UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

Aprendizaje por Refuerzo para la Toma de Decisiones Segura en Dominios con Espacios de Estados y Acciones Continuos

Autor

Francisco Javier García Polo

Director

Dr. D. Fernando Fernández Rebollo

Departamento de Informática
Escuela Politécnica Superior

Leganés, Noviembre 2012

TESIS DOCTORAL

Universidad
Carlos III de Madrid

TESIS DOCTORAL

Aprendizaje por Refuerzo para la Toma de Decisiones Segura en Dominios con Espacios de Estados y Acciones Continuos

Autor:
Francisco Javier García Polo
Director:
Dr. D. Fernando Fernández Rebollo

Departamento de Informática. Escuela Politécnica Superior

Leganés, Noviembre 2012

TESIS DOCTORAL

Aprendizaje por Refuerzo para la Toma de Decisiones Segura en Dominios con Espacios de Estados y Acciones Continuos

Autor: Francisco Javier García Polo

Director: Dr. D. Fernando Fernández Rebollo

Firma del Tribunal Calificador:

	Firma
Presidente: D.	.....
Vocal: D.	.....
Secretario: D.	.....

Calificación: .....

Leganés, ... de ..... de .....

Índice general

Índice de figuras

Índice de tablas

Resumen

Los problemas de decisión constituyen uno de los campos más fértiles para la aplicación de técnicas de Inteligencia Artificial (IA). Entre todas ellas, el Aprendizaje por Refuerzo ha surgido como un marco útil para el aprendizaje de políticas de comportamiento para la toma de decisiones a partir de la experiencia generada en entornos dinámicos y complejos. En Aprendizaje por Refuerzo, el agente interacciona con el entorno y una función de refuerzo se encarga de indicarle si está haciendo bien o mal la tarea que está aprendiendo. Gran parte del Aprendizaje por Refuerzo se fundamenta en las funciones de valor que proporcionan información acerca de la utilidad de encontrarse en un estado durante un proceso de toma de decisiones, o acerca de la utilidad de tomar una acción en un estado. Cuando se afrontan problemas donde los espacios de estados y acciones es muy grande o incluso continuo, la tradicional representación tabular de la función de valor no es práctica debido al alto coste que exigiría su almacenamiento y su cálculo. En estos casos, es necesaria la aplicación de técnicas de generalización que permitan obtener representaciones más compactas tanto del espacio de estados como del de acciones, de forma que se puedan aplicar eficientemente las técnicas de Aprendizaje por Refuerzo. Además de los espacios de estados y acciones continuos, otro problema importante al que debe hacer frente el Aprendizaje por Refuerzo es minimizar el número de daños (por colisiones, caídas) que se pueden ocasionar en el agente o en el sistema durante el proceso de aprendizaje (e.g., en una tarea donde se trata de aprender a volar un helicóptero, éste puede acabar chocando; cuando se trata de enseñar a andar a un robot, éste puede caerse).

En esta Tesis se plantean dos grandes objetivos. El primero **es cómo afrontar problemas donde los espacios de estados y acciones son de naturaleza continua (por tanto infinito) y de grandes dimensiones.** Una de las opciones se centra en las técnicas de generalización basadas en la discretización. En este Tesis se desarrollan algoritmos que combinan con éxito el uso de aproximación de funciones y técnicas de discretización, tratando de aprovechar las ventajas que ofrecen ambas técnicas. El segundo objetivo que se plantea para esta Tesis es **minimizar el número de daños que sufre el agente o el sistema durante el proceso de aprendizaje en problemas totalmente continuos y de grandes dimensiones**. En esta Tesis se da una nueva definición del concepto de riesgo, que permite identificar estados donde el agente es más propenso a sufrir algún tipo de daño. La consecución de los objetivos planteados implicará además **investigar sobre la utilización de comportamientos base o expertos subóptimos** que permitirán aportar conocimiento sobre la tarea que se trata de aprender, necesario cuando se abordan problemas complejos de grandes dimensiones y donde, además, el agente puede sufrir daños.

Parte I

Introducción y Marco Teórico

Capítulo 1

Introducción

En este capítulo se introduce la Tesis doctoral mediante la descripción del contexto en el que se sitúa el trabajo y de una breve descripción de los objetivos que se pretenden alcanzar. En el capítulo 2 se describe el marco teórico en el que se encuadra esta Tesis para posteriormente, en el capítulo 3, presentar de forma detallada las causas que la han motivado y los objetivos que se pretenden cubrir mediante su desarrollo.

1.1. Contexto e Hipótesis de la Tesis Doctoral

Esta Tesis se enmarca dentro del campo de la Inteligencia Artificial (IA) [Russell and Norvig, 2010] y dentro de éste, en el campo del Aprendizaje Automático [Mitchell, 1997; Bishop, 2006]. La investigación en Aprendizaje Automático se centra básicamente en el desarrollo de algoritmos que permitan a un computador aprender a partir de observaciones. Por ejemplo, el objetivo puede consistir en desarrollar un algoritmo que permita reconocer caras [Turk and Pentland, 1991], o que pueda jugar al backgammon [Tesauro, 1994]. No obstante, esta Tesis se centra en el campo de investigación del aprendizaje por refuerzo [Sutton and Barto, 1998] que puede ser considerado como un conjunto de técnicas enmarcadas dentro del Aprendizaje Automático. En aprendizaje por refuerzo, el agente, a través de su interacción con el entorno y mediante una función de refuerzo que le indica si está haciendo bien o mal la tarea que está aprendiendo, modifica su comportamiento con el fin de maximizar alguna medida de ganancia o utilidad a largo plazo. En aprendizaje por refuerzo, a diferencia que en programación dinámica [Bellman, 1958], se asume un desconocimiento completo del modelo que rige la tarea que se trata de aprender. En otras palabras, las únicas informaciones de que se disponen son las acciones que puede ejecutar el agente y los estados en los que se puede encontrar, pero se desconoce la dinámica del entorno (las probabilidades de transitar de un estado a otro cuando se ha ejecutado una determinada acción, y el refuerzo que se recibe tras ejecutar una acción en un determinado estado). Ante este desconocimiento existen dos posibles aproximaciones. En la primera, donde se engloban los métodos basados en el modelo [Kumar and Varaiya, 1986; Sutton, 1990; Strehl and Littman, 2008], se trata de aprender primero el modelo, para después aplicar técnicas de programación dinámica. En la segunda aproximación, donde se encuentran los llamados métodos libres del modelo, los algoritmos disponibles se aplican sin un conocimiento del modelo [Watkins, 1989]. Esta Tesis se centra exclusivamente en el desarrollo de métodos libres del modelo.

La mayor parte del aprendizaje por refuerzo se fundamenta en lo que se denominan las

funciones de valor que dan información sobre lo valioso que es para el desarrollo de una tarea, el encontrarse en una determinada situación o estado, o lo valioso que es ejecutar una determinada acción, cuando el sistema se encuentra en un determinado estado. Estas funciones generalmente se implementan como tablas y son utilizadas para obtener de forma directa la política de acción que debe guiar el comportamiento del sistema [Sutton and Barto, 1998]. Una de las problemáticas que plantea el uso de los algoritmos de aprendizaje por refuerzo, y que ha sido una de las áreas de investigación más activas en este área, es la elaboración de algoritmos que funcionen en problemas totalmente continuos o con unos espacios de estados y acciones de gran tamaño. Muchos de estos algoritmos utilizan una representación tabular de la política de acción. Dayan y Watkins [Dayan, 1992] probaron que el algoritmo Q-learning, uno de los más intensamente utilizados en aprendizaje por refuerzo, converge con probabilidad 1 a la política de comportamiento óptima bajo los supuestos de unos espacios de estados y acciones discreto y una representación tabular que contemple todos estos estados y acciones [Sutton and Barto, 1998]. Sin embargo, uno de los mayores problemas de los métodos de aprendizaje por refuerzo en su forma tabular es la explosión combinatoria que sucede cuando se aplican en entornos donde los espacios de estados y acciones son de gran tamaño. Esta situación produce una combinación tan elevada de pares *estado* $\times$ *acción* que, aunque en el mejor de los casos fuese posible su almacenaje en memoria, el proceso de aprendizaje necesitaría una enorme cantidad de tiempo para converger. De hecho, en la mayor parte de los casos el problema sería inabordable, y solo en algunos entornos es posible enumerar los estados y las acciones del espacio y almacenar tablas de valores. En el resto de entornos se utilizan *técnicas de generalización*, que permiten representaciones compactas de la información aprendida y transferir el conocimiento entre estados y acciones considerados *similares* [Kaelbling *et al.*, 1996]. Estas técnicas de generalización se dividen básicamente en dos ramas diferentes: técnicas de aproximación de funciones (e.g., redes de neuronas [Stone *et al.*, 2005]), y las basadas en la generalización del vecino más cercano utilizando métodos de discretización (e.g., cuantificación vectorial [Fernández and Borrajo, 1999]).

