en los que la información recibida por los sensores, así como de las posibles acciones a ejecutar, producen espacios de estados y de acciones continuos o de gran tamaño, requiriendo técnicas de

generalización adecuadas.

Algunas de las técnicas que podrían ser aplicables serían, por tanto los métodos de discretización de resolución variable [Moore, 1991, Moore and Atkeson, 1995], definidos en la sección 2.4.4, pero ya se comentó entonces que estos métodos suelen plantearse en dominios deterministas, por lo que deberían ser descartados, o, al menos, adaptados.

Algunos de los trabajos tratan de buscar codificaciones gruesas del espacio de estados como técnica de generalización de la entrada. Por ejemplo, en [Asada *et al.*, 1996] se utilizan hiper-elipsoides para agrupar vectores de entradas sensoriales para aprender mapas de transición de estados en términos de acciones mediante las cuales se pueden obtener secuencias de acciones óptimas. En [Ueno *et al.*, 1996], se aprenden y mantienen situaciones en el espacio de estados continuo agrupados por los refuerzos recibidos desde el entorno, aprendiendo las transiciones entre las distintas situaciones con el fin de aplicar planificación parcial sobre la red aprendida. En [Sawada *et al.*, 1999] se presenta otra heurística para partir el espacio de estados, basándose en los refuerzos. En general, todas estas técnicas han sido aplicadas en dominios de tamaño pequeño, típicamente con sólo dos o tres atributos para representar el espacio de estados.

Dentro del área de la robótica, cabe destacar también los dominios multi-robot inherentemente cooperativos, en los que la asignación de crédito es compleja, ya que el resultado de las acciones puede depender, no sólo de las acciones tomadas por un robot, sino de las ejecutadas por otros robots [Parker and Touzet, 2000].

2.5. Aprendizaje del Modelo y Generalización

El problema de la generalización introduce un nuevo apartado en la clasificación de las técnicas de aprendizaje por refuerzo que se dio al principio. Estas técnicas se dividían entre aquéllas que conocían la dinámica del modelo, y sobre las cuales se podía aplicar programación dinámica (DP), y aquéllas en las que no se conocía dicha dinámica. En ambos casos se presuponía el conocimiento del espacio de estados. Sin embargo, como hemos visto, algunas técnicas de generalización se basan en redefinir este espacio de estados, introduciendo un nuevo problema. Este problema podría solucionarse siguiendo tres líneas derivadas de las anteriores. Por un lado, aprender el conjunto de estados, con lo que estaríamos en el punto de partida de las técnicas anteriores. Por otro lado, se podría intentar aprender el conjunto de estados a la

vez que la dinámica del problema, con lo que estaríamos en técnicas basadas en el modelo. Por último, se podría aprender todo a la vez. La Figura 2.17 muestra la relación entre todos estos métodos.

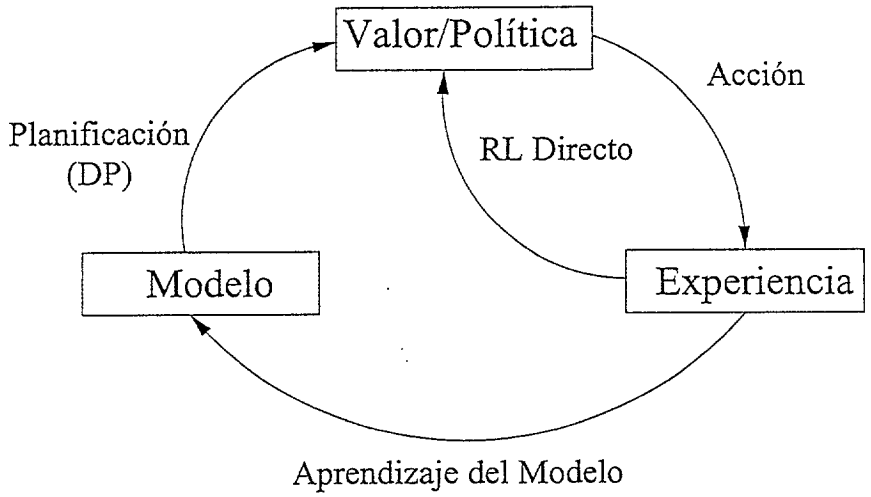


Figura 2.17: Relación entre Planificación, Aprendizaje y Acción.

En la figura se introducen tres módulos fundamentales, extraídos de [Sutton and Barto, 1998], donde se relacionan aprendizaje, planificación y acción. En primer lugar, están las funciones de valor y/o políticas, como elemento común de todos los algoritmos vistos hasta ahora. A partir de esta política se ejecutan acciones que generan una experiencia de la interacción con el entorno. Esta experiencia, que constituye el segundo módulo, puede utilizarse para aprender la política mediante técnicas de aprendizaje por refuerzo directo, es decir, los métodos libres de modelo, o para aprender el modelo. Este modelo constituye el tercer módulo y puede utilizarse en técnicas de programación dinámica para obtener la política óptima.

Sin embargo, este diagrama no permite incluir totalmente las cuestiones derivadas de las técnicas de generalización revisadas en este trabajo, ya que muchas de éstas dividen el conocimiento que se puede tener y utilizar del modelo: por un lado está el conjunto de estados de los que se compone; por otro lado, su dinámica. De esta forma, el aprendizaje del modelo puede seguir dos líneas distintas. Si se conoce el conjunto de estados, se puede pasar directamente a aprender la dinámica. Si no se conoce el conjunto de estados, o se conoce pero se quiere rediseñar por cuestiones de generalización, se puede aprender este conjunto de estados. Una vez que se conoce este conjunto, se puede optar por aplicar una técnica de aprendizaje por refuerzo directo,

en las que, como se ha mostrado anteriormente, no se conoce la dinámica, o aprender esta dinámica para resolver el problema con programación dinámica. El esquema quedaría tal y como se muestra en la Figura 2.18.

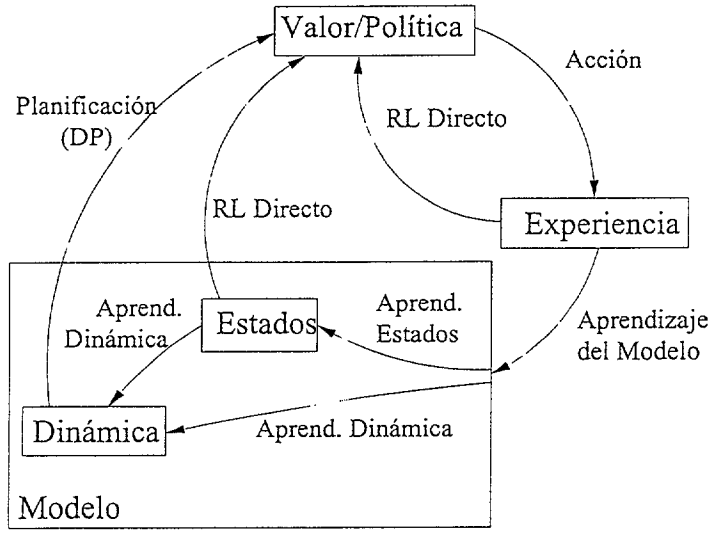


Figura 2.18: Relación entre Planificación, Aprendizaje y Acción Extendido.

En este esquema entran gran cantidad de modelos vistos anteriormente. Por ejemplo, los modelos de resolución adaptativa, como VRDP [Moore, 1991], pasan tanto por el aprendizaje de los conjuntos de estados como de la dinámica para luego aplicar programación dinámica en el estudio de la función de valor y la política. En [Claussen *et al.*, 1999] se limita el aprendizaje del modelo al aprendizaje del conjunto de estados, para luego utilizar una técnica directa de aprendizaje por refuerzo, como *Q-Learning*. En este caso el aprendizaje del conjunto de estados se realiza de forma incremental con el aprendizaje de la política, mediante un algoritmo que integra *Q-Learning* con los mapas auto-organizativos de Kohonen.

En el modelo de la Figura 2.18 se ha supuesto que el conjunto de las acciones que puede ejecutar el agente está definido, pero también se podría incluir la generalización de acciones con un aprendizaje de dichas acciones, que podría ser paralelo al aprendizaje de los estados, ya que ambos son muy dependientes. Por tanto, el modelo se compondría de la parte dinámica definida anteriormente, más una parte estática que incluye tanto los conjuntos de acciones como los de estados, tal y como se muestra en la Figura 2.19.

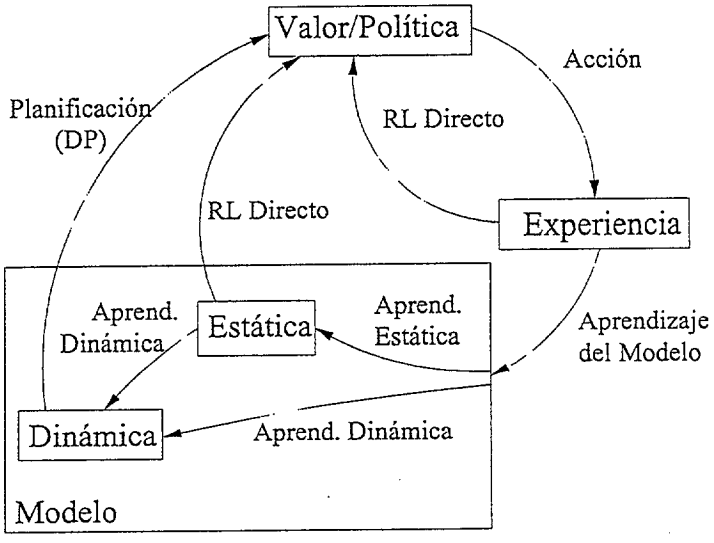


Figura 2.19: Relación entre Planificación, Aprendizaje y Acción Definitivo.

2.6. Problemas Comunes

Los métodos de aprendizaje por refuerzo presentados anteriormente introducen algunos problemas típicos, no sólo de este tipo de aprendizaje, sino de otros ámbitos de la inteligencia artificial. Cabe destacar el problema de exploración *versus* explotación [Kaelbling *et al.*, 1996], que se resume en decidir cuándo el proceso de exploración del dominio da paso a la explotación del conocimiento adquirido para resolver más eficientemente la tarea. Otro de los puntos más importantes es que la definición de estados en función de atributos heterogéneos implica la definición de medidas de distancia entre esos estados, sobre los que a veces las medidas de distancia típicas, como el error cuadrático medio, pueden no ser adecuadas [Gersho and Gray, 1992]. Por último, destacar que la aproximación de funciones puede estar precedida de una etapa de extracción de características [Tsitsiklis and Roy, 1996], que permitan hacer esta aproximación de forma más eficiente.

2.6.1. Exploración y Explotación

El problema de la exploración y la explotación es uno de los puntos más estudiados en las técnicas de aprendizaje por refuerzo, así como en la mayoría de los problemas de la inteligencia artificial en general. Minimizar la cantidad de exploración puede ser muy importante, no sólo por el factor tiempo, sino

también porque el entrenamiento de los sistemas puede ser costoso. Este coste puede venir por dos lados, sobre todo si se tiene en cuenta la aplicación en sistemas reales. Por un lado, la exploración implica el funcionamiento de estos sistemas. Por otro lado, los procesos de exploración pueden llevar a la degradación de los propios sistemas.

Para obtener un equilibrio entre explotación y exploración pueden establecerse estrategias de distinto tipo. En los dos polos opuestos están las estrategias de comportamiento aleatorio, en las que el agente siempre ejecuta una acción elegida al azar, y las estrategias totalmente avariciosas, que eligen siempre la acción que se supone es la mejor, teniendo en cuenta lo que se ha aprendido hasta ese momento. Entre estas dos estrategias, existe un amplio número de ellas, como la estrategia $\epsilon - greedy$ [Watkins, 1989], que define una probabilidad ϵ de ejecutar acciones de forma avariciosa, y una probabilidad de $1 - \epsilon$ de ejecutar acciones de forma aleatoria.

Sin embargo, como hemos visto anteriormente, este equilibrio puede quedar roto cuando se requieren exploraciones para aprender una representación del espacio de estados, dado que esta exploración debe ser realizada con anterioridad al aprendizaje de la política. Otro inconveniente se encuentra cuando se aplica aprendizaje por refuerzo retardado en el tiempo. Dado que sólo se reciben refuerzos positivos cuando se llega a meta, y el resto de refuerzos son nulos, la relación entre el número de estados que reciben un refuerzo positivo con los que reciben un refuerzo nulo puede estar muy desequilibrada. Para evitar estos problemas, se pueden establecer estrategias de búsqueda de metas, o incluso que un humano guíe esta exploración [Smart, 2002].

2.6.2. Medidas de Distancia

Muchas de las técnicas de representación del espacio de estados están basadas en medidas de distancia para calcular la pertenencia o no de un ejemplo a un conjunto. Por ejemplo, las funciones de base radial, o las técnicas basadas en el vecino más cercano, suelen utilizar el error cuadrático medio para definir cuándo un ejemplo pertenece o no a una región. Sin embargo, el uso del error cuadrático medio, si bien estadísticamente suele ofrecer muy buenos resultados, puede ser inadecuado en muchos casos, sobre todo en aquéllos en los que los valores de los atributos que componen un estado son muy dispares. Supongamos, por ejemplo, un dominio bidimensional, en el que un atributo tiene valores en el rango $(0, 1)$ y otro en el rango $(0, 100)$. Si se aplica error cuadrático medio como medida de distancia, es obvio que toda la aportación vendrá dada por el segundo atributo, con lo que toda la información que aportara el primero sería anulada totalmente.

Típicas soluciones a este problema suelen pasar por la normalización de

los atributos, de forma que todos estén en el mismo rango de valores. Sin embargo, estas técnicas pueden no ser eficientes en dominios donde los datos sigan distribuciones complejas.

Por último, hay que destacar que el uso de medidas de distancia recibe una gran cantidad de ruido cuando no se están utilizando los atributos adecuados, es decir, la introducción de más atributos (más dimensiones) en el espacio puede hacer que ejemplos que antes estaban muy cerca, pasen a estar lejos debido al ruido que pueden introducir los nuevos atributos.

2.7. Dominios de Experimentación

En la presente sección se pretende describir de forma breve algunos dominios de experimentación donde el aprendizaje por refuerzo ha sido aplicado, y que aparecerán a lo largo de esta tesis doctoral.

Uno de los primeros dominios es similar al de navegación de un robot por un espacio abierto, descrito en la introducción, y mostrado gráficamente en la Figura 1.1. Este dominio planteaba un robot que debía aprender a llegar a una zona de meta fija, desde posiciones iniciales aleatorias en cualquier punto del dominio. Una primera extensión de este dominio es el de la navegación por oficinas, que plantea la misma dinámica, pero en un entorno más complejo, y que se describe en la sección 2.7.1. En la sección 2.7.2 se describe brevemente el dominio *Car on the Hill*, un problema ampliamente extendido para validar métodos de aprendizaje por refuerzo. En la sección 2.7.3 se describe el dominio (*CMOMMT*), un dominio de cooperación multi-agente, mientras que en la sección 2.7.4 se describe el dominio de la *RoboCup*.

2.7.1. Navegación por Oficinas

Este dominio consiste en un robot que se debe mover por un edificio (Figura 2.20). En el entorno se distinguen los muros, las zonas libres por las que puede moverse, y zonas de meta. El dominio completo tiene un tamaño de 24×21, y tanto las zonas libres, como los bloques de muro y zonas de meta se cuentan en bloques de tamaño 1×1. Las acciones posibles que puede ejecutar el robot son "Ir Norte", "Ir Sur", "Ir Este" e "Ir Oeste", todas ellas de tamaño 1. Se supone que el robot conoce en todo momento su posición en el espacio, que viene dado por dos coordenadas continuas (x, y), proporcionadas por algún sistema de localización. Además, el robot dispone de un sistema para evitar obstáculos, de forma que si intenta ejecutar una acción que haría que chocara contra un muro, el robot se bloquea y no se mueve. En este caso, se han situado dos zonas de meta, en dos habitaciones

diferentes.

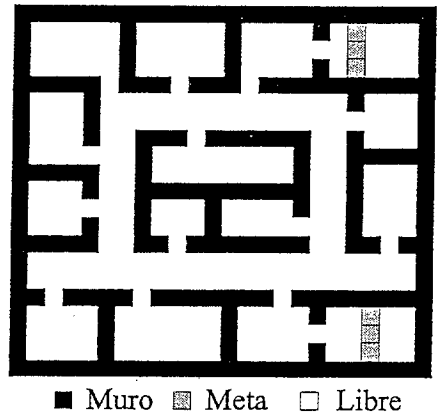


Figura 2.20: Dominio de navegación por oficinas.

El dominio, tal y como se ha planteado, es un dominio determinista. Sin embargo, una de las ventajas que ofrece es que es fácil convertirlo en un dominio estocástico. Esto se hace considerando que el sistema de localización o los actuadores pueden introducir errores, en este caso, de manera uniforme, en un determinado rango. Así, si consideramos que el dominio tiene un ruido de un $z\,\%$, se considera que el robot recibe como coordenadas de su posición las coordenadas $(x+ruido_x, y+ruido_y)$, donde x e y son las coordenadas a las que llegaría si los actuadores y la percepción no introdujeran error, y $ruido_x$ y $ruido_y$ son dos variables aleatorias que siguen distribuciones uniformes en el rango $[-z, z]$.

La dinámica concreta en este dominio es la siguiente. Dada una situación actual del agente, dada por las coordenadas (x, y), y una acción a, el siguiente estado al que se llega, denotado por (x', y'), se calcula del siguiente modo:

1. Calcular las nuevas coordenadas x' e y':
   - Si a ="Ir Norte", $x' = x$, e $y' = y + 1$
   - Si a ="Ir Sur", $x' = x$, e $y' = y - 1$
   - Si a ="Ir Este", $x' = x + 1$, e $y' = y$
   - Si a ="Ir Oeste", $x' = x - 1$, e $y' = y$
2. Verificar si la nueva posición es válida:
   - Si (x', y') están en un muro, el sistema de detección de obstáculos actúa, manteniendo el robot parado, por lo que se reestablecen los valores iniciales: $x' = x$, e $y' = y$

- Si (x', y') corresponde con una posición libre, se le introduce el ruido, haciendo $x' = x' + ruido_x$ e $y' = y' + ruido_y$, donde $ruido_x$ y $ruido_y$ son calculados siguiendo una distribución uniforme en el rango $[-z, z]$

3. Devolver x' e y'

2.7.2. *Car on the Hill*

Este dominio es ampliamente utilizado por la comunidad de aprendizaje por refuerzo [Munos and Moore, 1999]. El problema consiste en empujar un coche con el fin de localizarlo en la cima de la montaña (posición $x = 1$) con una velocidad nula ($v = 0$). Es por tanto también un dominio bidimensional, donde en todo momento se dispone de la posición del coche y de la velocidad. La posición del robot oscila entre -1 y +1, mientras que la velocidad lo hace desde -4 hasta +4. La Figura 2.21 muestra una representación gráfica de este problema.

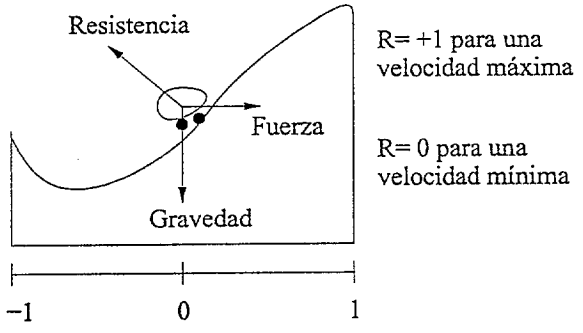


Figura 2.21: Dominio *Car on The Hill.*

Desde un punto de vista de aprendizaje por refuerzo, el objetivo del coche es obtener en cada intento un refuerzo máximo. El refuerzo inmediato depende del estado al que llega el coche en cada momento, o lo que es lo mismo, de su posición y su velocidad, tal y como muestra la función de refuerzo definida en la ecuación 2.17.

$$r(x, v) = \begin{cases} 0 & \text{if } x < 1 \\ f(v) & \text{if } x = 1 \end{cases} \tag{2.17}$$

donde $f(v)$ es una función lineal que da un máximo refuerzo +1 cuando la velocidad es nula, y un refuerzo mínimo de 0 cuando la velocidad es máxima (-4 ó 4). Por tanto, el refuerzo del agente es máximo cuando llega a la parte derecha del entorno con una velocidad mínima.

Las posibles acciones en este entorno son dos, y se resumen en aplicar sobre el carro una fuerza con módulo 4 ó -4, en la dirección que se muestra en la figura. La dinámica e implementación concreta del dominio está basada en la utilizada en [Munos and Moore, 1999]. [1]

2.7.3. *CMOMMT*

El dominio *Cooperative Multi-robot Observation of Multiple Moving Targets* (CMOMMT) [Parker, 1999] es un dominio de cooperación multiagente para seguimiento de objetivos. Este dominio se describe como sigue:

- $\mathcal{S}$: región espacial bidimensional y limitada
- $\mathcal{V}$: conjunto de m robots o vehículos, $v_i, i = 1, 2, ...m$, con 360^0 de rango de visión por sensores de capacidad limitada a una distancia definida.
- $\mathcal{O}(t)$: conjunto de n objetivos, $o_j(t)$, $j = 1, 2, ..., n$, tales que cada objetivo $o_j(t)$ está situado dentro de la región $\mathcal{S}$ en el instante t

Se dice que un robot v_i, está *observando* un objetivo cuando el objetivo está en el rango de visión de v_i. Se define una matriz, $B(t)$, de tamaño $m \times n$, como sigue:

$$B(t) = [b_{ij}(t)]_{m \times n} \text{ tal que } b_{ij}(t) = \begin{cases} 1 & \text{si el robot } v_i \text{ está } \textit{observando} \text{ al objetivo } o_j(t) \text{ en } \mathcal{S} \text{ en el instante } t \\ 0 & \text{en cualquier otro caso} \end{cases}$$

Dada esta matriz, se define como objetivo el desarrollar un algoritmo que maximice la medida A, definida en la ecuación 2.18.

$$A = \sum_{t=1}^{T} \sum_{j=1}^{n} \frac{g(B(t), j)}{T} \tag{2.18}$$

donde:

$$g(B(t), j) = \begin{cases} 1 & \text{si existe un } i \text{ tal que } b_{ij}(t) = 1 \\ 0 & \text{en cualquier otro caso} \end{cases}$$

Es decir, el objetivo de los robots es maximizar el número medio de objetivos en $\mathcal{S}$ que están siendo observados por al menos un robot a lo largo de un tiempo T. Adicionalmente, se define $sensor\_coverage(v_i)$ como la región visible por los sensores de cada robot v_i, para $v_i \in \mathcal{V}$. Entonces, se puede

[1] Disponible en http://www-2.cs.cmu.edu/~

asumir que la región máxima cubierta por los sensores de observación del equipo de robots es mucho menor que la región total a ser observada. Es decir, $\bigcup_{v_i \in \mathcal{V}} sensor\_coverage(v_i) \ll \mathcal{S}$. Esto implica que sensores de robots fijos o definidos para diversas zonas no son adecuados en general y, en contra, los robots deben vigilar de forma dinámica a los objetivos, con el fin de maximizar el tiempo que éstos están bajo observación.

La Figura 2.22 muestra una imagen del simulador CMOMMT. En este simulador, los círculos pequeños y negros representan robots, los cuadrados pequeños y rojos representan objetivos, y los círculos grandes y azules alrededor de los robots representan su rango de visión.

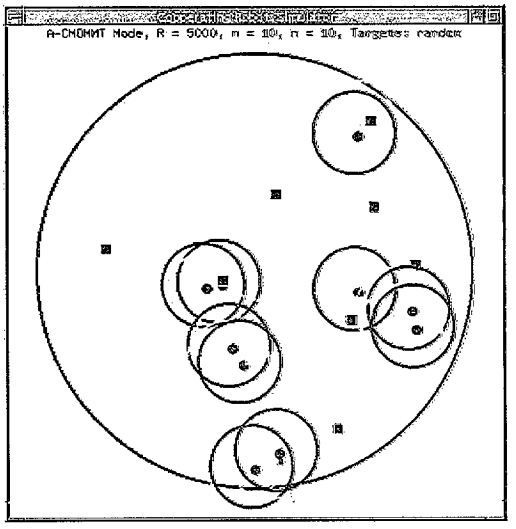


Figura 2.22: Simulador CMOMMT.

2.7.4. La *RoboCup*

La *Robot World Cup Initiative* (*RoboCup*) [Kitano *et al.*, 1997] es una iniciativa para desarrollar el estudio de la Inteligencia Artificial y los robots inteligentes proponiendo un problema estándar donde un amplio rango de tecnologías pueden ser integradas y estudiadas. Este problema estándar consiste en dos equipos de robots que juegan al fútbol. Entre las tecnologías que pueden ser integradas en estos dos equipos están el diseño de agentes autónomos, colaboración multiagente, adquisición de estrategias, razonamiento en tiempo real, etc.

El objetivo actual de la *RoboCup* es que para el año 2050 se haya desarrollado un equipo de robots humanoides totalmente autónomos que pueda ganar en un partido contra el campeón del mundo de fútbol de humanos. No obstante, a pesar de que el verdadero objetivo de la *RoboCup* son los robots reales, también se ofrece un soporte software en el que se pueden estudiar aspectos relacionados con ella, el simulador *Soccer Server*. *Soccer Server* [Noda *et al.*, 1998] es un sistema que permite que dos equipos implementados en diversos lenguajes de programación jueguen al fútbol entre ellos.

Para ello, se utiliza una arquitectura cliente-servidor. El servidor, *Soccer Server*, proporciona un campo virtual y simula los movimientos de los jugadores y el balón. La comunicación entre el servidor y los clientes se realiza vía *sockets* UDP/IP. Así, los clientes pueden ser implementados en cualquier arquitectura que permita comunicaciones de este tipo. Este servidor está compuesto por dos módulos principales. El servidor propiamente dicho, que establece las comunicaciones con los clientes y controla el juego, y un monitor que se encarga de generar la representación gráfica de lo que ocurre en el servidor y que, por tanto, muestra un campo de fútbol con los agentes disputando un partido. En la Figura 2.23 se muestra una imagen del simulador capturada durante un partido.

Los clientes definen el comportamiento de los jugadores, controlando sus movimientos de forma que cada uno de ellos dirige a un jugador. Mediante la conexión UDP/IP, recibe información sensorial (visual, auditiva y sobre el estado de su cuerpo) desde el servidor, y también a través de dicha conexión envía sus órdenes, o lo que es lo mismo, las acciones que quiere ejecutar sobre el entorno.

La información visual que los agentes reciben del entorno es variada. Pueden recibir información sobre la posición del balón, de los jugadores, y sobre marcas situadas por el campo que dan idea a los jugadores de dónde están situados los diversos elementos del entorno (como las porterías y el centro del campo). Toda esta información que reciben los agentes es subjetiva, es decir, relativa a su propia posición. Esto significa que un agente no recibe nunca un mensaje visual de que, por ejemplo, el balón está en la posición x e y del campo, sino que el balón está a una distancia d de él, y que lo está "viendo" con un ángulo a. Si se añade a esto que la información que se recibe del simulador, se recibe con un ruido proporcional a la distancia de los objetos, se obtiene una de las principales dificultades del dominio consistente en obtener una representación completa de la realidad.