En cualquier caso, existen pocos trabajos que aborden de forma conjunta el problema de la generalización de los espacios de estados y acciones [Prokhorov and Wunsch, 1997; van Hasselt and Wiering, 2007], proliferando en mayor medida los trabajos que abordan problemas con espacios de estados continuos y espacios de acciones discretos [Fernández and Borrajo, 2002; Goto *et al.*, 2002; Vollbrecht, 2003; Stone *et al.*, 2005; García *et al.*, 2007]. Es indicativo de esto último la competición de aprendizaje por refuerzo en su última edición, donde de los seis dominios existentes (el *acrobot*, el *adversarial tetris*, el *helicopter*, *infinite mario*, *octopus arm* y el *polyathlon*), solo hay dos dominios con los espacios de estados y acciones continuos (el *helicopter*, y el *octopus arm*). En el caso de la generalización conjunta de los espacios de estados y acciones se suelen emplear técnicas de aproximación de funciones para aproximar la función de valor, lo que conlleva, entre otros problemas, el problema de extracción de conocimiento sobre cuál es la mejor acción a ejecutar en cada momento. En cualquier caso, en el capítulo 2, se revisan las propiedades de cada una de estas formas de generalización, y las ventajas e inconvenientes que introducen a la hora de resolver un problema. Además, los algoritmos desarrollados en esta Tesis abordan problemas más grandes que los habitualmente encontrados en la literatura. Si tomamos nuevamente como marco de referencia la competición de aprendizaje por refuerzo, el problema totalmente continuo más grande (el *octopus arm*), que se abordará en esta Tesis junto con otro de mayores dimensiones, tiene un espacio de estados de 82 dimensiones y un espacio de acciones de 32, siendo el único dominio de toda la competición (i.e., considerando dominios continuos

y discretos) que sobrepasa las 6 dimensiones para el espacio de acciones. Por lo tanto, **el primer objetivo de esta Tesis es la obtención de métodos de aprendizaje por refuerzo libre del modelo que sean capaces de aprender en dominios con unos espacios de estados y acciones continuo y de gran tamaño, utilizando para ello métodos de aproximación de funciones** [Stone *et al.*, 2005] **y de discretización basados en el método del vecino más cercano** [Cover and Hart, 1967; Macleod *et al.*, 1987; Aha, 1997]. De hecho, no existen pruebas formales de convergencia cuando el espacio de estados no es discreto, pero se ha aportado evidencia empírica de que la utilización de técnicas basadas en el vecino más cercano, cuando son aplicables, convergen a una política cercana al óptimo [Fernández and Borrajo, 2008a] [1]. Para cumplir este objetivo, se han desarrollado algoritmos que emplean de forma combinada técnicas de discretización y aproximación de funciones para aprovechar los beneficios que ofrecen ambas técnicas, así como algoritmos que emplean únicamente técnicas de discretización para generalizar tanto el espacio de estados como el de acciones. Ambos tipos de algoritmos se emplean para la generalización en dominios con espacios de estados y acciones n-dimensionales y continuos, donde, además, n es un valor más grande que el encontrado habitualmente en la literatura.

Los algoritmos desarrollados en el punto anterior tendrán como objetivo alcanzar los máximos niveles de aprendizaje posibles, sin tener en cuenta si durante este proceso de aprendizaje, y debido a la exploración utilizada y a la naturaleza de la tarea, se alcanzan situaciones de riesgo. No obstante, actualmente existe un auge en la aplicación de aprendizaje por refuerzo en robótica y en problemas reales donde el concepto de riesgo en determinadas tareas juega un papel fundamental. Este riesgo está en la mayoría de los casos asociado con estados no deseados y peligrosos del sistema, resultantes de la ejecución equivocada de acciones, o son el resultado inesperado (dadas las características del dominio) de aparentemente buenas acciones [Geibel, 2001; Geibel and Wysotzki, 2005]. En la tarea de controlar el vuelo de un helicóptero [Abbeel *et al.*, 2007], por ejemplo, los estados no deseados son aquellos en los que el helicóptero se estrella. La ejecución de acciones equivocadas en este dominio conduce inevitablemente a que el helicóptero se estrelle. No obstante, otros factores presentes en el entorno (e.g., el viento), también pueden hacer que aparentemente buenas acciones hagan que el helicóptero se estrelle. El riesgo también está relacionado con la estocasticidad del entorno donde se llevan a cabo las acciones: si la rentabilidad de una inversión, por ejemplo, tiene cierta varianza, una ganancia inferior a la media puede considerarse como un riesgo también (en el sentido de que es menos de lo que se esperaba obtener) [Heger, 1994; Coraluppi and Marcus, 1999; Mihatsch and Neuneier, 2002].

En otras ocasiones, y como se defiende en esta Tesis, la noción de riesgo está relacionada con estados no conocidos a los que se puede llegar durante el proceso de aprendizaje, y en los cuales se desconoce por completo qué acción tomar [García and Fernández, 2011]. En estos estados no conocidos, la decisión de qué acción tomar recae en un experto, que es capaz de hacer que el sistema regrese a una situación segura o conocida. En el caso del dominio del helicóptero, podemos imaginar a un piloto novel manejando los controles y a un piloto experto a su lado que tomará los controles en el caso de que el piloto novel se encuentre ante una situación nueva para él en la que no sepa cómo actuar. El piloto novel quiere mejorar su forma de pilotar, y para ello debe explorar nuevas acciones ante situaciones conocidas. Estas nuevas acciones serán pequeñas variaciones de las que ya conoce, porque sabe que

[1] Puesto que en muchos dominios complejos no se puede garantizar el comportamiento óptimo debido a las técnicas de generalización utilizadas y otros factores como la estocasticidad, en la literatura de aprendizaje por refuerzo es habitual encontrar el término comportamiento cercano al óptimo (*near-optimal behavior/policy*). Véase por ejemplo [Abbeel and Ng, 2005].

variaciones bruscas en los mandos del helicóptero harán que inevitablemente se estrelle. Este proceso de exploración le puede conducir a situaciones no conocidas previamente por él (e.g., el helicóptero está muy cerca del suelo y está a punto de estrellarse). En estas nuevas situaciones le cede los controles al experto, que retornará el helicóptero a una situación estable, punto en el cual cede nuevamente los mandos al piloto novel para que continue con su aprendizaje. El piloto novel toma nota de todas estas nuevas situaciones y de las acciones tomadas por el experto en cada caso, ampliando así su conocimiento. Mediante este proceso de exploración, el piloto novel conseguirá mejorar su estilo de pilotaje (puede incluso mejorar al del experto), y, en el mejor de los casos, habrá conseguido no estrellarse.

El riesgo juega un papel central en Aprendizaje Automático, que a menudo se enfrenta a problemas donde hay que tomar acciones *correctas* (entendiendo por acciones *correctas* a las acciones que no conducen a estados peligrosos) en entornos de decisión complejos basándose en experiencias previas. **El segundo objetivo de esta Tesis es investigar el concepto de riesgo relacionado con el aprendizaje por refuerzo, evitando o minimizando visitar estados peligrosos que dañen al agente o al sistema durante el proceso de aprendizaje**, un problema al que se ha prestado mucha atención en los últimos años. La introducción del concepto de riesgo en los procesos de aprendizaje se ha vuelto más y más relevante para las aplicaciones reales, que cada vez tienen mayor proliferación [Morimoto *et al.*, 2004; Morimoto and Atkeson, 2007; Navarro *et al.*, 2011], y donde es esencial evitar situaciones de riesgo (e.g., que el robot no se caiga mientras aprende a andar, que una empresa no entre en bancarrota). Para ello, en esta Tesis se ha definido una nueva forma de identificar situaciones de riesgo que pueden acabar con daños en el agente o en el sistema de aprendizaje. La función de riesgo propuesta en esta Tesis, independiente del dominio y de la función de refuerzo, no estará basada ni en la varianza del refuerzo acumulado [Heger, 1994; Mihatsch and Neuneier, 2002], ni en la definición de estados/transiciones de riesgo [Geibel, 2001; Geibel and Wysotzki, 2005; Hans, Alexander *et al.*, 2008], sino en la distancia existente entre el espacio de estados conocido por el agente (que almacena en memoria), y el espacio desconocido. Con lo cual, asociamos el concepto de *riesgo* al concepto de *distancia*, considerando que situaciones alejadas y desconocidas por el agente pueden ser peligrosas.

Además, para cumplir los objetivos anteriores, en esta Tesis **se investiga la utilización de comportamientos base o expertos subóptimos conocidos a priori para aprender otros mejores**. En problemas donde el sistema de aprendizaje no tiene conocimiento previo sobre la tarea que se trata de aprender, éste se ve *obligado* a moverse de forma más o menos aleatoria por el espacio de estados y el espacio de acciones con el fin de recuperar suficiente información del entorno. En ocasiones, la exploración guiada por un comportamiento aleatorio puede causar que sólo se explore de forma parcial el espacio, o no se visiten nunca las zonas más relevantes de éste, aquellas zonas donde se encuentran el comportamiento óptimo o cercanos al óptimo. Este hecho se ve agravado en problemas realistas con una alta dimensionalidad, donde no sólo se puede tardar un tiempo prohibitivo en explorar la inmensidad del espacio, sino que además se pueden ejecutar acciones incoherentes que aún siendo válidas no producen los resultados deseados (e.g., contraer todos los músculos al mismo tiempo en el dominio del *octopus arm*). Por tanto, en el primer objetivo planteado en esta Tesis, estos comportamientos base se utilizan para exponer al agente a zonas relevantes del espacio, posibilitando el aprendizaje en problemas de gran tamaño, y acelerando la velocidad de convergencia de los algoritmos. En dominios con riesgo, la exploración del espacio que desencadena la falta de conocimiento inicial puede ser aún peor. Esta exploración en dominios con riesgo es demasiado agresiva y puede acabar dañando al

sistema de aprendizaje o al entorno. Por lo tanto, en el segundo objetivo planteado en esta Tesis, los comportamientos base se emplean para una doble labor. Por un lado, proveer al sistema de aprendizaje de conocimiento inicial acerca de la tarea que se trata de aprender. De esta forma, se indica al agente cómo debe comportarse inicialmente en el entorno. Por otro lado, durante el proceso de exploración posterior, el comportamiento base proporciona un apoyo que permite al agente regresar a una situación segura cuando se visitan situaciones consideradas peligrosas.

1.2. Organización de la Tesis

Esta Tesis doctoral se encuentra dividida en cinco grandes bloques. El primer bloque *I Introducción y Marco Teórico*, al que pertenece este primer capítulo de introducción, contiene además el capítulo 2 donde se repasan los principales conceptos del aprendizaje por refuerzo haciendo especial hincapié en las técnicas existentes para cada uno de los objetivos planteados: técnicas de generalización empleadas en la resolución de problemas totalmente continuos, y técnicas de aprendizaje por refuerzo con riesgo. Este capítulo se acompaña de una discusión final sobre las técnicas existentes en cada uno de los objetivos planteados, en la que se analizan sus limitaciones e inconvenientes y las posibles oportunidades de mejora. De esta forma, el marco teórico en el que se encuadra esta Tesis doctoral, y que se comenzó a esbozar en este capítulo, queda totalmente dibujado.

El segundo bloque *II Objetivos y Evaluación de la Tesis Doctoral* contiene el capítulo 3 donde, una vez analizados los principales métodos presentes en la literatura para abordar problemas continuos, y las diferentes formas de considerar el riesgo en aprendizaje por refuerzo, se resumen las deficiencias encontradas que sirven para motivar el desarrollo de esta Tesis, y que a su vez marcan los objetivos que se pretenden alcanzar. En el capítulo 4, también dentro de este segundo bloque, se detallan los algoritmos con los que se han comparado los algoritmos propuestos en esta Tesis, y se describen los dominios de experimentación donde se ha llevado a cabo esta comparación.

En el tercer bloque *III Métodos* se describen y evalúan los algoritmos propuestos en esta Tesis. Está compuesto por dos capítulos, cada uno dedicado a uno de los objetivos planteados. En el capítulo 5 se describen los algoritmos G-VQQL y CMAC-VQQL propuestos para la resolución de problemas de grandes dimensiones con espacios de estados y acciones continuos. En el capítulo 6 se describe el algoritmo PI-SRL para la resolución de problemas con riesgo en aprendizaje por refuerzo. Ambos capítulos se acompañan de una extensa evaluación de los algoritmos propuestos, junto con una discusión final donde se analizan sus propiedades, su aplicabilidad, y sus aportaciones pero también sus limitaciones y sus inconvenientes.

El cuarto bloque *IV Conclusiones y Líneas Futuras* contiene el capítulo 7 donde se muestran las principales conclusiones derivadas de la investigación llevada a cabo durante la realización de esta Tesis doctoral, y un resumen de las principales aportaciones. Por último, en el capítulo 8, también en este bloque, se resumen las líneas de investigación y trabajos que se plantean para el futuro.

En el quinto y último bloque *V Apéndices*, se encuentra en primer lugar el apéndice A con experimentación adicional en el dominio del simulador empresarial SIMBA empleado en esta Tesis, en segundo lugar el apéndice B con la descripción de los comportamientos base utilizados, y por último el apéndice C con el listado de las publicaciones derivadas de esta Tesis.

Capítulo 2

Marco Teórico

En este capítulo se realiza un repaso al marco teórico en el que se encuentra encuadrada esta Tesis. En primer lugar se da una breve introducción a la toma de decisiones automática en IA, que sirve de base para después hacer una descripción formal del problema mediante MDPs. Dado que la Tesis consta de dos grandes objetivos, aprendizaje por refuerzo en espacios de estados y acciones continuos, y aprendizaje por refuerzo con riesgo, en este capítulo se da una perspectiva teórica a cada uno de estos temas.

2.1. Toma de Decisiones Automática

La toma de decisiones consiste en dar respuesta a la pregunta: *"Si yo me encuentro en esta situación, ¿qué decisión debería tomar?"*. En esta Tesis se trata de dar respuesta a esta pregunta teniendo en cuenta que "situación" es un estado de entre un conjunto infinito de estados, "decisión" es una acción de entre un conjunto infinito de acciones, "debería" consiste en maximizar una medida de refuerzo a largo plazo, y el "yo" hace referencia al sistema de aprendizaje o agente [Sutton and Barto, 1998]. Dependiendo del entorno en el cual el agente se haga la pregunta, del conocimiento que tenga de este entorno y del objetivo que persiga (i.e., si se trata de maximizar una utilidad o se trata de alcanzar una meta), el problema de decisión se puede modelar de una forma u otra. En aprendizaje por refuerzo la información a la que tiene acceso el agente es vía percepción/acción. El agente es responsable tanto de extraer la información del entorno, aprender una política de comportamiento a través de esta información, y comportarse de acuerdo a la política aprendida. Además, el agente elige cada acción para maximizar una medida de utilidad (refuerzo) a largo plazo.

En los problemas de decisión secuenciales (*Sequential Decision Problems*, *SDPs*) aparecen cuatro componentes fundamentales [Barreto *et al.*, 2010]: el agente encargado de tomar las decisiones, el entorno con el cual interactúa, el comportamiento que exhibe, y los refuerzos que recibe. Aunque los problemas de decisión secuencial pueden ser tratados en diferentes niveles de abstracción, en el modelo considerado en este trabajo, un *agente* simplemente es el sistema responsable de interactuar con el mundo y tomar las decisiones. De forma general, el *entorno* sería todo aquello externo al agente. El entorno cambia de estado en estado en respuesta a las acciones ejecutadas por el agente de acuerdo a una dinámica, que en la mayoría de los casos es de naturaleza estocástica (aunque podría ser determinista). La interacción entre el agente y el entorno se realiza en intervalos discretos de tiempo. Las acciones de los agentes, además, sirven a un propósito en los problemas que se consideran

aquí: el propósito de maximizar un *refuerzo*. Todas estas decisiones responden a un comportamiento llevado a cabo por el agente, una *política*. Atendiendo a estas definiciones, un problema de decisión secuencial puede ser clasificado como:

- *Markoviano/No Markoviano*: En un problema de decisión de Markov las transiciones y las recompensas dependen únicamente del estado actual y la acción seleccionada por el agente [Puterman, 1994]. Otra forma de decir esto mismo es que un estado de Markov contiene toda la información relativa a la dinámica de una tarea: una vez conocido el estado actual, la historia de las transiciones que llevaron al agente hasta esta posición es irrelevante a efectos de la toma de decisiones del problema. En el ajedrez por ejemplo, una configuración particular de la partida en cualquier momento proporciona toda la información necesaria para realizar el próximo movimiento. Por otro lado, al decidir si procede o no conceder el empate al adversario, el jugador podría beneficiarse sobre la historia de los juegos anteriores. En este caso, nos encontraríamos con un problema de decisión no markoviano. Los algoritmos de aprendizaje por refuerzo que se consideran en esta Tesis están desarrollados basándose en esta asunción Markoviana. No es difícil ver que los algoritmos de aprendizaje por refuerzo son particularmente sensibles a la propiedad de Markov: si la dinámica de la tarea depende de la historia de transiciones ejecutada por el agente, tiene poco o ningún sentido asociar *estado - acción* con una secuencia específica de recompensas. Esto no quiere decir que sea imposible emplear algoritmos de aprendizaje por refuerzo a tareas no markovianas, aunque actualmente representan un verdadero obstáculo para los algoritmos de aprendizaje por refuerzo en general.

- *Determinista/Estocástico*: En un problema de decisión determinista la ejecución de una determinada acción, a_t, en un determinado estado, s_t, siempre lleva al agente a un mismo estado, s_{t+1}. En contraste, en un entorno estocástico, cada transición está asociada con una distribución de probabilidad sobre el espacio de estados S, es decir, el agente podría acabar en estados diferentes en dos ejecuciones distintas de la misma acción, a, en s_t. El ajedrez, por ejemplo, es un juego determinista mientras que el *blackjack* es estocástico [Sutton and Barto, 1998].