Además, las acciones que ejecutan los agentes no son deterministas, lo que quiere decir que un agente que se encuentra en un determinado estado en el juego, si ejecuta una acción aislada, puede llegar a estados distintos en distintas ejecuciones de la acción. Esto es debido a que sobre las acciones de

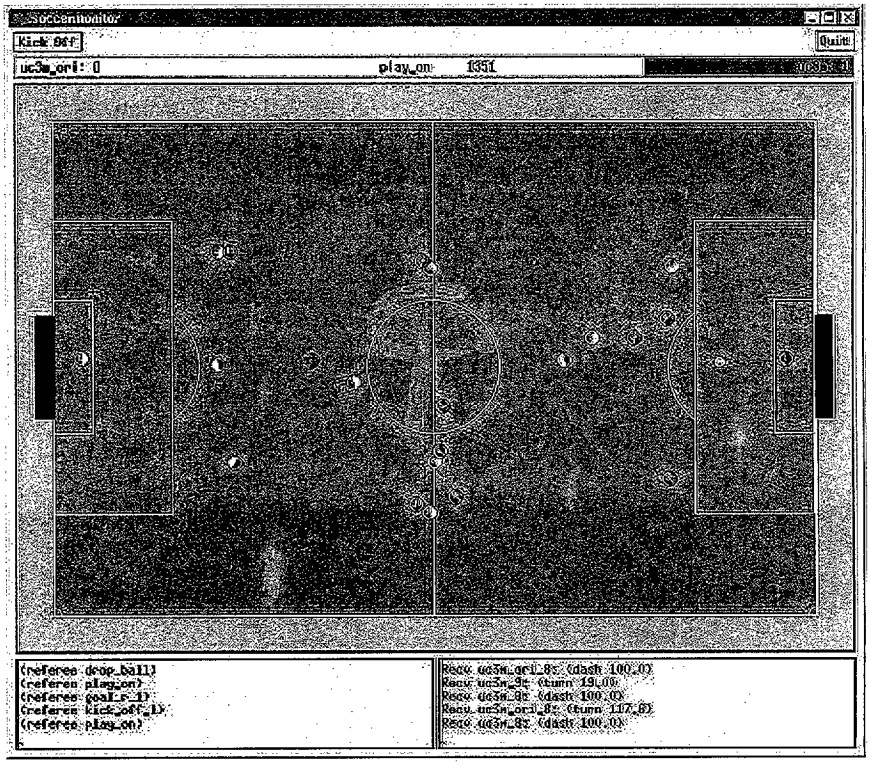


Figura 2.23: El simulador de fútbol *Soccer Server*.

los jugadores también se ejerce un ruido, que produce este indeterminismo.

Hasta ahora, los principales trabajos que se han desarrollado sobre la *RoboCup*, en lo que a la liga de simulación se refiere, han estado muy orientados al desarrollo de sistemas de control de agentes autónomos en sistemas multi-agente, que además permita la colaboración entre ellos [Stone, 2000].

Sin embargo, el desarrollo de habilidades de bajo nivel es un elemento imprescindible para el buen funcionamiento de sistemas de comportamiento como los nombrados anteriormente. En este sentido, se ha aplicado aprendizaje por refuerzo para el desarrollo de habilidades, tanto sencillas como detener el balón [Stone, 2000], como algunas más complejas como disparar a meta ante adversarios [Uchibe, 1999] o colaborar para defender la meta propia [Stone and Sutton, 2001],

2.8. Conclusiones

El propósito de este capítulo ha sido resumir el estado del arte del aprendizaje por refuerzo en general, y de los métodos aplicados para la generalización del espacio de estados en particular. Esta tesis doctoral se centra principalmente en los métodos de generalización basados en la discretización del espacio de estados y en los métodos de aproximación supervisada de las funciones de valor, por lo que en la Tabla 2.1 se muestran las principales conclusiones que se han extraído de ellas. En dicha tabla, se plantean las principales ventajas (marcadas con el símbolo $\surd$) y desventajas (marcadas con el símbolo $\otimes$) de cada uno de ellos, en lo que a distintos conceptos se refiere.

Concepto	Aproximación de Funciones Supervisada	Discretización del Espacio de Estados
Cantidad de Métodos	$\surd$ Cualquier método de aproximación de funciones obtenido de la literatura en aprendizaje supervisado	$\surd$ Principalmente discretizaciones uniformes, árboles de decisión, árboles *kd*, y métodos basados en el vecino más cercano
Aprendizaje en línea o por lotes	$\surd$ Ambos métodos	$\surd$ Ambos métodos
Mecanismos de aprendizaje	$\surd$ Los propios del aproximador elegido, por lo que están ampliamente estudiados en la literatura	$\otimes$ Dependientes del modelo de representación de la discretización
Convergencia	$\otimes$ Casos de divergencia tanto en la funciones de valor como en la política aprendida	$\otimes$ Casos de pérdida de la propiedad de Markov
Parámetros definidos por el usuario	$\otimes$ Arquitectura del aproximador, condiciones iniciales, parámetros de aprendizaje, etc.	$\otimes$ Resolución inicial, otras condiciones iniciales, parámetros de aprendizaje, etc.

Tabla 2.1: Diferencias entre los métodos de generalización basados en la discretización del espacio de estados y los basados en la aproximación supervisada de las funciones de valor.

La tabla muestra que existe una gran variedad de métodos de ambas aproximaciones de generalización. Entre estos métodos, además, se encuen-

tran modelos que permiten procesos de aprendizaje en línea y en procesos por lotes. Una de las ventajas que ofrece el uso de aproximadores basados en algoritmos de aprendizaje supervisado, es que tanto la arquitectura del aproximador, como el algoritmo de aprendizaje, están ya muy desarrollados en la bibliografía, por lo que sólo es necesario buscar el que se considere más adecuado, y utilizarlo. Sin embargo, los métodos basados en discretizaciones, por lo general requieren algoritmos de aprendizaje propios del método de discretización.

De la tabla también se desprende que los principales inconvenientes de uno y otro método son los que se refieren a la convergencia a las funciones de valor, y por tanto, a las políticas de acción óptimas. Se ha mostrado que los aproximadores de funciones, a veces no convergen a las funciones de valor óptimas, y tampoco a las políticas óptimas, por lo que la búsqueda del aproximador adecuado, puede no ser sencilla. Por otro lado, los métodos basados en la discretización del espacio de estados pueden producir la pérdida de la propiedad de Markov, que será extendida en la sección 4.5.

Por último, cabe destacar que la definición de parámetros y condiciones iniciales que se requieren para hacer efectivos los procesos de aprendizaje, pueden hacer que estos procesos no sean cómodos para el diseñador, y que requieran de un largo proceso de afinamiento de dichos parámetros.

Capítulo 3

Objetivos de la Tesis Doctoral

El objetivo de esta tesis doctoral se centra en el desarrollo de una técnica de aprendizaje por refuerzo libre de modelo que tenga la capacidad de generalizar en el espacio de estados. A este método se le exigen, además, unas características que se describen en la sección 3.1. El cumplimiento de estos objetivos puede medirse con determinadas variables que son descritas en la sección 3.2.

3.1. Objetivos

Los principales objetivos que se le exigen al modelo de aprendizaje por refuerzo desarrollado en esta tesis doctoral son:

- Eficiencia. Uno de los principales problemas que se plantean al aplicar técnicas de aprendizaje automático basados en prueba y error es cuántas veces es necesario intentar ejecutar una tarea para que el sistema aprenda a ejecutarla correctamente. Un elemento importante en este punto es la selección de un método de aprendizaje que no requiera la ejecución en tiempo de entrenamiento de las trayectorias óptimas para poder aprender la política óptima. Esto permitiría hacer exploraciones más aleatorias y un mayor aprovechamiento de la experiencia.
- Robustez. El sistema debe ser capaz de solucionar las tareas en cualquier situación, incluso en situaciones de ruido, típicas en la mayoría de los dominios.
- Generalización. El sistema debe ser capaz de adaptarse a situaciones nuevas que probablemente no se hayan planteado anteriormente, ni siquiera en la fase de entrenamiento, siendo capaz de obtener políticas completas a partir de experiencias reducidas.

- Independencia del dominio. Es deseable que la técnica pueda ser aplicada para aprender comportamientos en cualquier sistema que pueda ser representado como un problema de aprendizaje por refuerzo.
- Optimalidad. El objetivo del proceso de aprendizaje debe ser la obtención de políticas de comportamiento lo más cercanas al óptimo posible, donde el concepto de optimalidad dependerá de cada dominio en cuestión, aunque suele representarse con alguna función de coste. Por tanto, es necesario evitar problemas de convergencia a las funciones de valor y a las políticas.
- Dominios estocásticos. El método de aprendizaje a desarrollar debe ser aplicable no sólo en dominios deterministas, sino también en dominios estocásticos.
- Pocos parámetros en su definición/utilización. A la hora de aplicar cualquier método de aprendizaje es muy útil que el número de parámetros que se necesita definir para su utilización sea el menor posible, así como que pequeñas variaciones en los valores de esos parámetros no influyan mucho en los resultados obtenidos. Esto permite que el proceso de aprendizaje sea más rápido, y no requiera un gran proceso de refinamiento de todos los parámetros.

Algunas de estas características deseables u objetivos del sistema a desarrollar se convierten en variables o parámetros del sistema. Por ejemplo, el grado de indeterminismo de algunos dominios puede ser variado artificialmente, y por tanto, permite verificar el comportamiento de distintos métodos en función de ese parámetro. Además, otros objetivos pueden ser cuantificados, y nos sirven para medir la "calidad" de la soluciones obtenidas, y por tanto, de los métodos desarrollados, como por ejemplo, la robustez. Estos dos tipos de variables se detallan ampliamente cada vez que se plantee una experimentación, si bien se describen a alto nivel en la siguiente sección.

3.2. Evaluación de la Tesis Doctoral

La evaluación de la tesis doctoral se basa en la aplicación directa de las técnicas desarrolladas en los problemas tanto teóricos como prácticos, y que se concretan en los dominios de experimentación definidos en la sección 2.7. Con ellos se pretende, como principal objetivo, verificar la validez de las ideas presentadas, así como de las soluciones propuestas.

En estas evaluaciones, los parámetros a medir vienen dados por los objetivos de la tesis, en lo que a eficiencia, robustez, generalización, etc. se refiere,

tal y como se ha planteado anteriormente. Estos parámetros se pueden dividir en dos grupos. Por un lado, están aquéllos que van a ser entrada del sistema y que, por tanto, pueden derivar en distintos resultados que deberán ser objeto de análisis (variables independientes). Por otro lado, este análisis deberá tener en cuenta determinados parámetros de calidad a medir al final de cada experimentación (variables dependientes). Ambos tipos de variables se enumeran y definen brevemente a continuación.

- Variables independientes:

  1. Indeterminismo. Que el entorno sea determinista o estocástico, puede restringir el tipo de técnicas que se pueden aplicar a ese problema, ya que existen muchos métodos que presuponen dominios deterministas. Ese determinismo puede ser introducido en algunos dominios de forma controlada, por ejemplo, introduciendo ruido en la información que se recibe del entorno o en la señal de refuerzo. Se debe estudiar, por tanto, cómo afecta ese ruido al aprendizaje de una determinada tarea.
  2. Complejidad. La complejidad del modelo utilizado puede variarse también para la mayoría de los métodos que existen en la actualidad. Por ejemplo, si se utiliza como método de aproximación una discretización del espacio de estados, esta discretización puede ser de más o menos niveles. Estudiar la influencia de este parámetro y el cómo obtener un parámetro óptimo puede ser decisivo en la aplicación de muchos métodos.
  3. Parámetros propios del aprendizaje por refuerzo. Las funciones de Bellman introducidas en la sección 2.3.1 sirven de base para la mayor parte de los métodos de aprendizaje por refuerzo. Así, parámetros como el factor de descuento, γ, estarán presentes en los métodos desarrollados.
  4. Exploración y explotación. El tipo de exploración que se haga del entorno es también muy importante, tal y como se introdujo en la sección 2.6.1. Si se realizan exploraciones aleatorias, o si estas exploraciones están guiadas por la política que se está aprendiendo, es un elemento importante a comparar. Sin embargo, se mostrará posteriormente que los métodos utilizados en esta tesis requieren de exploraciones iniciales aleatorias, por lo que no dan lugar a discusión. No obstante, en los trabajos futuros de esta tesis se marcan nuevas líneas de trabajo en este sentido.
  5. Experiencia requerida. La cantidad de intentos que se van a realizar para aprender una tarea es un parámetro que define la duración

del proceso de aprendizaje por prueba y error, y que, por tanto, condiciona la calidad del aprendizaje realizado.

6. Parámetros propios de cada técnica. Cada método de generalización a utilizar puede introducir nuevos parámetros que es posible que el usuario tenga que afinar hasta encontrar soluciones aceptables.

- Variables dependientes:

1. Experiencia requerida. La experiencia requerida es, además de una variable de entrada del sistema, una medida de calidad muy importante, ya que si la cantidad de experiencia requerida por un método es pequeña, el coste de realizar el aprendizaje será también bajo.
2. Calidad del comportamiento aprendido. La calidad del comportamiento adquirido viene dado, en la mayoría de los dominios, por una función de coste, que suele ir asociada a la cantidad de recursos utilizados para completar una tarea.
3. Robustez. O la capacidad del método de aprender comportamientos correctos sea cual sea la situación inicial. En aprendizaje por refuerzo, este parámetro suele ser medido verificando el porcentaje de intentos que terminan en la solución adecuada y es, por tanto, independiente del parámetro de calidad o eficiencia, que mide, para aquellos intentos que han llegado a la solución, el coste de alcanzar dicha solución.
4. Sensibilidad a parámetros definidos por el usuario, y cantidad de estos. La facilidad con que un método es aplicable es también un factor muy importante. Así, aquellos métodos con muchos parámetros definidos por el usuario, o con parámetros difíciles de ajustar, juegan en desventaja, a la hora de ser aplicados de forma práctica, con respecto a aquéllos con pocos parámetros fácilmente ajustables.

Capítulo 4

El Modelo VQQL

En los capítulos anteriores, se ha mostrado que uno de las principales soluciones que se plantean para generalizar en el espacio de estados, es dividir el espacio original en regiones, o *clusters*, que agrupen estados del espacio inicial. La aproximación más sencilla que puede plantearse es, por tanto, dividir el espacio en regiones uniformes, de un tamaño suficiente como para aproximar correctamente la función de valor, pero no demasiado grande, para que la cantidad de experiencia requerida para aprender sobre ese conjunto de estados no sea demasiado alta. El problema es que encontrar esa discretización óptima puede no ser sencillo, o incluso imposible, dado que unas zonas del espacio pueden requerir una determinada resolución de la discretización, y otras zonas, una resolución distinta.

Una de las aproximaciones más directas en este sentido, fueron los métodos de discretización de resolución variable [Moore and Atkeson, 1995], introducidos en la sección 2.4.4. Estos métodos se basan en discretizar el entorno utilizando árboles *kd*, de forma que se va aumentando la resolución de la discretización en aquellas zonas donde hay gran varianza en la función de valor para los estados de dicha región. Estos métodos, sin embargo, sólo han mostrado su utilidad en dominios pequeños de dos dimensiones, ya que para más dimensiones son superados incluso por discretizaciones uniformes [Munos and Moore, 2002].

La primera aproximación que se plantea en esta tesis doctoral es utilizar regiones de Voronoi [Gersho and Gray, 1992] para discretizar el espacio de estados. Para definir estas regiones, basta con definir una serie de prototipos distribuidos por el espacio y una medida de distancia (generalmente distancia euclídea). Los prototipos, y la regla del vecino más cercano, permiten dividir el espacio en regiones de forma muy sencilla. La potencia de utilizar regiones de Voronoi consiste en que hay muchos métodos que permiten definir el conjunto de prototipos a utilizar, como el Algoritmo de Lloyd Generalizado

(GLA) [Lloyd, 1982].

Este modelo fue resultado de la investigación desarrollada durante el proyecto fin de carrera en Ingeniería en Informática del autor de esta tesis doctoral [Fernández, 1999], y fue denominado VQQL (*Vector Quantization for Q-Learning*). El modelo surge a partir de las ideas planteadas anteriormente, y utiliza el algoritmo GLA para obtener un conjunto de prototipos que discretice el espacio de estados, para posteriormente usar Q-Learning para aprender la función $Q(s, a)$ en forma tabular.

Este capítulo recoge la descripción del modelo VQQL a partir de la realizada en [Fernández, 1999]. Así, en la sección 4.1 se resumen los principales conceptos de la cuantificación vectorial y en la sección 4.2 se introduce el algoritmo GLA. La sección 4.3 describe el algoritmo VQQL, mientras que la sección 4.4 muestra los principales experimentos que se realizaron sobre dicho modelo. El primero de ellos, en el dominio de la *RoboCup*, ampliamente descrito en [Fernández, 1999], y el segundo de ellos, sobre *CMOMMT*, realizado con posterioridad a la finalización del proyecto fin de carrera. Por último, la sección 4.5 muestra el principal inconveniente de este método, como es la pérdida de la propiedad de Markov.

4.1. Cuantificadores de Voronoi

La cuantificación surgió en el campo de las comunicaciones digitales como una herramienta necesaria para la transmisión digital de información analógica [Haykin, 1988], que debía ser discretizada o aproximada a un conjunto de valores finito, de forma que pudieran ser transmitidos digitalmente. Su desarrollo, no obstante, vino por la necesidad no sólo de realizar la discretización, sino de que la cantidad de información requerida para transmitir la señal fuera la mínima posible, pero suficientemente grande como para que la señal original pudiera ser reconstruida en el receptor. Se trata, por tanto, de un método para obtener un conjunto de niveles reducido que permita aproximar un conjunto continuo. A partir de esta breve descripción, se extrae directamente su aplicación para la discretización del espacio de estados en problemas de aprendizaje por refuerzo, donde, además de discretizar un espacio de estados, se desea que esta discretización se realice a un número de estados pequeño, pero que permita posteriormente aproximar correctamente las funciones de valor.

Un cuantificador vectorial, S, de dimensión K y tamaño N es una correspondencia de un vector (o un punto) en el espacio euclídeo K-dimensional, R^K, en un conjunto finito C con N salidas (distintas) o prototipos llamado alfabeto $S : R^K \longrightarrow C$, donde $C = \{y_1, y_2, \ldots, y_N\}$, e $y_i \in R^K$ para cada y

desde 1 hasta N [Gersho and Gray, 1992].

Dado el cuantificador C, y un vector $x \in R^K$, $S(x)$, la cuantificación asigna a x el vector de C más cercano. Siguiendo el ejemplo mostrado en la Figura 4.1, todos los valores pertenecientes a una celda se cuantificarían como el punto que representa dicha celda. Para medir el concepto de cercanía o lejanía, se debe definir una medida de distancia entre cualquier par de vectores.

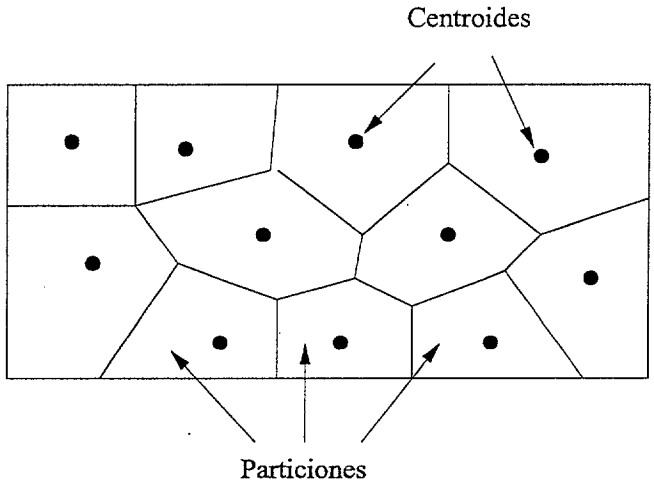


Figura 4.1: Particiones y Centroides.

Una medida de distorsión es una asignación de coste no negativa $d(x, \hat{x})$ asociada a la cuantificación de cualquier vector $x \in R^K$ sobre un vector de reproducción $\hat{x} = S(x) \in C$. Se han utilizado numerosas medidas para el cálculo de la distorsión, cada una de ellas más adecuada para un problema concreto y conviene cambiar de heurística según se trate con diferentes dominios. En este trabajo se ha utilizado el error cuadrático medio, descrito en la ecuación 4.1.

$$dist(x, \hat{x}) = \sum_{i=0}^{K-1} (x_i - \hat{x}_i)^2 \tag{4.1}$$

donde:

- x es el vector original
- $\hat{x}$ es el prototipo
- $x_i, i = 0, \ldots, K-1$ es la componente i-ésima del vector original

- $\widehat{x}_i, i = 0, \ldots, K-1$ es la componente i-ésima del prototipo

Cada uno de los N puntos definidos en el alfabeto está asociado a una partición de R^K o celda, R_i, definida mediante la ecuación 4.2:

$$R_i = \{x \in R^K : S(x) = y_i\} \tag{4.2}$$

La idea intuitiva para explicar el concepto de partición dentro del contexto de los cuantificadores es la siguiente. Uno de los parámetros del cuantificador es el número de niveles o tamaño N y otro la dimensión K. El número de niveles indica cuántas regiones, o lo que es lo mismo, cuántos puntos compondrán el cuantificador. Estos puntos formarán unas regiones de dimensión K o particiones; en realidad estos puntos son los centroides de estas regiones. Se define el centroide de una partición R dada, $y^* = cent(R)$, como el vector y que minimiza la distorsión media entre los puntos x de R e y^*:

$$y^* = cent(R) \text{ si } E[d(x, y^*)|x \in R] \leq E[d(x, y)|\forall y \in R] \tag{4.3}$$

Un método sencillo para calcular el centroide de una partición es la media aritmética:

$$cent(R) = \frac{1}{\|R\|} \sum_{i=1}^{\|R\|} x_i \tag{4.4}$$

donde $R = \{x_i : \quad i = 1, \ldots, \|R\|\}$ y $\|R\|$ es la cardinalidad de R.

En la Figura 4.1 se ha representado un espacio de 2 dimensiones ($K = 2$), que se ha dividido en 11 particiones ($N = 11$). Cada una de las particiones contiene un punto que representa al centroide de la partición.

Existe una clase especial de cuantificadores vectoriales, denominados cuantificadores vectoriales de *Voronoi* o del vecino más cercano (*Nearest Neighbor Quantizers*). Estos cuantificadores se caracterizan porque las particiones quedan totalmente definidas por un alfabeto y una medida de distorsión. La principal ventaja que ofrecen es que no requieren ninguna definición explícita de la descripción geométrica de las celdas, sino que esta información va definida implícitamente con el alfabeto y la medida de distorsión. Así, se puede definir un cuantificador vectorial de Voronoi o *NN* (*Nearest Neighbor*) como aquél en que la partición en celdas viene dada por la ecuación 4.5.

$$R_i = \{x : d(x, y_i) \leq d(x, y_j) \forall y_j \in C\} \tag{4.5}$$

En otras palabras, con un codificador *NN*, cada celda R_i se forma con todos los puntos x que tienen una distorsión cuando son codificados con el vector y_i menor que la que tendrían si se codificaran con cualquier otro vector del alfabeto.

De esta forma, podemos definir la función Q como en la ecuación 4.6.

$$Q(x) = \arg \min_{y \in C} \{d(x, y)\} \tag{4.6}$$

Normalmente, cuando se habla de cuantificadores vectoriales, se habla de cuantificadores vectoriales *NN*. En el resto de esta memoria se asumirá también esta equivalencia.

La cuantificación vectorial ofrece ventajas estadísticas sobre el caso escalar [Gersho and Gray, 1992]. El aprovechamiento de estas ventajas está determinado por el diseño del propio cuantificador. En este sentido, cobra vital importancia el Algoritmo de Lloyd Generalizado [Lloyd, 1957, Lloyd, 1982], que permite diseñar fácilmente cuantificadores que minimizan la distorsión para un conjunto de ejemplos de entrenamiento.

4.2. El Algoritmo de Lloyd Generalizado

El Algoritmo de Lloyd Generalizado (GLA) es una extensión del Algoritmo de Lloyd para el caso escalar. Dicho algoritmo está basado en el uso iterativo de operaciones de modificación de un alfabeto inicial, operaciones que se resumen en la denominada Iteración de Lloyd. Ambos casos, el escalar y el vectorial, son paralelos y siguen los mismos principios. Esta sección se centra en el caso vectorial.

Existen dos variantes de la Iteración de Lloyd. En la primera de ellas, se conoce la distribución de la señal de entrada, con lo cual el cálculo de las particiones se hace más fácil (*Iteración de Lloyd Conocidas las Estadísticas*). En la segunda, esa distribución se desconoce, y se utiliza una señal de entrada que sirve como casos de ejemplo o conjunto de entrada, y a partir de los cuales operará el algoritmo (*Iteración de Lloyd para Datos Empíricos*). Este apartado se centra en la segunda variante y todo lo escrito en adelante hace referencia a este caso. La descripción detallada de ambos métodos puede encontrarse también en [Gersho and Gray, 1992] y en [Linde *et al.*, 1980].

La Iteración de Lloyd

La Iteración de Lloyd para Casos Empíricos se aplica directamente sobre la distribución definida por los casos de prueba, T, para obtener un cuantificador para esta distribución. La idea esencial es que si se toma un conjunto finito T de tamaño M suficientemente grande, y se obtiene un cuantificador suficientemente bueno para este conjunto, este cuantificador será también bueno para la distribución real. La Iteración de Lloyd para Casos Empíricos consta de dos pasos, y se describe en la Figura 4.2

Iteración de Lloyd para casos Empíricos (C_m, T, N)

1. Dado un alfabeto $C_m = \{y_i; i = 1, \ldots, N\}$, partir el conjunto de entrada T en particiones R_i usando la siguiente condición:

$$R_i = \{x \in T : d(x, y_i) \leq d(x, y_j);\ \text{para todo } j \neq i\}$$

2. Usando la definición de centroide dada en la ecuación 4.3, calcular los centroides de cada partición para calcular el alfabeto. Hacer $C_{m+1} = \{cent(R_i)\}$. Si se generó una celda vacía en el paso 1, se asignará un vector alternativo (en vez del cálculo del centroide) para esa celda.

Figura 4.2: Iteración de Lloyd para casos empíricos.

En el segundo paso de la Iteración de Lloyd se plantea el problema de la celda vacía. Existen diversas alternativas para solucionar este problema, como generar un centroide aleatoriamente, o partir otras celdas en dos siguiendo distintas heurísticas para elegir la celda a partir. Este problema será nuevamente tratado posteriormente.

El Algoritmo de Lloyd Generalizado

En la Figura 4.3 se concretan los pasos de que consta el Algoritmo de Lloyd Generalizado. Un problema que surge en el primer paso de este algoritmo es el de elegir un alfabeto inicial. Este problema será discutido posteriormente, aunque se adelanta que su solución está relacionada con los mecanismos introducidos anteriormente para resolver el problema de las celdas vacías que nació con la Iteración de Lloyd.

En el paso 3 del algoritmo de Lloyd generalizado aparece también un concepto nuevo, la distorsión media, que se utiliza como medida de la calidad del diseño del cuantificador. La distorsión media puede definirse como la media de la distorsión que se produce entre cada vector del conjunto de casos de prueba, y el vector del alfabeto al que se cuantifica:

$$D = \frac{1}{M} \sum_{j=1}^{M} \min_{y \in C}(d(x_j, y)) \tag{4.7}$$

La forma más común de utilizar esta medida para generar la condición

Algoritmo de Lloyd Generalizado (C_1, T, N)

1. Comenzar con un alfabeto inicial C_1. Sea $m = 1$.
2. Dado un alfabeto, C_m, ejecutar la Iteración de Lloyd para generar un nuevo alfabeto C_{m+1}.
3. Calcular la distorsión media para C_{m+1}.
4. Si ha cambiado en una pequeña cantidad solamente desde la iteración anterior, parar. Sino, hacer $m = m + 1$ e ir al paso 2.

Figura 4.3: Algoritmo de Lloyd Generalizado.

de fin del algoritmo de Lloyd se muestra en la ecuación 4.8.

$$(D_m - D_{m+1})/D_m < \epsilon \tag{4.8}$$

donde ϵ es un umbral que se utiliza como parámetro de diseño, $0 \leq \epsilon \leq 1$.