- *Pequeño/Grande*: Aquí los términos *pequeño* y *grande* se refieren al tamaño de los espacios de estados y acciones. Si para un problema de decisión secuencial se pueden almacenar cada par *estado - acción* del problema, teniendo en cuenta la capacidad de almacenamiento de los computadores actuales, el problema puede considerarse *pequeño*. En caso contrario, el problema se considera *grande*. Obviamente, un problema de decisión secuencial con unos espacios de estados y acciones continuos es siempre *grande* [Barreto *et al.*, 2010].

- *Episódico/Continuo*: En una tarea episódica, la interacción entre el agente y el entorno está dividida en episodios. Cada episodio comienza en un estado inicial y termina en un estado especial llamado estado terminal (*terminal state*) [Sutton and Barto, 1998]. Los distintos episodios se van ejecutando secuencialmente. En problemas continuos, no existe la división en episodios y la tarea continua sin un criterio claro de interrupción.

- *No Generalizado/Generalizado*: En un problema de decisión secuencial no generalizado la dinámica del entorno es fija, es decir, las transiciones que gobiernan al agente y la función de refuerzo no cambian a lo largo del tiempo. Los problemas de decisión generalizados se caracterizan precisamente por lo contrario, por cambios en el

entorno, lo que significa que el rendimiento de la política de decisión puede cambiar a lo largo del tiempo. De hecho, el problema de decisión secuencial podría ser diferente en los diferentes pasos/episodios del proceso de aprendizaje. Un dominio de decisión generalizado queda formalmente definido por la tupla $\mathcal{G} =< \Theta, \mathcal{P} >$ donde: Θ es el conjunto de todos los posibles problemas de decisión secuencial y $\mathcal{P}$ es la distribución de probabilidad sobre el conjunto de posibles problemas [Whiteson *et al.*, 2009].

En general los problemas de decisión secuencial que se considerarán en el contexto de esta Tesis son **markovianos, estocásticos, grandes, episódicos y generalizados** (aunque también se considerarán problemas **no generalizados**).

2.2. Aprendizaje por Refuerzo

Una de las ideas más utilizadas en el campo del aprendizaje inductivo es el aprendizaje supervisado a partir de ejemplos [Mitchell, 1997]. En dicho aprendizaje se suministra al sistema pares de la forma *(entrada - salida deseada)* de forma que ante nuevas entradas desconocidas el sistema sea capaz de predecir la salida deseada. La estrategia general de aprendizaje por refuerzo utiliza un enfoque diferente al de las técnicas supervisadas. El aprendizaje por refuerzo [Sutton and Barto, 1998] es un área dentro del aprendizaje automático cuya meta es encontrar una política que mueva un agente de forma óptima por el entorno, del que generalmente se asume que es un Proceso de Decisión de Markov (MDP). Muchas aproximaciones de aprendizaje por refuerzo se han utilizado de forma satisfactoria en tareas importantes (e.g., control de robots [Smart and Kaelbling, 2002; Hester *et al.*, 2011], juegos estocásticos [Mannor, 2004; Konen and Bartz-Beielstein, 2009], y optimización de sistemas de control complejos y dinámicos [Salkham *et al.*, 2008]). En el aprendizaje por refuerzo hay que definir el algoritmo de forma que se premie o castigue su comportamiento en función de las acciones que se vayan realizando. El modelo de aprendizaje por refuerzo puede entenderse por tanto como un agente conectado al entorno de forma que reciba información del mismo a través de sus sensores. A su vez, el agente es capaz de interactuar con el entorno mediante la realización de determinadas acciones. Estas acciones tendrán repercusiones positivas o negativas que le llegarán en forma de refuerzo y que le harán transitar a otro estado diferente del que se encontraba (figura 2.1) [Sutton and Barto, 1998; Kaelbling *et al.*, 1996].

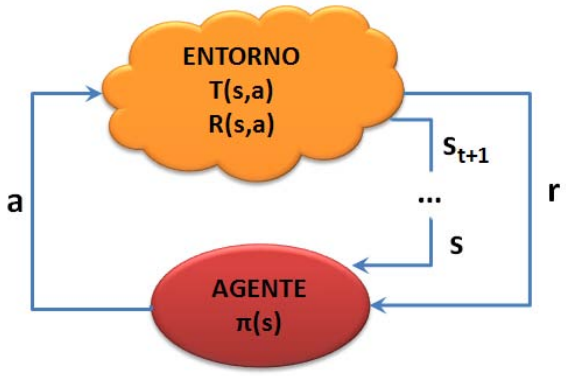


Figura 2.1: Modelo de aprendizaje por refuerzo

En cada instante, el agente recibe del entorno el estado en el que se encuentra, s. Es entonces cuando el agente, a través de esta información, decide ejecutar una acción determinada, a, que generará como salida. La ejecución de esta acción cambia el estado que es percibido por el agente a s', y que es recibido por el agente junto con una señal de refuerzo, r. Este refuerzo informa al agente de la utilidad de ejecutar la acción a cuando estaba en el estado s para lograr un objetivo concreto. El agente sigue un comportamiento determinado decidiendo en cada momento las acciones que ha de ejecutar. A este comportamiento o política lo denotaremos por π. La política del agente, π, debe ser tal que elija acciones que incrementen la suma de todas las señales de refuerzo recibidas a lo largo del tiempo, siguiendo algún criterio de optimalidad.

2.2.1. Procesos de decisión de Markov

Los denominados procesos de decisión de Markov (MDPs) se pueden considerar como un tipo de problema de decisión secuencial donde se cumple la propiedad de Markov [Bellman, 1958; Howard, 1960; Puterman, 1994; Boutilier *et al.*, 1999]. Son el modelo más utilizado para definir los problemas de aprendizaje por refuerzo. Siguiendo una notación similar que la ofrecida por Sutton y Barto [Sutton and Barto, 1998], un MDP queda definido mediante la tupla $< S, A, T, R >$ de forma que:

- S: Se trata de un conjunto de estados, donde $s_t \in S$ es el estado en el que se encuentra el agente en el momento t.
- A: Es un conjunto de acciones que el agente dispone cuando se encuentra en el estado s_t, donde $a_t \in A(s_t)$ denota la acción que el agente ejecuta en el instate t cuando se encontraba en el estado s_t, siendo $A(s_t)$ el conjunto de acciones disponibles por el agente en el estado s_t.
- $T : S \times A \to P(S)$: donde cada miembro de P(S) es una distribución de probabilidad sobre el conjunto S. $T(s, a, s')$ es la probabilidad de que se transite del estado s a s' ejecutando la acción a.
- $R : S \times A \to \mathbb{R}$: *R(s, a)* es el refuerzo recibido tras ejecutar la acción a cuando se estaba en el estado s. El refuerzo que recibe el agente tras ejecutar la acción a_t en s_t puede contener ruido, y se denota por r_{t+1}.

La tarea del agente es encontrar la política, $\pi : S \times A \to [0, 1]$, donde $\pi_t(s, a)$ denota la probabilidad de que el agente seleccione la acción a en el estado s, de forma que se maximice alguna medida de refuerzo.

DEFINICIÓN 2.1 ***Política Determinista***. *Una política determinista π es una política donde $\forall s : \pi(s, a) = 1$ para exactamente una acción $a \in A(s)$ y $\pi(s, b) = 0$ para el resto de acciones $b \in A(s)$.*

$$\forall s \; \exists a : \; \pi(s, a) = 1 \mid \forall b \neq a \; \pi(s, b) = 0 \tag{2.1}$$

DEFINICIÓN 2.2 ***Política Estacionaria***. *Una política estacionaria es una política que no cambia a lo largo del tiempo, es decir, donde $\forall t : \pi_t = \pi$.*

La acción que ejecuta el agente depende en cada momento únicamente del estado en el que se encuentra y no de las acciones pasadas ya realizadas, por lo que parece interesante definir una señal de estado que resuma de forma adecuada el pasado ocurrido. Por lo tanto, no importa las acciones que el agente haya ejecutado hasta el momento, porque es el estado actual lo único necesario para el agente en la toma de la decisión de la próxima acción a realizar. 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* tal y como se introdujo anteriormente. Matemáticamente esta propiedad se describe cómo muestra la ecuación 2.2.

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

Lo que viene a decir esta ecuación, es que la probabilidad de que un agente en un instante $t + 1$ se encuentre en un estado y reciba un refuerzo concreto sólo depende del estado en el que se encontraba y de la ejecución de la acción en dicho estado en el instante t. Formalmente, la propiedad de Markov queda descrita en la definición 3.

DEFINICIÓN 2.3 ***Propiedad de Markov***. *Un proceso estocástico tiene la propiedad de Markov si la probabilidad condicional del siguiente estado del proceso, s_{t+1}, depende sólo del estado en el que se encontraba el agente, s_t, y la acción ejecutada en dicho estado, a_t.*

Un MDP finito se define mediante su conjunto de estados, acciones y las funciones de transición de estados y de refuerzo. La función de transición de estados se define como muestra la ecuación 2.3.

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

donde Pr define la probabilidad que se comentaba anteriormente. Como la función T define una distribución de probabilidades, se cumple que $\forall s \in S, \forall a \in A, \sum_{s_i \in S} T(s, a, s_i) = 1$. La función de refuerzo viene definida por la siguiente ecuación 2.4.