Anteriormente se han dejado dos cuestiones abiertas. Por un lado, cómo resolver el problema de la celda vacía, de la iteración de Lloyd (Figura 4.2). Por otro lado, cómo seleccionar un conjunto de prototipos inicial en el algoritmo GLA. Ambos problemas tienen una solución relacionada, que está basada en el algoritmo de *splitting* descrito en [Gersho and Gray, 1992]. Este algoritmo se muestra en la Figura 4.4 y resuelve el problema del alfabeto inicial, y parece resolver también el de la celda vacía, ya que si se va partiendo el alfabeto inicial, parece que no se debería llegar nunca a obtener celdas vacías.

Sin embargo, se puede comprobar empíricamente que el problema de la celda vacía sigue apareciendo, por lo que hay que buscar alguna variante que permita solucionarlo totalmente. Esta variante consiste en integrar el concepto de *split* o división de particiones en la misma iteración de Lloyd, de forma que ante el problema de la celda vacía, baste con dividir otra celda en dos, y reasignar la nueva celda generada a la que se había quedado vacía. De esta forma, en la iteración de Lloyd, todas las celdas vacías son reasignadas. Por tanto, la iteración de Lloyd se modificaría tal y como se muestra en la Figura 4.5.

Dado que el problema del alfabeto inicial sigue presente, se puede mantener el algoritmo definido en la Figura 4.4, pero sin hacer la reasignación de celdas. Es decir, se multiplica por dos el número de estados que contiene el alfabeto, y a todos aquellos que no están definidos se les asigna una

Algoritmo de splitting (T, N)

1. Comenzar con un alfabeto inicial, C_1, con n (número de niveles del alfabeto) igual a 1. El único nivel del alfabeto es, por tanto, el centroide del conjunto de vectores de entrada.

2. Repetir

   a) Dado el alfabeto C_n, con n vectores $y_i, i = 1, \ldots, n$, partir cada vector y_i en dos muy cercanos, $y_i - \epsilon, y_i + \epsilon$. La colección $\{y_i - \epsilon, y_i + \epsilon; i = 1 \ldots, n\}$ tiene $2 * n$ niveles.

   b) $n \leftarrow n * 2$

   c) Ejecutar el algoritmo GLA con los parámetros C_n, T, n.

   Hasta que n sea el nivel N deseado.

Figura 4.4: Algoritmo de *Splitting* para resolver el problema del alfabeto inicial.

Iteración de Lloyd para casos Empíricos con Splitting (C_m, T, N)

1. Dado un alfabeto $C_m = \{y_i; i = 1, \ldots, N\}$ partir el conjunto de entrada T en particiones R_i usando la siguiente condición:

$$R_i = \{x \in T : d(x, y_i) \leq d(x, y_j); \forall j \neq i\}$$

2. Usando la definición de centroide dada en la ecuación 4.3, calcular los centroides de cada partición para calcular el alfabeto. Hacer $C_{m+1} = \{cent(R_i)\}$.

3. Sea la colección $y_i, i = 1 \ldots, r$, con $r \leq N$, que representa las celdas no vacías obtenidas en el paso anterior, ordenadas por algún criterio, como la distorsión media de la partición, o el número de vectores de la entrada que contiene la partición. Sea *partir* el número de celdas vacías. Generar la colección:
   $\{y_1 - \epsilon, y_1 + \epsilon, \ldots, y_{partir} - \epsilon, y_{partir} + \epsilon, y_{partir+1}, y_{partir+2}, \ldots, y_r\}$

4. Si no se han completado los N niveles necesarios ($r + partir < N$), rellenar con vectores nulos.

Figura 4.5: Iteración de Lloyd para casos Empíricos que soluciona el problema de la celda vacía.

celda cualquiera, de forma aleatoria, o el vector origen. Todos estos niveles serán reasignados correctamente en la iteración de Lloyd, ya que allí se catalogarán con mucha seguridad como celdas vacías. La descripción definitiva del algoritmo de *Splitting* utilizada se muestra en la Figura 4.6.

Algoritmo de splitting (T, N)

1. Comenzar con un alfabeto inicial, C_1, con n (número de niveles del alfabeto) igual a 1. El único nivel del alfabeto es, por tanto, el centroide del conjunto de vectores de entrada.
2. Repetir
   a) Transformar el alfabeto de n niveles en otro de $2 * n$ niveles, que contenga al anterior más n nuevos niveles generados aleatoriamente, o iguales al vector nulo.
   b) Asignar a n el valor $n * 2$
   c) Ejecutar el algoritmo GLA con los parámetros C_n, T, n.

   Hasta que n sea el nivel N deseado.

Figura 4.6: Algoritmo de *Splitting* definitivo para resolver el problema del alfabeto inicial.

A pesar de las modificaciones que se han introducido en la iteración de Lloyd, el algoritmo de Lloyd Generalizado definido en la Figura 4.3 sigue siendo válido, y es el que se ha implementado en el modelo VQQL.

4.3. Descripción del Algoritmo VQQL

El algoritmo VQQL tiene como objetivo el aprendizaje de una política de acción sobre un espacio de estados discretizado. Por tanto, es un método que sigue el esquema planteado en la Figura 2.14. No obstante, la discretización del espacio de estados viene dada por regiones de Voronoi, y por tanto, por un prototipo. Por tanto, el esquema que se plantea es el mostrado en la Figura 4.7, en la que se muestra que cada prototipo representa una fila en la tabla de valor-acción.

El algoritmo VQQL se compone de dos pasos fundamentales:

1. **Aprender el Cuantificador.** Diseñar un cuantificador vectorial de

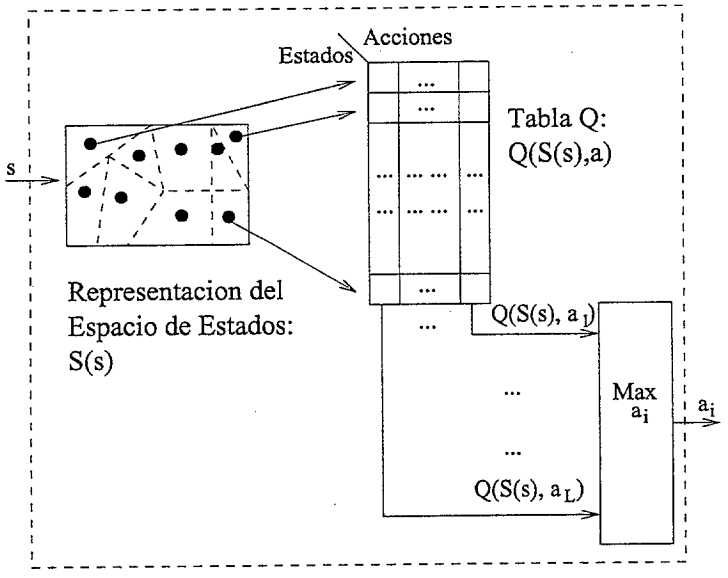


Figura 4.7: Representación de la función $Q(s, a)$ basada en la discretización con regiones de Voronoi.

N niveles a partir de estados ejemplo obtenidos del entorno. Este cuantificador proporciona una nueva discretización del espacio de estados.

2. **Aprender la Política de Acción.** Una vez que se obtiene una nueva representación del espacio de estados finita y reducida, utilizarla para aprender la política a través de la función Q representada de forma tabular.

La aplicación de la cuantificación vectorial con el fin de obtener una reducción del dominio sobre la que aplicar un método de aprendizaje por refuerzo discreto, puede ser desarrollado aprendiendo la política de forma interactiva o en línea, tras una exploración inicial para aprender el cuantificador, tal y como se muestra en la Figura 4.8.

Esta versión del algoritmo conlleva, por tanto, dos exploraciones. La primera de ellas para obtener los estados de ejemplo con los que construir la discretización. La segunda, para aprender la política. No obstante, dado que el diseño del cuantificador requiere una extensa exploración inicial en el entorno, esa misma exploración puede ser utilizada como experiencia para aprender la política. Por tanto, la interacción del agente con el entorno realizada en el paso 3 del algoritmo, puede ser sustituida por una serie de actualizaciones de la tabla Q a partir de las transiciones de estados generada en la exploración del paso 1. El conjunto de pasos completo se muestra en la Figura 4.9.

VQQL en Línea (N)

1. Obtener un conjunto T de ejemplos para diseñar el cuantificador vectorial mediante GLA de tamaño N, realizando una exploración del agente en el entorno.

2. Diseñar el cuantificador, C, utilizando los estados en T generados anteriormente como vectores.

3. Aprender la tabla Q mediante la interacción del agente con el entorno. Se deben realizar los siguientes pasos:

   a) Obtener una tupla de experiencia, $< s, a, s', r >$, desde el entorno, utilizando en la selección de la acción a ejecutar una política de exploración/explotación.

   b) Cuantificar los estados de la tupla de experiencia mediante el cuantificador, C, diseñado anteriormente. Se genera una nueva tupla $< \hat{s}, a, \hat{s}', r >$.

   c) Actualizar la tabla Q con la tupla de experiencia anterior, utilizando la función de actualización de *Q-Learning* definida en la ecuación 2.12.

Figura 4.8: Algoritmo VQQL en línea.

Los experimentos preliminares que se han realizado con el algoritmo VQQL muestran que la aplicación de métodos de agrupación no supervisada en general, y del algoritmo GLA en particular, para obtener una nueva representación del espacio de estados es muy útil, principalmente porque permite una gran reducción en el tamaño del espacio de estados, y por tanto, una gran reducción del proceso de prueba y error para obtener políticas adecuadas. Esto permite obtener buenos resultados cuando el espacio de estados inicial tiene pocas dimensiones como en *Car on the Hill*. Además, en espacios de estados de hasta 6 dimensiones, como en CMOMMT, los resultados también son bastante satisfactorios. A continuación se muestran algunos de estos resultados, si bien en el capítulo 7 se mostrarán más comparativas.

Sin embargo, este método tiene un principal inconveniente, y es la pérdida de la propiedad de Markov. Este inconveniente, presente en la mayoría de los métodos de codificación gruesa del espacio de estados, se define en la sección 4.5, y muestra la motivación de un método supervisado en la discre-

VQQL Fuera de Línea (N)

1. Obtener un conjunto T de ejemplos para diseñar el cuantificador vectorial mediante GLA de tamaño N, realizando una exploración del agente en el entorno. En el mismo proceso exploratorio, obtener las tuplas de experiencia, generando un conjunto P de tuplas del tipo $< s, a, s', r >$.
2. Diseñar el cuantificador C utilizando los estados en T generados anteriormente como vectores.
3. Cuantificar los estados de todas las tupla de experiencia almacenadas en P mediante el cuantificador C diseñado anteriormente.
4. Aprender la tabla Q a partir de las tuplas de P, utilizando la función de actualización de *Q-Learning* definida en 2.12.

Figura 4.9: Algoritmo VQQL fuera de línea.

tización del espacio de estados.

4.4. Experimentos

En esta sección se plantean los primeros experimentos que se realizaron con el modelo VQQL, si bien en posteriores capítulos se presentarán experimentos adicionales y comparativas con otros métodos en otros dominios.

4.4.1. *RoboCup*

En la sección 2.7.4 se ha introducido el dominio de la *RoboCup*, y del simulador *Soccer Server*, para el desarrollo de arquitecturas de control y habilidades de jugadores de fútbol simulados. Una de las habilidades más básicas que un jugador de fútbol, en concreto el portero, debe tener es la de interceptar el balón. Con el objetivo de interceptar el balón, un agente debe analizar la información sensorial que recibe desde el simulador para obtener la información que necesita. De entre toda la información que recibe, se ha

decidido utilizar la siguiente:[1]

- Distancia relativa del balón al jugador (*Distance*).
- Dirección relativa del balón al jugador (*Direction*).
- La variación de la distancia, que da una idea de cómo está cambiando el parámetro distancia (*DistChng*)
- La variación de la dirección, que da una idea de cómo está cambiando el parámetro dirección (*DirChng*)

Estos cuatro parámetros componen un estado a efectos del aprendizaje. Una vez obtenidos estos parámetros el portero puede ejecutar diversas acciones para atrapar el balón:

Turn. Cambiar la dirección del jugador de acuerdo a un ángulo entre -180 y 180 grados.

Dash. Incrementar la velocidad del jugador en la dirección en la que se encuentra, con una potencia que puede oscilar entre -30 y 100.

Catch. Atrapar el balón, indicando la dirección en la que se encuentre el balón, con la condición de que el balón debe estar en una zona predeterminada relativa al jugador denominada *zona de portero* (*catchable area*) de dos metros de largo y un metro de ancho, tal y como muestra la Figura 4.10. [2]

Dado que este conjunto de acciones es finito, es necesario también discretizarlo. Para ello, se definen un conjunto de macro-acciones. Dichas macro-acciones están compuestas de dos acciones consecutivas: $turn(T)$ y $dash(D)$, que generan la macro-acción $turn - dash(T, D)$. Esto significa que, en cada momento, el agente realizará un giro y una acción de avanzar de forma consecutiva. Sin embargo, hasta ahora no se ha hecho más que trasladar el problema, ya que se siguen manteniendo los parámetros T y D, que pueden tomar múltiples valores. Para resolver esto de una forma sencilla, se ha decidido que D siempre valga 100, y que T sea calculado de acuerdo a la ecuación 4.9.

$$T = A + \Delta_A \tag{4.9}$$

[1] En este experimento se utilizó el protocolo 4 del simulador *Soccer Server*. Con otros protocolos u otras versiones del simulador, se podrían haber utilizado distintos parámetros.

[2] Las dimensiones del terreno de juego del simulador *Soccer Server* son 105 metros de largo por 68 metros de ancho.

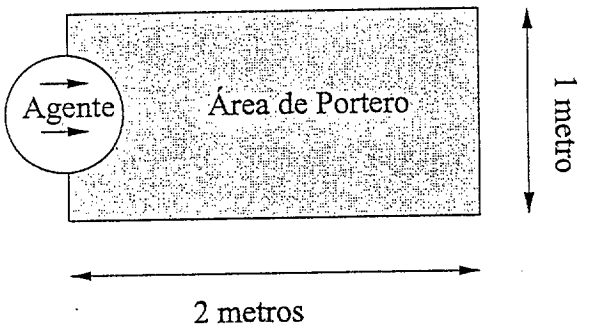


Figura 4.10: Descripción de "área de portero". El agente está orientado en el sentido de las flechas.

donde A es el ángulo entre el agente y la pelota, y Δ_A puede ser uno de los valores siguientes: +45, +10, 0, -10, -45. De esta forma, se ha reducido el número de acciones posibles sólo a cinco.

El objetivo del aprendizaje es que la pelota se sitúe en dicha zona de portero, para que así el comando *catch* pueda ser ejecutado con éxito. Sin embargo, la ejecución del comando *catch* en la zona de portero no siempre consigue que el portero atrape el balón, debido a las dificultades que existen para sincronizar totalmente los agentes con el entorno. Así, se considera que el portero ha conseguido su objetivo, no cuando consigue atrapar el balón, sino cuando consigue situarse lo suficientemente cerca de él como para que el balón esté en su zona de portero.

Como se ha comentado anteriormente se ha definido un escenario para el aprendizaje. La adquisición de datos para diseñar el cuantificador y aprender el comportamiento se basa en la repetición de muchas jugadas sobre ese mismo escenario, en el que dos openentes, el portero y el delantero, se comportan de forma aleatoria. El escenario concreto para realizar el aprendizaje es similar al utilizado en [Stone, 2000] y queda definido en la Figura 4.11.

Usando una Heurística Programada

Antes de mostrar los resultados obtenidos al utilizar el modelo VQQL para adquirir la habilidad de interceptar el balón, cabe indicar que si el portero sigue una política aleatoria para elegir la macro-acción a ejecutar en cada momento, no consigue situarse en la zona de portero en más de un 20 % de los lanzamientos que van a gol. Otra heurística básica consiste en que el portero siempre se dirija hacia el balón, en una actitud de persecución ciega, es decir, sin intentar extraer ningún tipo de conocimiento del resto de

Escenario de Aprendizaje de HIB

1. El portero comienza en medio de la portería a una distancia de 4 metros delante de ella, mirando hacia el centro del campo.

2. El balón y el delantero se sitúan justo delante del portero, a una distancia que oscila entre 15 y 25 metros delante del portero.

3. El delantero avanza hacia la pelota y dispara (comando *kick*) con una fuerza máxima hacia la portería, y con un ángulo aleatorio en el rango $(-20, 20)$.

4. El objetivo del portero es moverse de forma que el balón quede situado en su área de portero. Para ello, espera a que el balón esté a una distancia menor o igual a 14 metros, y entonces empieza a ejecutar las macro-acciones definidas anteriormente hasta que se produce el final de la jugada. El final de dicha jugada puede venir dado por distintos motivos:

   - El portero sitúa el balón en su área de portero.
   - El delantero marca gol.
   - El balón sale fuera del campo por la línea de fondo.
   - El jugador pierde de vista el balón.

5. Si el portero ha conseguido colocar el balón en su zona de portero, es que ha conseguido su objetivo, y recibirá un refuerzo positivo. En cualquier otro caso, se considera una jugada no satisfactoria, y recibirá un refuerzo cero.

Figura 4.11: Escenario para el aprendizaje de la habilidad de interceptar el balón.

parámetros que recibe desde el entorno. Si el portero sigue esta política de dirigirse siempre hacia el balón, no supera un 30 % de objetivos alcanzados.

Sobre este escenario se ha aplicado el modelo VQQL fuera de línea. Dado que el modelo de aprendizaje reconoce dos fases independientes, el diseño del cuantificador y el aprendizaje del comportamiento, inicialmente van a mostrarse por separado los resultados obtenidos en cada uno de los pasos, para posteriormente obtener las conclusiones de cómo afectan los resultados

del primero en el segundo; es decir, cómo afecta la calidad del diseño del cuantificador en el aprendizaje de la habilidad.

Diseño del cuantificador

Para el diseño del cuantificador se tomó un conjunto de 94.852 estados de ejemplo. En la Figura 4.12(a) se muestran los parámetros de distancia y dirección de dichos estados. En ella se muestran las dependencias estadísticas que existen entre estas dos componentes.

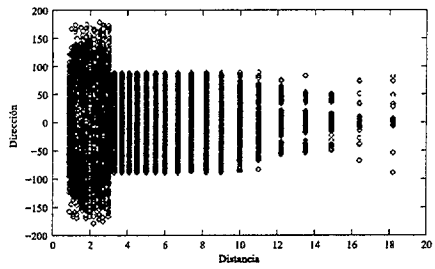
Una primera solución podría ser utilizar un cuantificador escalar simple para hacer la partición del espacio de estados. Así, se podría diseñar un cuantificador vectorial uniforme de 4096 estados, lo cual es equivalente a utilizar un cuantificador escalar de 8 niveles por cada atributo que forma un estado. En la Figura 4.12(b) se muestra el cuantificador que se obtendría siguiendo este mecanismo.[3] En esa figura se observa que se forma una matriz de 8×8 puntos que ocupa todo el espacio euclídeo representado. En la Figura 4.12(c) se muestra la superposición de las figuras que representan los datos originales (Figura 4.12(a)) y el cuantificador escalar (Figura 4.12(b)).

En dicha figura se comprueba cómo hay una gran cantidad de estados en el cuantificador que nunca serían utilizados, porque no hay datos de la señal de entrada que pudieran cuantificarse a esos niveles; es decir, se da el problema de la celda vacía para un alto número de niveles. Esto provoca que haya muchas celdas vacías y celdas con una alta densidad de estados en ellas. Como consecuencia, la distorsión media se mantiene muy alta (cerca de 200). Además, se comprueba que si se intenta realizar un aprendizaje con este cuantificador, el agente no adquiere ningún comportamiento.

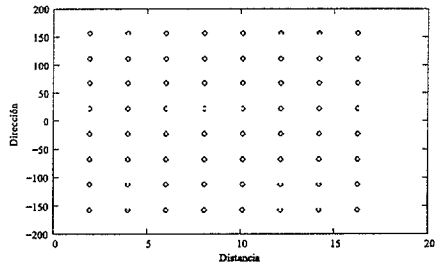
Para diseñar el cuantificador vectorial, se han utilizado el algoritmo GLA definido en la sección 4.2. Así, se han obtenido discretizaciones del espacio de estados de distintos tamaños, como se mostrará a continuación. El único parámetro reseñable es el de ϵ, que se ha fijado a 0,5. En la Figura 4.13 se muestra la distorsión media de la entrada para distintos niveles de cuantificación.

En esta figura se ha representado en el eje de abscisas el logaritmo en base dos del número de niveles de cuantificación, mientras que en el eje de ordenadas se ha representado la distorsión obtenida. Se comprueba cómo la distorsión media desciende drásticamente desde un valor de más de 1000 con un solo nivel de cuantificación hasta prácticamente 0. Además se comprueba que la reducción en los niveles de cuantificación es muy fuerte hasta 64 niveles ($\log_2 N = 6$), punto a partir del cual la curva es cada vez más horizontal.

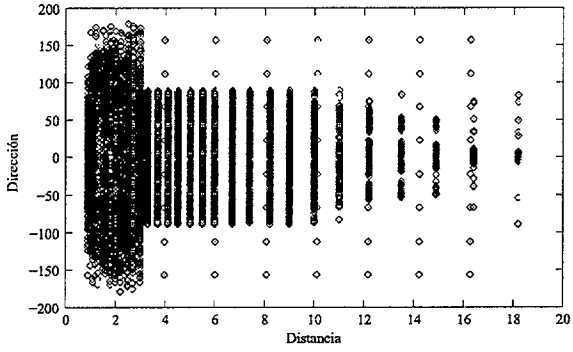
[3] En la Figura 4.12(b), cada punto que se ha pintado representa realmente 64 puntos, ya que se cuantifican también los parámetros de cambio de dirección y cambio de distancia.



(a) Datos Originales



(b) Discretización Uniforme



(c) Datos Originales y Discretización Uniforme

Figura 4.12: Datos originales y discretizaciones uniformes.

Una vez diseñada la discretización del espacio de estados, falta utilizar esta discretización para aprender la política de acción. En la figura 4.14 se muestran los resultados obtenidos para distintos niveles de cuantificación, o lo que es igual, para distintos tamaños de la tabla Q.

Se observa en esta figura que para 32 niveles ($N = 5$), el comportamiento del agente no supera el de un agente que siempre se dirige hacia el balón. Se puede considerar que para 32 niveles, aún no se ha aprendido prácticamente nada. Sin embargo, se observa cómo la curva va ascendiendo según aumenta el número de niveles del cuantificador, obteniendo ya un 50 % de efectividad para 256 niveles de cuantificación ($N = 8$). La curva sigue subiendo, obteniendo valores de más de un 60 % para 1024 y 2048 niveles de cuantificación. Por tanto, el modelo ha sido muy efectivo para resolver el problema de la generalización de estados, ya que ha permitido al agente aprender la habilidad a un nivel bastante alto.

Aunque inicialmente pudiera parecer que aumentando continuamente el número de niveles del cuantificador se obtendrían mejores comportamientos del agente, esto no ocurre así. Se comprueba en la curva que para 4096

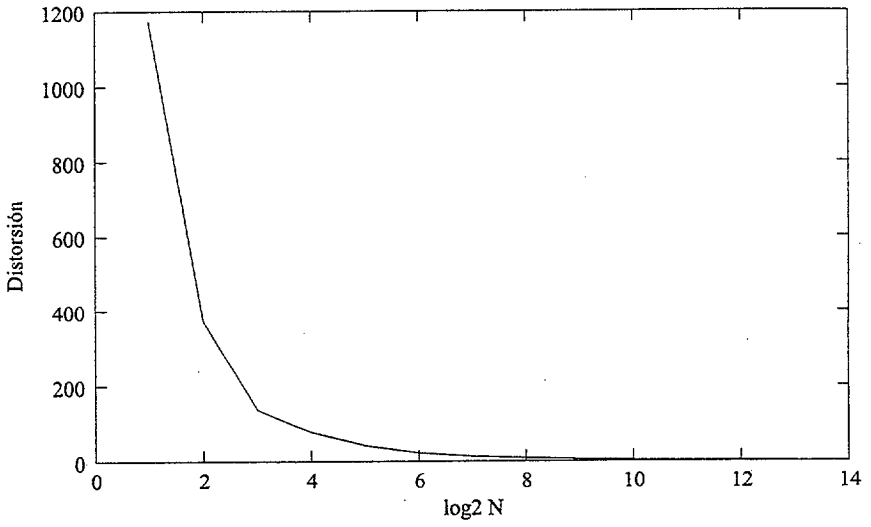


Figura 4.13: Evolución de la distorsión media de la entrada dependiendo del número de niveles de cuantificación.

y 8192 niveles de cuantificación, el comportamiento del agente comienza a degradarse, a pesar de que los errores de cuantificación son menores, tal y como se mostró en la Figura 4.4.1. Esto es debido a que el aprovechamiento de la experiencia vuelve a ser escaso, debido al alto número de estados. En [Fernández, 1999] se puede encontrar una descripción más detallada de la experimentación con VQQL en el dominio de la *RoboCup*, cuyos resultados se encuentran publicados también en [Fernández and Borrajo, 2000]

4.4.2. *CMOMMT*

El dominio de cooperación en el seguimiento de múltiples objetivos móviles, *CMOMMT*, fue presentado en la sección 2.7.3. No obstante, este problema puede ser redefinido como un problema de aprendizaje por refuerzo retardado en el tiempo, definiendo los siguientes elementos:

Datos de Entrada. En el dominio *CMOMMT*, los principales datos que se conocen son la localización de los objetivos a seguir y la localización de otros robots. Sin embargo, en cada momento, sólo se dispone de un conocimiento parcial, dado que la información sobre los objetivos, así como la de los robots que están fuera del rango de visión, es desconocida. Por tanto, unas veces puede disponerse de más información, y otras veces de menos, en función de los robots y los objetivos que estén

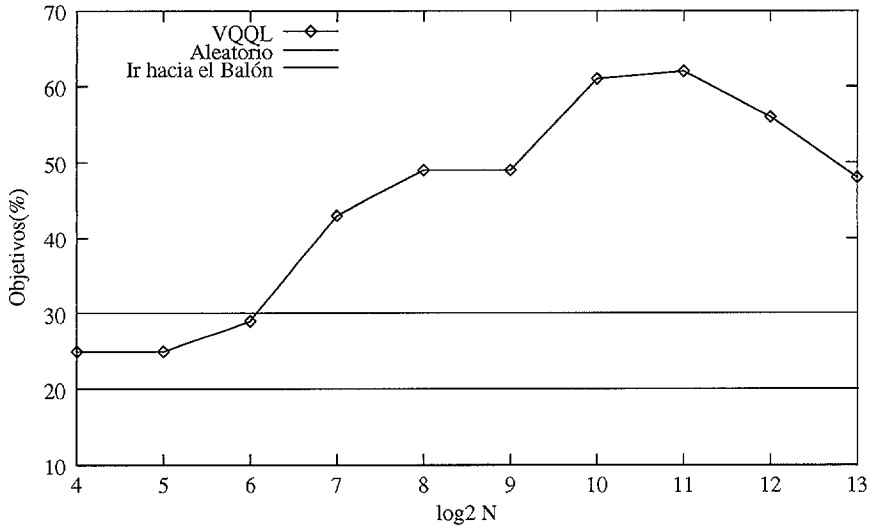
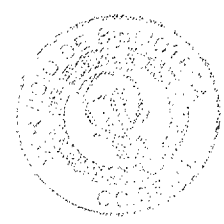


Figura 4.14: Eficiencia del aprendizaje dependiendo del número de niveles del cuantificador.

dentro del rango de visión. Una primera aproximación puede ser, por tanto, utilizar información sólo del objetivo y del robot más cercanos, utilizando una máscara cuando alguno de estos datos no esté disponible. En este caso, el tamaño de los datos de entrada sería 4, correspondientes a las componentes de los vectores de distancia relativa del robot y del objetivo. Variando estos parámetros, es decir, el número de robots y objetivos que se tienen en cuenta, puede variarse la dimensión del espacio de estados.