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

donde E define una esperanza sobre el valor a recibir.

DEFINICIÓN 2.4 ***MDP finito***. *Un MDP finito es un MDP con un conjunto finito de estados y acciones: $|S| < \infty$ y $|A| < \infty$.*

Habitualmente el proceso de aprendizaje en aprendizaje por refuerzo se organiza en episodios. Un episodio consiste en una secuencia de pasos que van desde un estado inicial hasta un estado final o terminal.

DEFINICIÓN 2.5 ***Estado terminal*** *[van Hasselt and Wiering, 2007]. Un estado terminal o absorbente es un estado donde todas las acciones le hacen transitar a sí mismo con probabilidad 1, con un refuerzo igual a 0. Además, cuando se alcanza, el episodio finaliza.*

Existen problemas en los cuales la propiedad de Markov no es válida para todos los estados y acciones observables. Estos problemas normalmente se modelan como parcialmente observables MDPs (POMDPs) [Sondik., 1971; Smallwood and Sondik, 1973; Monahan, 1982].

Los POMDPs asumen que hay un MDP que describe el problema en cuestión, sin embargo el agente no puede observar el estado completo. Por el contrario el agente recibe en cada instante de tiempo t una observación $o_t \in O$. Debido a que los algoritmos más utilizados en aprendizaje por refuerzo asumen la existencia de la propiedad de Markov, hay algoritmos específicos para los POMDPs [Lovejoy, 1991; Jaakkola *et al.*, 1995; Parr and Russell, 1995; Kaelbling *et al.*, 1998]. No obstante, este tipo de algoritmo quedan fuera de los objetivos de esta Tesis.

2.2.2. Funciones de valor

La mayoría de los algoritmos de aprendizaje por refuerzo tratan de obtener la política de acción π mediante la aproximación de las funciones de valor. Estas funciones evalúan, para una determinada política, cómo de bueno es para el agente estar en un estado o cómo de bueno es ejecutar una acción desde un estado. Existen por tanto dos funciones de valor, la función de valor-estado (*state-value*), y la función de valor-acción (*action-value*).

2.2.2.1. Función de Valor-Estado

La función de valor para un estado s y una política π, $V^\pi(s)$, es el refuerzo que se espera obtener si seguimos la política π a partir del estado s. La función queda matemáticamente descrita 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_π denota el valor esperado dado que el agente sigue una política π.

2.2.2.2. Función de Valor-Acción

De forma parecida se puede definir la función de valor-acción $Q^\pi(s, a)$ como el refuerzo que se espera recibir si seguimos una política de acción π tras ejecutar la acción a en el estado s. La expresión matemática para esta función se muestra en la ecuación 2.6.

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

Además, siempre hay una o más políticas que son mejores que el resto y que nos hacen conseguir mejores resultados. A estas políticas se las denomina políticas óptimas y se denotan por π^*. Por lo tanto, la función de valor-estado óptima se definiría como queda reflejado en la ecuación 2.7.

$$V^*(s) = \max_\pi V^\pi, \forall s \in S \tag{2.7}$$

En cambio, la función de valor-acción óptima quedaría como muestra la ecuación 2.8.

$$Q^*(s, a) = \max_\pi Q^\pi(s, a), \forall s \in S, \forall a \in A \tag{2.8}$$

Lo que hace tan útiles a estas funciones de valor óptimas es que a partir de ellas puede obtenerse fácilmente la política óptima. En el caso de la función valor-acción óptima la política óptima podría obtenerse como se muestra en la ecuación 2.9.

$$\pi^*(n) = \arg_a \max Q^*(s,a) \tag{2.9}$$

2.2.3. Métodos de resolución

En aprendizaje por refuerzo pueden darse dos circunstancias a la hora de abordar un problema. Por un lado puede que se tenga un conocimiento completo del MDP y se conozcan el conjunto de estados, acciones y las funciones T y R. En este caso se podrán utilizar técnicas de programación dinámica muy estudiadas y desarrolladas matemáticamente [Bellman, 1958]. Por otro lado puede darse el caso de que se desconozca parcialmente o por completo el modelo a tratar. En este segundo caso se pueden seguir dos aproximaciones. En la primera de ellas se trata de aprender primero el modelo para después aplicar técnicas de programación dinámica. Esta aproximación engloba un conjunto de métodos denominados *métodos basados en el modelo*. En el otro extremo se encontrarían los *métodos libres del modelo* que tratan de resolver el problema sin un conocimiento *a priori* del modelo. En las siguientes secciones se dará una pequeña introducción a cada una de estas técnicas haciendo especial hincapié en los métodos libres del modelo, que son en los que se centran los algoritmos planteados en esta tesis.

2.2.3.1. Programación dinámica

La programación dinámica engloba a un conjunto de técnicas y algoritmos que se pueden aplicar para obtener políticas de comportamiento óptimas cuando se tiene un conocimiento completo del MDP. Lo que tratan de conseguir estos algoritmos es el mantenimiento de las funciones de valor $V(s)$ o $Q(s,a)$ de forma que una vez obtenidos los valores óptimos de estas funciones se pueda derivar de ellas la política de comportamiento óptima. Dichas funciones cumplen el principio de optimalidad de Bellman [Bellman, 1958] que se enuncia seguidamente:

DEFINICIÓN 2.6 ***Principio de Optimalidad de Bellman:*** *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.*

Este principio queda matemáticamente formalizado en la ecuación 2.10 y la ecuación 2.11.

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

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

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

La definición de estas funciones tiene forma recursiva. Este paso recursivo hace depender los valores de V^* y Q^* para unos estados de los valores de V^* y Q^* de otros estados

imposibilitando de esta forma el cálculo de estas funciones. Existen no obstante varios algoritmos que permiten resolver la recursividad en el cálculo de las funciones de valor de forma iterativa. A modo de ejemplo, se muestra en la tabla 2.1 el algoritmo de *iteración de la política*. Este algoritmo actualiza de forma sucesiva una política que inicializa arbitrariamente. Consta principalmente de dos pasos. En el primero de ellos calcula el valor de la función *V(s)* de acuerdo a la política que está utilizando en ese momento. Para llevar a cabo esta actualización se basa en la definición recursiva de esta función que se mostraba en la ecuación 2.10. Este paso concluye cuando las modificaciones realizadas en la función valor con respecto a los valores de la iteración anterior no superan cierto parámetro θ. En el segundo paso la política se vuelve a evaluar teniendo en cuenta los valores obtenidos en el paso anterior para cada estado.

Iteración de la política $(S, A) \rightarrow \pi(s)$	
00	**Dados:**
01	El conjunto de estados, S.
02	El conjunto de acciones, A.
03	**1. Inicializar:**
04	La función de valor, $\forall s \in S : V(s) \in \mathbb{R}$.
05	La política de acción inicializada de forma aleatoria, $\forall s \in S : \pi(s) \in A$.
06	**2. Repetir:**
07	**2.1. Evaluación de la política**
08	**Repetir**
09	**Para** $s \in S$ **hacer** ;; Para cada estado $s \in S$
10	$v \leftarrow V(s)$
11	$V(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s)) + \gamma V(s')]$
12	$\Delta \leftarrow max(\Delta, \|v - V(s)\|)$
13	**hasta** $\triangle < \theta$
14	**2.2. Mejora de la política**
15	*politica_estable* $\leftarrow$ *verdadero*
16	**Para** $s \in S$ **hacer** ;; Para cada estado $s \in S$
17	$b \leftarrow \pi(s)$
18	$\pi(s) \leftarrow \arg\max_a \sum_{s'} T(s, a, s')[R(s, a) + \gamma V(s')]$
19	**Si** $b \neq \pi(s)$, entonces *politica_estable* $\leftarrow$ *falso*
20	**hasta** *politica_estable* $\leftarrow$ *verdadero*
21	**3. Devolver** $\pi(s)$

Tabla 2.1: Algoritmo de iteración de la política.

El algoritmo de iteración de la política genera una sucesión de cada vez mejores políticas. Esto implica que en la repetición k $\forall s \in S : V^{\pi_k}(s) \geq V^{\pi_{k-1}}(s)$ [Puterman, 1994; Bertsekas and Tsitsiklis, 1996]. En la práctica, el algoritmo de iteración de la política converge en un número reducido de iteraciones, aunque el tiempo de convergencia es prohibitivamente grande cuando el espacio de estados es grande. Además, este algoritmo requiere un conocimiento de las función de transición T y la función de refuerzo R.

2.2.3.2. Métodos basados en el modelo

En el apartado anterior se introdujeron algunas técnicas que se pueden emplear cuando se tiene un conocimiento completo del modelo. Sin embargo, esto raras veces ocurre, encontrándonos en la mayor parte de los casos con un conocimiento parcial del problema. Ante esta situación, existe la posibilidad de aprender primero la dinámica del problema mediante exploración para después emplear técnicas de programación dinámica. A continuación se

describen brevemente algunos de los algoritmos que se emplean para aprender la dinámica del entorno.