Las Acciones. Las acciones se discretizan en 8 habilidades, correspondientes con el desplazamiento en dirección de un punto cardinal: "Ir Norte", "Ir Nordeste", "Ir Este", etc. Acciones adicionales podrían ser introducidas si se deseara, como una acción de mantenerse parado. El tiempo que se está realizando este desplazamiento es hasta que la situación cambie. Se considera cambio a una modificación en el estado en el que se encuentra el agente, de entre el conjunto de estados obtenidos tras la discretización del espacio de estados utilizando GLA. Es decir, las transiciones de estado sólo existen cuando se cambia de región en el espacio de estados discretizado.

La Función de Refuerzo. La función de refuerzo puede ser distinta en diferentes experimentos, dependiendo también de la información de en-



trada de que se dispone. En la mayoría de los casos se obtienen refuerzos positivos cuando se están observando objetivos, por lo que a mayor número de objetivos localizados, mayor refuerzo. Como se comentará posteriormente, este refuerzo puede ser reducido si hay otros robots en el campo de visión, ya que se considera que si dos robots no se ven, probablemente estén siguiendo objetivos distintos. Por tanto, se pueden recibir refuerzos negativos si hay otros robots en el campo de visión. Por último, se utiliza una aproximación de aprendizaje por refuerzo retardado en el tiempo, por lo que estos refuerzos sólo son recibidos al final de un intento (de duración fija), y deberán ser propagados adecuadamente.

Los experimentos consisten en aplicar el modelo VQQL al dominio *CMOMMT* con el fin de obtener una gran eficiencia en el seguimiento de los objetivos, siguiendo las medidas de eficacia definidas en [Parker, 1999], es decir, el número medio de objetivos bajo observación en ejecuciones de 1000 ciclos. Al igual que en [Parker, 1999], el rango de visión de cada robot tiene un valor de 1000 unidades, por un radio total del espacio de 5000.

En esta ocasión se ha seguido la versión en línea del modelo VQQL. Se han ejecutado distintos experimentos con el fin de comparar la eficacia del modelo siguiendo dos variables principales. Por un lado, el número total de estados utilizados para representar el espacio de estados total. Por otro lado, si los robots colaboran o no con el fin de obtener mejor rendimiento.

Experimento 1

En este primer conjunto de experimentos, cada estado está compuesto únicamente por las coordenadas x e y del vector distancia desde el robot al objetivo más lejano en su rango de visión. La Figura 4.15 muestra dichas componentes obtenidas tras la exploración inicial definida como primer paso del modelo VQQL. Dicha exploración inicial se ha realizado ejecutando acciones de manera aleatoria, obteniendo un total de 9900 instancias.

En este caso, se observa cómo estos datos de entrada ya introducen información estadística que puede ser explotada por GLA. Por ejemplo, las coordenadas x e y son tales que el módulo del vector distancia se mantiene por debajo del rango de visión del robot, excepto para el punto $(1000, 1000)$, que es utilizado como máscara cuando el robot no ve ningún objetivo.

Cuando se aplica el algoritmo GLA para obtener una representación menor de toda esta información, se pueden generar espacios de estados de distintos tamaños. La Figura 4.16 muestra la evolución de la distorsión media para los conjuntos de entrenamiento y de test (compuestos por un 80 % y

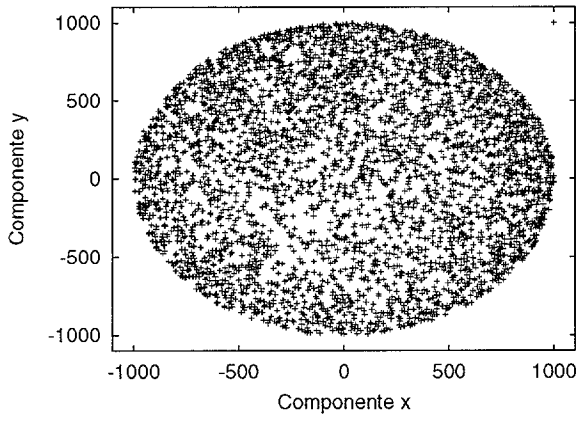


Figura 4.15: Vector distancia desde el robot hasta el objetivo más lejano en su rango de visión.

un 20 % del total de los datos, respectivamente), y cómo en ambos casos, la distorsión media obtenida es menor que 1000 para alfabetos o espacios de estados de 256 o más elementos.

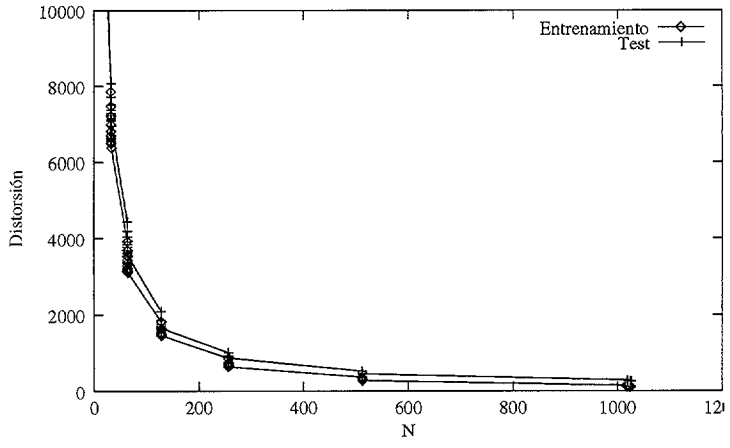
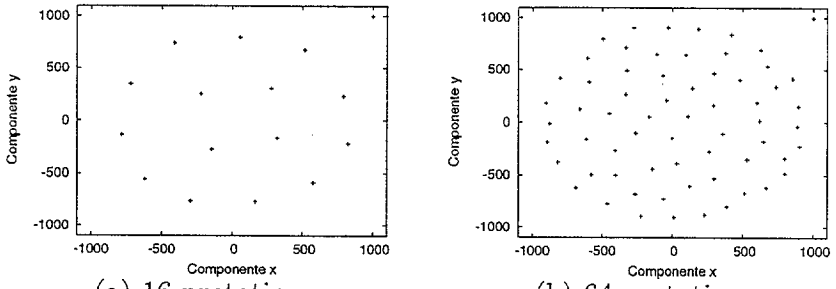


Figura 4.16: Evolución de la distorsión media versus tamaño del espacio de estados.

La Figura 4.17 muestra la representación obtenida con un alfabeto de 16 prototipos, y otro con 64 prototipos, es decir, un dominio discretizado a 16 y 64 estados respectivamente. Se puede ver cómo esta representación está adaptada a los datos de entrada mostrados en la Figura 4.15, incluyendo

un estado para el punto (1000, 1000).



(a) 16 prototipos (b) 64 prototipos

Figura 4.17: Prototipos de las Representaciones de los Espacios de Estados Obtenidos.

Dada estas representaciones, el siguiente paso es aprender la política de acción de los robots. Al igual que en el resto de experimentos, se han mantenido las 8 acciones introducidas anteriormente, siguiendo los puntos cardinales extendidos. La función de refuerzo retardada es el número de objetivos bajo el rango de visión al final de cada ejecución. El número de objetivos es 10, mientras que el número de robots es 1 en la fase de aprendizaje, y 10 en la de test, en la que utilizarán la misma política aprendida. En este caso, por tanto, no se introduce ninguna estrategia de colaboración entre los robots, y los refuerzos positivos sólo se obtienen al final de cada ejecución, en función de los objetivos que se encuentran en el rango de visión en ese momento. Además, si el robot pierde todos los objetivos, recibe un refuerzo negativo, y se mantiene parado hasta que uno de ellos entra de nuevo en su rango de visión. La duración de cada ejecución en la fase de aprendizaje es de 100 ciclos.

La Figura 4.18 muestra los resultados del aprendizaje para diferentes tamaños del espacio de estados, y aprendizajes de distinta duración. En la figura se observa que VQQL obtiene un rendimiento de aproximadamente un 59 % para la mayoría de los casos estudiados. Este valor es ya un incremento sobre aproximaciones previas definidas en [Parker and Touzet, 2000], donde se obtuvo un 50 % con *Pessimistic Lazy Q-Learning*.

En este caso, sólo con 16 estados, los robots obtienen el 59 % de éxito; además, espacios de estados mayores no proporcionan mejores resultados. Otro factor a tener en cuenta es la velocidad de aprendizaje. En espacios de estados pequeños (16 o 64 estados), el aprendizaje es muy rápido, requiriendo sólo entre 100 y 150 estados, mientras que con un mayor número de estados, por ejemplo, 256, el número de ejecuciones necesario para obtener resultados

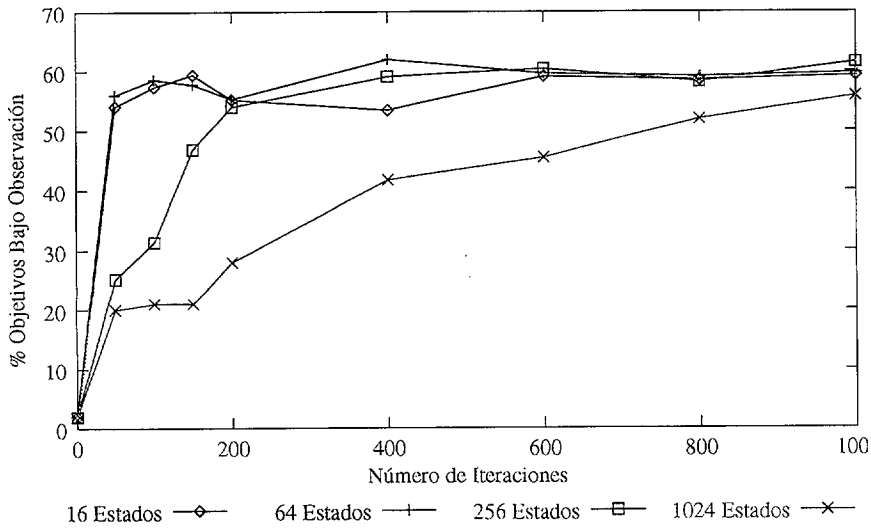


Figura 4.18: Rendimiento de VQQL en el dominio *CMOMMT* para distintos tamaños del espacio de estados.

similares es de 200. Por último, la convergencia en el aprendizaje es difícil de alcanzar, ya que se utiliza un parámetro de aprendizaje fijo de α de 0,05, para un parámetro de descuento de $\gamma = 0{,}6$, ambos obtenidos empíricamente. La convergencia podría ser mejorada con un factor α que se fuese reduciendo con el tiempo.

Llegados a este punto, cabe preguntarse por qué no se alcanzan niveles mayores de eficiencia, y si es posible superarlos. La respuesta puede encontrarse en la Figura 4.19, que muestra un estado de la simulación con el mejor de los resultados anteriores. En dicha figura se muestra cómo los 10 robots están reunidos en tres grupos. Dicha formación hace que el rendimiento de los 10 robots sea prácticamente equivalente a la que se podría dar con únicamente 3 de ellos. Esto es debido a que dado que no existe ningún mecanismo de cooperación entre ellos, ni implícito ni explícito, puede ocurrir que varios robots sigan al mismo objetivo. Y dado que todos siguen la misma política de comportamiento, terminan por agruparse. Esta situación motiva la necesidad de incluir algún tipo de cooperación entre los robots.

Experimento 2

El objetivo de este segundo grupo de experimentos es obtener mejores resultados mediante la introducción de comportamientos cooperativos entre los robots. En este sentido, dado que el aprendizaje se realiza a partir de la

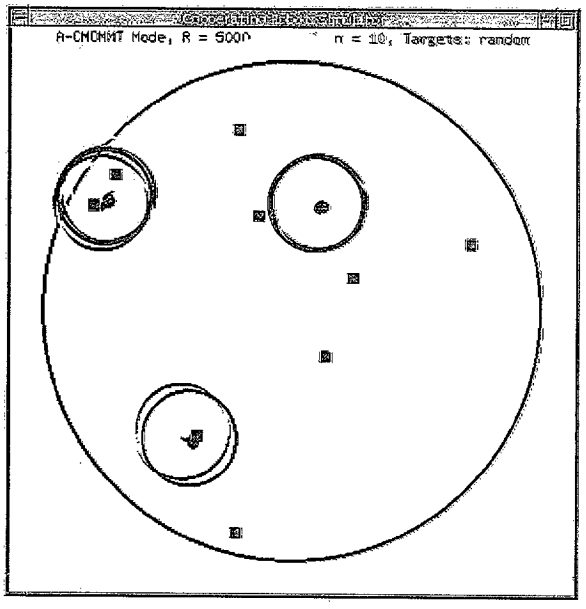


Figura 4.19: Robots agrupados y siguiendo objetivos en CMOMMT.

señal de refuerzo del entorno, la colaboración debe ser incluida implícitamente en dicha señal.

En este caso, la representación del espacio de estados es incrementada para incorporar más información acerca de los objetivos y otros robots. Concretamente, los datos de entrada se componen de la información sobre el objetivo más cercano, sobre el objetivo más lejano, y sobre el robot más cercano. Obviamente, todos ellos dentro de su rango de visión. Por tanto, el espacio de estados se incrementa hasta 6 dimensiones.

Añadir esta nueva información requiere además un cambio en la fase de entrenamiento. Así, en este caso, los 10 robots aprenden simultáneamente una única política de acción, compartiendo la misma tabla Q durante el aprendizaje y durante el test. La implicación de esta aproximación es que el comportamiento es aprendido más rápido (hasta 10 veces, aunque este punto no ha sido estudiado en profundidad).