- *Método de Equivalencia de Certeza (Certainty Equivalence Method).* Aprende directamente la dinámica del entorno, es decir, las funciones T y R mediante exploración. Una vez aprendidas estas funciones utiliza técnicas de programación dinámica [Kumar and Varaiya, 1986]. Aunque este método tiene el gran inconveniente de cómo realizar la exploración inicial [Whitehead, 1991].

- *Dyna-Q.* Este algoritmo realiza las fases de aprendizaje de las funciones T y R y ajuste de la política simultáneamente [Sutton, 1990].

Existen otros muchos algoritmos basados en estas ideas como el algoritmo *Priorized Sweeping* [Moore and Atkeson, 1993], *Queue-Dyna* [Peng and Williams, 1993], etc.

2.2.3.3. Métodos libres del modelo

En este tipo de métodos, al igual que en los métodos basados en el modelo, se asume un desconocimiento total de la dinámica del entorno. Las únicas informaciones de que disponen son las acciones que puede ejecutar el agente y los estados en los que se puede encontrar, pero no se conoce la forma en que el agente puede interactuar con el entorno, es decir, desconoce las funciones T y R. En este tipo de técnicas, la forma en que se consideran los refuerzos obtenidos por el agente cuando interactúa con el entorno puede ser diversa, dando lugar a diferentes métodos, algunos de los cuales se describen brevemente a continuación.

- *Métodos de Monte Carlo.* En la interacción del agente con el entorno se pueden extraer secuencias completas de comportamiento desde una situación de partida hasta una situación final. Estas secuencias completas de comportamiento no serían más que la secuencia de recompensas observadas por el agente mientras interactuaba con el entorno.

- *Métodos de diferencia temporal* (*TD Methods*). Estos métodos tratan de utilizar la experiencia paso a paso, o lo que es lo mismo, por cada acción realizada por el agente. La actualización de los métodos de TD la hacen utilizando secuencias de comportamiento cuya longitud viene especificada por un parámetro $\lambda \in [0, 1]$.

Además, dependiendo del uso que se haga de la política que se está aprendiendo distinguimos otros dos tipos de métodos:

- *Métodos on-policy.* Utilizan la política de acción que se está aprendiendo para decidir la interacción que se realiza con el entorno, es decir, se evalúa y mejora la misma política usada para tomar decisiones

- *Métodos off-policy.* La política de interacción con el entorno es independiente de la política que se está aprendiendo, es decir, se evalúa y mejora una política diferente a la usada para tomar decisiones.

Dentro de los métodos de diferencia temporal podemos enmarcar los algoritmos *Q-Learning* y *Sarsa*. El algoritmo *Q-Learning* aprende a partir de la experiencia que se genera durante la exploración del entorno. Es uno de los desarrollos más importantes en aprendizaje

por refuerzo de un algoritmo *off-policy*. La experiencia se representa mediante tuplas de experiencia $< s, a, s', r >$, donde s representa el estado en el que nos encontramos, a es la acción que se ejecuta desde s para llegar a s' y r es el refuerzo obtenido en el proceso. Son estas tuplas de experiencia las que se emplean en el proceso de aprendizaje tal y como se muestra en la tabla 2.2.

Q-Learning $(S, A, K, H, \alpha, \gamma) \rightarrow Q(s, a)$

Dados:
- El conjunto de estados S.
- El conjunto de acciones A.
- Un número máximo de episodios a ejecutar, K.
- Un número máximo de pasos por episodio, H.
- Los parámetros de aprendizaje, α y γ.

1. Inicializar:
- La función de valor-acción arbitrariamente, $\forall s \in S$ y $\forall a \in A : Q(s, a) \in \Re$.

2. Ejecutar:
- **Para** $k = 1$ a K **hacer** ;; Para cada episodio
  - Seleccionar s_1
  - **Para** $h = 1$ a H **hacer** ;; Para cada paso del episodio
    - Seleccionar acción a_h usando la política derivada de Q con algún tipo de exploración
    - Ejecutar la acción a_h y obtener el refuerzo r_{h+1} y el estado siguiente s_{h+1}
    - Actualizar la entrada de la tabla $Q(s_h, a_h)$

$$Q(s_h, a_h) \leftarrow Q(s_h, a_h) + \alpha[r_{h+1} + \gamma max_{a'} Q(s_{h+1}, a') - Q(s_h, a_h)] \tag{2.12}$$

3. Devolver $Q(s, a)$

Tabla 2.2: Algoritmo Q-Learning.

La versión on-policy de este algoritmo se llevaría a cabo sustituyendo la ecuación 2.12 por la ecuación 2.13

$$Q(s_h, a_h) \leftarrow Q(s_h, a_h) + \alpha[r_{h+1} + \gamma Q(s_{h+1}, a_{h+1}) - Q(s_h, a_h)] \tag{2.13}$$

donde s_{h+1} es el estado al que se transita y a_{h+1} la acción que se ejecuta en este estado, dando lugar al algoritmo *Sarsa*. En este caso las tuplas de experiencia que se emplean en el proceso de aproximación de la función Q no tienen la forma $< s_h, a_h, s_{h+1}, r_{h+1} >$ sino que tienen la forma $< s_h, a_h, s_{h+1}, a_{h+1}, r_{h+1} >$, es decir, las actualizaciones se realizan teniendo en cuenta además la acción que se toma en el estado s' al que se ha transitado.

2.2.4. Exploración y explotación

En aprendizaje por refuerzo la exploración es entendida por la ejecución de acciones de forma más o menos aleatoria, de forma que se visiten nuevos estados y por lo tanto nuevas alternativas de solución. En cambio, por explotación se entiende el utilizar la información obtenida en el aprendizaje para maximizar el beneficio.

Para establecer un equilibrio entre exploración y explotación se pueden seguir diversas estrategias. En los extremos de estas estrategias están, por un lado, las estrategias que siguen un comportamiento totalmente aleatorio y, por otro lado, las estrategias denominadas totalmente avariciosas que seleccionan siempre la acción que se supone mejor en un momento determinado. Entre estas dos estrategias se sitúan un amplio abanico de técnicas. Entre estas técnicas destaca $\epsilon - greedy$ donde el parámetro ϵ define la probabilidad de ejecutar la mejor

acción posible en un momento dado y $1-\epsilon$ define la probabilidad de ejecutar acciones aleatorias. Uno de los inconvenientes de la estrategia $\epsilon - greedy$ consiste en que cuando decide explorar (seleccionar una acción aleatoria) elige equiprobablemente entre todas las acciones disponibles. Esto significa que es tan probable seleccionar la mejor acción como la peor. Para solucionar este inconveniente surgen otro tipo de técnicas conocidas como *softmax*, que asignan probabilidades a cada acción en función del valor que tienen para Q. La asignación de probabilidades para cada acción se realiza de acuerdo a la ecuación 2.14.

$$Prob(a_i) = \frac{e^{Q(s,a_i)/\tau}}{\sum_{a_j \in A} e^{Q(s,a_j)/\tau}} \tag{2.14}$$

En la ecuación se incluye el parámetro τ denominado *temperatura*. Altas temperaturas causan que las acciones sean equiprobables. En cambio, bajas temperaturas causan que las diferencias entre las probabilidades de acción sean muy grandes. Se puede comprobar que en el límite $\tau \to 0$, *softmax* se comporta de igual forma que $\epsilon - greedy$.

2.2.5. El parámetro λ

Como se comentó anteriormente, los métodos de Monte Carlo realizan la actualización de las funciones de valor teniendo en cuenta la secuencia completa de recompensas observadas desde que se estaba en un estado inicial hasta que se llega a un estado final. En cambio, la actualización en los métodos *TD(0)* se realizaba utilizando únicamente la siguiente recompensa que se obtenía en la interacción con el entorno. Los métodos $TD(\lambda)$, que se discuten brevemente aquí, son métodos intermedios que actualizan muchos pares estado-acción en cada paso logrando una aceleración importante en la convergencia de los algoritmos.

Tanto los métodos de Monte Carlo como los TD(0) utilizan una actualización en la cual el nuevo valor estimado para la función de valor de un estado se obtiene haciendo que el valor estimado calculado anteriormente tienda a un objetivo en la proporción de una tasa de aprendizaje. Los métodos de Monte Carlo utilizan como objetivo la secuencia completa de recompensas aceptadas tal como se muestra en la ecuación 2.15.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-t-1} r_T \tag{2.15}$$

donde T es el tiempo del último paso del episodio.

En cambio, la predicción en TD(0) utiliza como objetivo la primera recompensa más el valor descontando del próximo estado como se muestra en la ecuación 2.16.

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}) \tag{2.16}$$

Esto tiene sentido porque $\gamma V_t(s_{t+1})$ sustituye a la secuencia $\gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-t-1} r_T$. En este sentido se habla de la ecuación 2.16 como un objetivo truncado después de un paso de tiempo (*n-step target*). El objetivo de dos pasos (*two-step target*) se definiría como se aprecia en la ecuación 2.17.

$$R_t^{(2)} = r_{t+1} + r_{t+2} + \gamma^2 V_t(s_{t+2}) \tag{2.17}$$

En este caso $\gamma^2 V_t(s_{t+2})$ sustituye a la secuencia $\gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... + \gamma^{T-t-1} r_T$. El caso general, el objetivo de n pasos (*n-step target*) puede verse en la ecuación 2.18.

$$\begin{aligned} R_t^n &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \\ &\quad + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}) \end{aligned} \tag{2.18}$$

El algoritmo $TD(\lambda)$ puede ser entendido desde dos enfoques diferentes. Desde el enfoque hacia delante (*forward view*) se puede entender como una actualización de la estimación de la función de valor utilizando como objetivo un valor llamado λ. El valor λ es proporcional a la suma de todos los retornos de n pasos, cada uno de ellos ponderado por λ^{n-1} donde $0 \leq \lambda \leq 1$. El objetivo de $TD(\lambda)$ queda expresado matemáticamente en la ecuación 2.19.

$$R_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^n \tag{2.19}$$

Entonces, según el algoritmo $TD(\lambda)$ se actualiza la estimación de la función de valor siguiendo la ecuación 2.20.

$$V_t(s_t) = V_{t-1}(s_t) + \alpha[R_t^\lambda - V_{t-1}(s_t)] \tag{2.20}$$

Puede comprobarse que si $\lambda = 1$ el algoritmo $TD(\lambda)$ tendría el mismo comportamiento que *Monte Carlo* y que si $\lambda = 0$ entonces $TD(\lambda)$ se reduce a $TD(0)$. De esta forma, la ecuación 2.20 ofrece una gran variedad de métodos de predicción o evaluación, teniendo a *TD(0)* y a Monte Carlo en sus extremos.

Ahora bien, el algoritmo $TD(\lambda)$ no puede implementarse debido a su no causalidad, es decir, en cada paso de tiempo utiliza información de lo que va a suceder en pasos de tiempo futuros. Es aquí donde aparece el segundo enfoque, el enfoque hacia atrás (*backward view*). En el enfoque hacia atrás existe una variable adicional de memoria asociada a cada estado denominada traza de elegibilidad. Las trazas de elegibilidad son variables encargadas de almacenar información sobre cómo de reciente fue visitado cada estado. En cada paso de tiempo estas trazas de elegibilidad se actualizan según la ecuación 2.21.

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{si } s \neq s_t \\ \gamma\lambda e_{t-1}(s,a) + 1 & \text{si } s = s_t \end{cases} \tag{2.21}$$

En el caso de las *trazas reemplazantes* en vez de incrementar el valor de la traza en 1, se sustituye su valor por 1 tal como muestra la ecuación 2.22.

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{si } s \neq s_t \\ 1 & \text{si } s = s_t \end{cases} \tag{2.22}$$

donde $e_t \in \Re$ representa la traza de elegibilidad del estado s en el tiempo t, γ es el factor de descuento y λ es la misma variable introducida en la ecuación 2.20 denominado ahora parámetro de decaimiento de la traza (*trace-decay parameter*). Si un estado no es visitado en el paso del tiempo t su traza se disminuye exponencialmente según $\gamma\lambda$, mientras que si es visitado su traza se incrementa en 1 (trazas acumulativas) o se establece a 1 (trazas reemplazantes) (figura 2.2).

La evaluación de la política según $TD(\lambda)$ produce, por tanto, actualizaciones en los valores de la función valor correspondiente a cada estado dependiendo de cómo de reciente han sido visitados de acuerdo a sus trazas de elegibilidad. Esta actualización se lleva a cabo como indica la ecuación 2.23

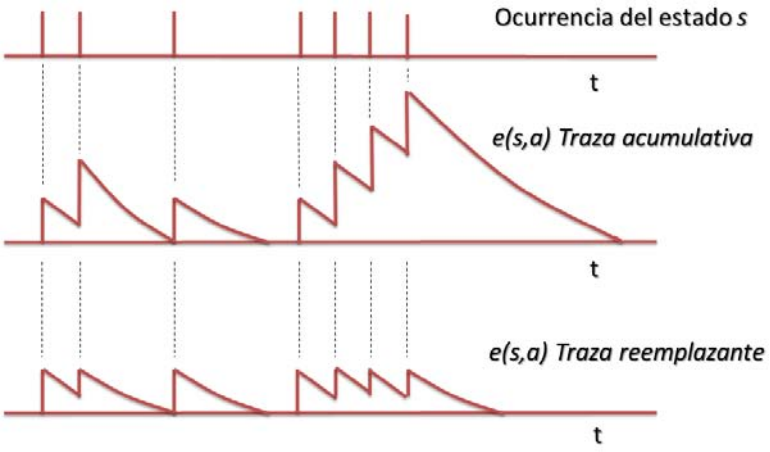


Figura 2.2: Ejemplo de la evolución de las trazas de elegibilidad.

$$V_t(s) = V_{t-1}(s) + \alpha\delta_t e_t(s), \forall s \in S \tag{2.23}$$

donde δ_t es el *error TD*.

$$\delta = r + \gamma V(s') - V(s) \tag{2.24}$$

2.2.6. Aprendizaje por Refuerzo Multi-Agente

Un sistema multi-agente [Weiss, 1999] puede ser definido como un grupo de entidades autónomas que interactúan compartiendo un entorno común, el cual perciben mediante sensores y en el que actúan mediante actuadores [Vlassis, 2003]. Estas entidades autónomas que interactúan entre sí son conocidas como agentes, y pueden ser programados con comportamientos establecidos *a priori*, aunque suele ser bueno que aprendan nuevos comportamientos *on-line* a medida que el rendimiento del agente o del sistema multi-agente en conjunto evoluciona gradualmente [Sen and Weiß, 1999]. Esto es así porque el entorno puede evolucionar conforme avanza el tiempo, lo que hace muy difícil diseñar buenos comportamientos *a priori*. Por este motivo y por la buena convergencia y consistencia de los algoritmos, la utilización de técnicas de aprendizaje por refuerzo en entornos con un único agente resulta muy atractiva para el aprendizaje en entornos multi-agentes también.

Existen diferentes criterios a la hora de clasificar los sistemas multi-agentes que utilizan aprendizaje por refuerzo (*Multi-Agent Reinforcement Learning, MARL*). Por un lado existe la clasificación que se establece en función del objetivo del aprendizaje de cada agente (estabilidad, adaptación o la combinación de ambas), y por otro lado existe la clasificación que se establece utilizando el tipo de tarea (totalmente cooperativa, totalmente competitiva, o la combinación de ambas) [Buşoniu *et al.*, 2008]. A continuación se describe cada una de ellas.

2.2.6.1. Clasificación en Función del Objetivo de Aprendizaje

En este caso, existen dos tipos de objetivos fundamentales. Por un lado la estabilidad de la dinámica de aprendizaje del agente, y por otro la adaptación a los comportamientos del resto de agentes en el sistema. La estabilidad se refiere a la convergencia a algún tipo de

política estacionaria, mientras que la adaptación se asegura que el rendimiento se mantiene o se mejora a medida que el resto de agentes cambia sus políticas de comportamiento. La convergencia a algún tipo de equilibrio es un requisito básico de estabilidad [Hu and Wellman, 2003]. Esto significa que los comportamientos de los agentes tienden a converger a algún tipo de equilibrio coordinado. En este caso, el equilibrio de Nash [1] se ha utilizado frecuentemente. Aunque en algunos trabajos se arguye que la convergencia al equilibrio de Nash en dominios estocásticos no está clara [Shoham *et al.*, 2003].

Otros trabajos defienden que la convergencia es un requisito para la estabilidad, y añaden el concepto de racionalidad como un criterio de adaptación. La racionalidad está definida como el requisito de que un agente converja a una mejor política a medida que el resto de agentes permanecen con una política estacionaria [Bowling and Veloso, 2002]. En este caso, a pesar que la convergencia a un equilibrio de Nash no está explícitamente requerida, ésta aparece de forma natural si todos los agentes son racionales. Otros objetivos alternativos al de racionalidad, como el de optimalidad, compatibilidad, seguridad, que son requisitos de adaptación, se discuten en otros trabajos [Powers and Shoham, 2005].

2.2.6.2. Clasificación en Base al Tipo de Tarea

Se distinguen tres tipos de tarea: tareas totalmente cooperativas, tareas totalmente competitivas y tareas mixtas que presentan la combinación de las anteriores [Buşoniu *et al.*, 2008]. En las tareas totalmente cooperativas todos los agentes tienen la misma función de refuerzo ($R_1(s,a) = R_2(s,a) = \ldots = R_n(s,a)$) y el objetivo de aprendizaje es maximizar el refuerzo común. Ejemplos de algoritmos para este tipo de tarea son *Team Q-Learning* [Littman, 2001] y *Distributed Q-Learning* [Lauer and Riedmiller, 2000]. En cambio, en las tareas totalmente competitivas (para dos agentes, cuando $R_1(s,a) = -R_2(s,a)$) se puede aplicar el principio minimax: tratar de obtener los máximos beneficios asumiendo que el adversario va a tratar de seguir la estrategia que más nos perjudica, es decir, una estrategia que minimiza nuestros beneficios [Littman, 2001]. En el último caso, en el caso de las tareas mixtas, no se imponen restricciones en las funciones de refuerzo de los agentes. De hecho, es común utilizar directamente los algoritmos propios del aprendizaje por refuerzo dedicados al aprendizaje de un único agente debido a su simplicidad [Mataric, 1997; Fernández *et al.*, 2005].