Además, la señal de refuerzo ahora incorpora un refuerzo negativo que se da al final de cada ejecución con el fin de obtener el comportamiento cooperativo. Este refuerzo negativo se basa en si el robot tiene o no a más robots en su rango de visión. Por tanto, la función de refuerzo para cada robot i al final de cada intento se calcula del siguiente modo (siguiendo la

notación introducida en la sección 2.7.3):

$$r_i(T) = (\sum_{j=1}^{n} b_{ij}(T)) - k(T) \tag{4.10}$$

donde

- $b_{ij}(t) = \begin{cases} 1 & \text{si el robot } v_i \text{ está } \textit{observando} \text{ al objetivo } o_j(t) \text{ en el instante } t \\ 0 & \text{En cualquier otro caso} \end{cases}$
- n es el número de objetivos (10 en este experimento), y
- $k(T)$ es una función cuyo valor es 2 si el robot puede ver a otros robots, y 0 en caso contrario.

La idea básica es que el comportamiento óptimo se obtendrá cuando cada robot siga a un objetivo distinto, situándose lo más separados posible unos de otros. Este refuerzo negativo no asegura esta característica, pero es fácil de entender que podría ayudar. Además, se mantiene el refuerzo negativo que se recibe en caso de que el robot pase a una situación de no ver nada, tal y como se planteó en el experimento anterior.

Los resultados mostrados en la Figura 4.20 confirman las expectativas planteadas. En este caso se han utilizado espacios de estados de mayor tamaño, ya que tenemos una dimensión mayor para dichos estados. Para representaciones del espacio de estados de 64 estados, sólo se alcanza un 40 % medio de objetivos vigilados. Sin embargo, con 256 estados este valor se ve incrementado hasta el 50 %, y para 1024 estados, el 60 % es alcanzado, igualando los resultados del experimento 1. Sin embargo, con 2048 estados, se consigue una mejora de 5 puntos, alcanzando un 65 %, situándose cerca de la mejor solución generada a mano en [Parker and Touzet, 2000].

La Figura 4.21 muestra los robots siguiendo los objetivos usando la política aprendida sobre la representación del espacio de estados de tamaño 2048. En esta imagen se observa cómo sólo un objetivo no está siendo seguido por ningún robot, mostrando que se ha obtenido un comportamiento cooperativo.

La Figura 4.22 muestra los resultados de ambos experimentos, y su comparativa con trabajos anteriores. El primer método comparado es A-CMOMMT, una implementación realizada "ad hoc", basada en vectores de atracción y de repulsión de cada robot hacia los objetivos y hacia otros robots. El resultado de esta aproximación es que cada robot es atraído por los objetivos cercanos, y es repudiado por los robots cercanos, calculando el movimiento de los robots ponderando todas estas fuerzas. El segundo método con el que

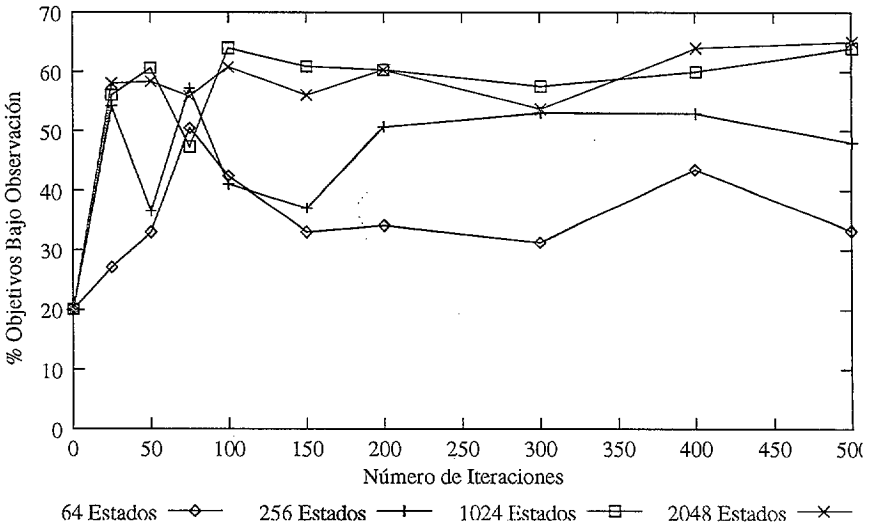


Figura 4.20: Éxito de VQQL en el dominio CMOMMT.

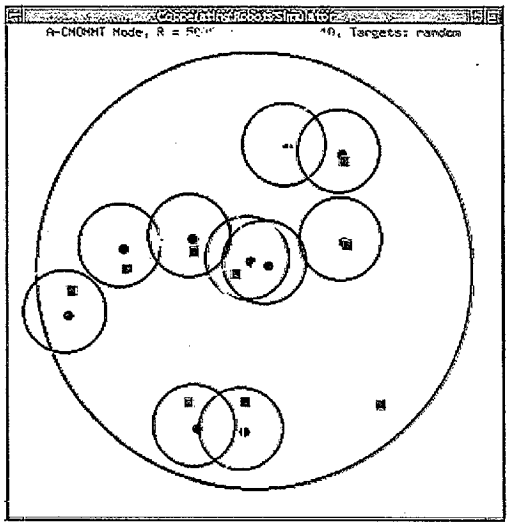


Figura 4.21: Robots cooperando en el seguimiento de objetivos en CMOMMT.

se compara es denominado *Pessimistic Lazy Q-Learning*, que está basado en una combinación de aprendizaje perezoso (*Lazy*) [Aha, 1997], *Q-Learning*, y un algoritmo pesimista para la evaluación de los refuerzos. La primera conclusión que se puede obtener de estos experimentos es que el modelo VQQL es útil para obtener representaciones del espacio de estados que permitan la aplicación de métodos de aprendizaje basados en el modelo sobre conjuntos de estados discretos. En este sentido, con sólo dos atributos, el modelo es capaz de obtener resultados de hasta un 59 % de objetivos vigilados con sólo 16 estados distintos, resultados ya superiores a *Pessimistic Lazy Q-Learning*. Además, el modelo es capaz de aprender en entornos de mayor dimensión, donde seis atributos han sido utilizados para obtener información sobre objetivos y otros robots. En este caso, se obtienen comportamientos cooperativos que emergen únicamente con penalizar a los robots que mantienen a otros robots en su rango de visión, obteniendo un 65 % de éxito, cerca del 71 % alcanzado con la mejor solución obtenida por A-CMOMMT. Se puede concluir que se han logrado comportamientos colaborativos utilizando únicamente información local del entorno, obteniendo mejores resultados que con la versión no colaborativa.

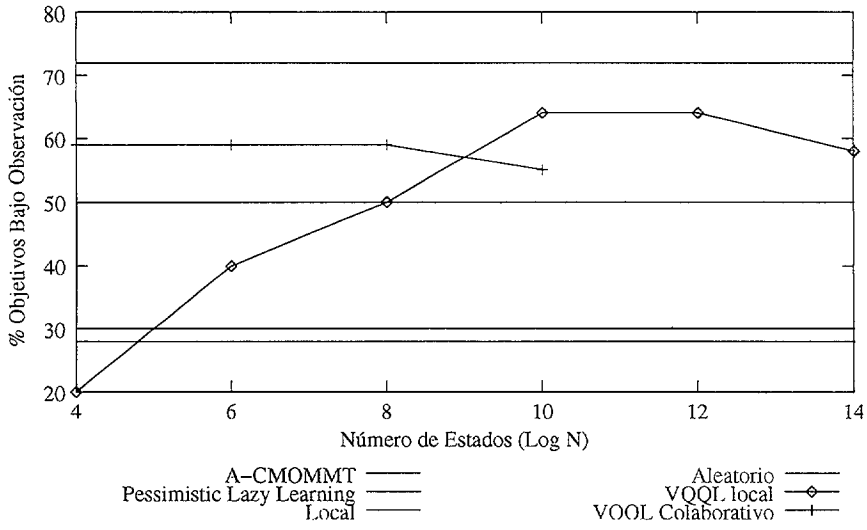


Figura 4.22: Éxito de distintas aproximaciones en el dominio CMOMMT.

No obstante, cabe destacar que encontrar el tamaño adecuado de la discretización es una problemática que dentro del modelo VQQL pasa por probar por un rango de tamaños posibles. Además, este tipo de discretizaciones pueden introducir la pérdida de la propiedad de Markov, que se describe a

continuación.

4.5. Pérdida de la Propiedad de Markov en Espacios de Estados Discretizados

La propiedad de Markov fue introducida en la sección 2.2.2, y se resumía en que la decisión sobre cuál es la acción que debe elegir un agente en un determinado momento depende únicamente del estado en el que se encuentra en ese momento, y no de cómo se haya llegado a ese estado. De la misma forma, de la ecuación 2.2 se derivaba que la probabilidad de que un agente se encuentre en un estado y que reciba un determinado refuerzo sólo depende del estado en el que se encontraba y la acción ejecutada en el instante inmediatamente anterior.

Sin embargo, la discretización del espacio de estados en dominios continuos puede producir la pérdida de esta propiedad [Boyan and Moore, 1995]. Esto se ilustra de forma sencilla siguiendo el ejemplo planteado en la introducción, ilustrado en la Figura 1.1. En 1.1(a) se mostraba un dominio continuo, con una zona de meta, mientras que en 1.1(b) se mostraba una discretización óptima para aprender el problema. Se dice que es una discretización óptima, en el sentido de que mantiene la propiedad de Markov.

Sin embargo, es fácil ver que otras discretizaciones pueden hacer que se pierda la propiedad de Markov. Por ejemplo, si en ese mismo dominio se plantea una discretización 6×6, tenemos el resultado mostrado en la Figura 4.23.

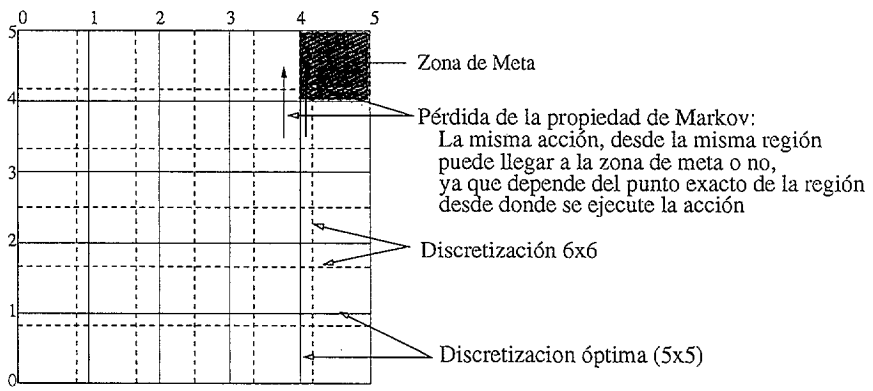


Figura 4.23: Pérdida de la propiedad de Markov al discretizar.

En la figura se observa que la misma acción (ir hacia el norte) ejecutada desde la misma región de la discretización 6×6, recibe distintos refuerzos, ya

que en una llega a la zona de meta, mientras que en la otra no. En este caso, por tanto, se dice que se pierde la propiedad de Markov, ya que el llegar a la zona de meta no sólo depende de la región desde donde se ejecute la acción, sino del punto exacto dentro de la región. Y ese punto exacto depende de estados y acciones ejecutadas anteriormente.

La pérdida de la propiedad de Markov puede verse, no obstante, como una pérdida del determinismo del dominio. Se dice que un dominio es determinista cuando la ejecución de una misma acción, a, desde un mismo estado, s, produce siempre una misma transición de estado, es decir, lleva siempre a un mismo estado s', y obtiene siempre un mismo refuerzo inmediato r. Matemáticamente, y siguiendo la notación introducida en secciones anteriores, se puede decir que un dominio es determinista cuando, dado un estado s, y una acción $a \in A(s)$, se cumple que:

$$\exists s' \text{ tal que } P^a_{ss'} = 1 \text{ y } P^a_{ss''} = 0, \forall s'' \neq s' \tag{4.11}$$

$$R^a_{ss'} = r \text{ siendo } r \text{ constante para } s, s' \text{ y } a \tag{4.12}$$

Un dominio que no cumple las condiciones anteriores se denomina indeterminista o estocástico. La mayoría de los dominios suelen introducir una fuerte componente de indeterminismo, que puede venir dada por gran variedad de factores, como el ruido en los sensores, en las acciones, etc. Pero además, cuando se utilizan técnicas de generalización de tipo teja, o de vecino más cercano, este indeterminismo puede verse multiplicado, o incluso aparecer en dominios que en principio son deterministas, como consecuencia directa de la pérdida de la propiedad de Markov, tal y como se ha mostrado anteriormente. En la figura se observa que la nueva discretización produce un entorno no determinista, ya que la ejecución de la misma acción, desde el mismo estado (en este caso, región) en unas ocasiones produce un refuerzo inmediato, y en otras, otro distinto, dependiendo de si llega o no a la meta.

Este problema podría tratarse en principio utilizando la versión estocástica de las funciones de actualización. Sin embargo, tal y como se introducía en [Boyan and Moore, 1995], y se mostrará a lo largo de este trabajo, esta solución no es adecuada, ya que puede producir políticas de acción subóptimas. Por ejemplo, en el ejemplo de la Figura 4.23, en la región utilizada para mostrar la pérdida de la propiedad de Markov, unas veces conviene ejecutar la acción de ir hacia el norte (si estamos en la zona este de la región) pero otras veces puede convenir ir hacia el este (si estamos en la zona norte).

Por tanto, para alcanzar políticas óptimas, parece adecuado encontrar discretizaciones del espacio de estados que mantengan la propiedad de Markov, o lo que es lo mismo, que mantengan el determinismo del entorno (si

inicialmente era determinista) o que no introduzcan más. Para ello, se considera necesario supervisar la construcción de la discretización con la función de valor que se está aprendiendo.

Esto es fundamental en dominios donde la función de valor tiene fronteras muy definidas, como es este ejemplo. Sus consecuencias principales son la aparición de situaciones de bloqueo o ciclos en la ejecución, como se mostrará en el capítulo 7.