2.3. Generalización en aprendizaje por refuerzo

En las secciones anteriores donde se han descrito las técnicas de aprendizaje por refuerzo se ha asumido un conjunto finito de estados y de acciones. Lo cierto es que esto último se trata de la situación ideal siendo en la mayoría de las ocasiones falso. En muchos dominios los estados y acciones se representan como un conjunto de atributos de valores continuos haciendo que estos conjuntos sean intratables computacionalmente. Ante esta situación se deberá buscar una representación alternativa tanto del espacio de estados como del de acciones. En este sentido existen dos aproximaciones básicas:

- **Aproximación de funciones**. Se denomina de esta forma porque dado un conjunto de ejemplos de la función objetivo que se desea aprender (e.g. la función de valor-

[1] El **equilibrio de Nash** en juegos de dos o más jugadores es la solución en la que todos los participantes maximizan su beneficio, y donde cada jugador individual no gana nada modificando su estrategia mientras los otros mantengan las suyas [Rasmusen, 2007].

estado), intenta generalizar esos ejemplos para construir una aproximación completa de dicha función objetivo. La aproximación de funciones, como se detallará más adelante, puede ser lineal y no lineal. El principal inconveniente de las funciones de aproximación lineal frente a las no lineales es la correcta selección de las características, que normalmente se modela *ad-hoc*, lo cual requiere, en la mayoría de los casos, de un fuerte conocimiento del dominio. Por este y por otros motivos que se analizarán más adelante, los métodos que se encuentran con más frecuencia en la literatura son los métodos no lineales de descenso de gradiente, en los que se suele emplear redes de neuronas como aproximadores [Lin, 1991; Tesauro, 1992; Haykin, 1994; Bishop, 1996; Bishop, 2006], y en los últimos años máquinas de vectores de soporte (*support vector machines*) [Goto *et al.*, 2002; Hu *et al.*, 2006; B. Bethke and J. How and A. Ozdaglar, 2008; Xu *et al.*, 2007; Xu *et al.*, 2011].

- **Discretización del espacio**. Se basan en la obtención de conjuntos reducidos de los espacios de estados y acciones, que permitan agrupar en regiones distintos grupos de estados y acciones. Estos métodos normalmente tratan de forma lineal la aproximación de la función de valor-acción [Munos and Moore, 2002; Fernández and Borrajo, 1999; García *et al.*, 2007].

Estas dos técnicas son ampliamente descritas en las siguientes secciones.

2.3.1. Aproximación de funciones

Los métodos de diferencia temporal descritos en la sección 2.2.3.2 se basan en estimar la función de valor-estado, V, o la función de valor-acción, Q. La aproximación que se tiene en un instante de tiempo t de la función valor-estado, $\hat{V}_t$, o de la función valor-acción, $\hat{Q}a_t$, se puede representar como una función de un conjunto de parámetros representados por el vector θ_t en vez de representarla. Esto significa que el valor de la función valor-estado $\hat{V}_t$ o de la función valor-acción Qa_t depende únicamente del vector de parámetros θ_t, variando en cada instante de tiempo en función de cómo varían los valores contenidos en θ_t. La idea es que la dimensión del vector θ_t es mucho menor que el número de estados, de forma que variando unos pocos parámetros del vector se modificará el valor estimado de la función de valor de muchos estados. Por tanto, este problema puede entenderse como el clásico problema de aproximación de funciones en aprendizaje supervisado y por tanto se podrán emplear todas las técnicas que existen para este efecto como redes de neuronas, árboles de decisión, etc. En el caso de que se quiera aproximar la función *Q(s, a)* en vez de la función *V(s)*, se tiene el inconveniente de que se introduce un parámetro de entrada más. En este caso, si se dispone de un conjunto de acciones reducido, $D_a \subseteq A$, se pueden emplear tantos aproximadores como acciones, es decir $\hat{Q}a_i, i = 1, 2, ..., |D_a|$.

En el proceso de estimación de funciones es necesario decidir cuál es la medida que nos permite determinar si la aproximación obtenida es buena o mala. La mayoría de los métodos supervisados utilizan para este efecto el error cuadrático medio. En aprendizaje por refuerzo, la función que se quiere aproximar es la función $V^\pi(s)$ en caso de que se quiera aproximar la función de valor-estado. En este caso, el error cuadrático medio se define como la diferencia entre el valor real, $V^\pi(s)$, y el valor estimado, $\hat{V}_t(s)$ en el instante de tiempo t tal y como se muestra en la ecuación 2.25.

$$ECM(\theta_t) = \sum_{s \in S} P(s)[V^\pi(s) - \hat{V}_t(s)]^2 \tag{2.25}$$

Se obtendría una ecuación similar para calcular el error cuadrático medio cuando se trata de aproximar la función de valor-acción, tal y como se muestra en la ecuación 2.26.

$$ECM(\theta_t) = \sum_{s \in S, a \in A} P(s)[Q_a^\pi(s) - \hat{Q}_a(s)]^2 \tag{2.26}$$

En ambas ecuaciones, la aportación de cada estado se pondera con la probabilidad de que se dé ese estado, $P(s)$. El objetivo, por tanto, es encontrar el conjunto de parámetros θ_t que minimice el error cuadrático medio. El objetivo de los métodos de descenso de gradiente es buscar el punto mínimo de la función $ECM(\theta_t)$ realizando pequeñas modificaciones en el sentido que reduzca más el error para cada ejemplo de entrada, como se muestra el la ecuación 2.27.

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha[V^\pi(s_t) - \hat{V}_t(s_t)]\nabla_{\vec{\theta}_t}\hat{V}_t(s_t) \tag{2.27}$$

donde $\nabla_{\vec{\theta}_t}\hat{V}_t(s_t)$ es el gradiente de $\hat{V}_t(s_t)$ con respecto a $\vec{\theta}_t$.

Para la función valor-acción las modificaciones en el vector de parámetros $\vec{\theta}$ se realizan siguiendo la dirección negativa del gradiente $\nabla_{\vec{\theta}_t}\hat{Q}_a(s_t)$.

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha[Q_a^\pi(s_t) - \hat{Q}_a(s_t)]\nabla_{\vec{\theta}_t}\hat{Q}_a(s_t) \tag{2.28}$$

En este caso, las modificaciones que se realizan sobre el vector de parámetros son proporcionales al gradiente negativo del error cuadrático para ese ejemplo, s_t, que es el camino más rápido para encontrar el mínimo de la función y por tanto minimizar el error. La ecuación 2.27 presenta sin embargo varias limitaciones. El primer gran inconveniente es que se presupone conocimiento a priori de $V^\pi(s_t)$, es decir, se presupone conocido el valor de la función de valor que se está aproximando para el estado s_t. En los problemas de aprendizaje supervisado esto se cumple siempre, mientras que dicho valor es desconocido en la mayoría de los problemas de aprendizaje por refuerzo. No obstante, se podría substituir este valor por un aproximador, v_t, quedando la ecuación 2.27 como se muestra en la ecuación 2.29.

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha[v_t - V_t(s_t)]\nabla_{\vec{\theta}_t}V_t(s_t) \tag{2.29}$$

En el caso de los métodos de Monte Carlo, el objetivo $v_t = R_t$, donde el valor R_t queda expresado en la ecuación 2.15. Para el caso de los métodos $TD(\lambda)$, se emplea $v_t = R_t^\lambda$ como aproximación de $V^\pi(s_t)$ donde R_t^λ queda definido en la ecuación 2.19. Los métodos $TD(\lambda)$ pueden ser vistos desde dos enfoques diferentes. Desde el enfoque denominado *forward view*, las actualizaciones de los pesos se realizarían como muestra la ecuación 2.30.

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha[R_t^\lambda - V_t(s_t)]\nabla_{\vec{\theta}_t}V_t(s_t) \tag{2.30}$$

Desafortunadamente, el algoritmo $TD(\lambda)$ presenta dos inconvenientes. Por un lado, no puede implementarse debido a su no causalidad, es decir, en cada paso de tiempo utiliza información de lo que va a suceder en pasos de tiempo futuros. Aquí aparece el segundo enfoque, *backward view*, en el cual las actualizaciones de los pesos se realizan como muestra la ecuación 2.31.

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha\delta_t\vec{e}_t \tag{2.31}$$

donde δ_t es el error TD como se muestra en la ecuación 2.32 para la función de valor-estado y $\vec{e}$ es el vector de elegibilidad.

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{2.32}$$

En cambio, si se trata de aproximar la función de valor-acción, la fórmula que se empleará en la actualización será 2.33 en el caso de que se utilice *Sarsa* y 2.34 en caso de que se utilice *Q-learning* como algoritmo de aprendizaje por refuerzo.

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{2.33}$$

$$\delta_t = r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t) \tag{2.34}$$

El segundo inconveniente es cómo definir el vector de parámetros θ_t. A continuación se muestran dos aproximaciones en función de si se utilizan funciones de aproximación lineales o no lineales.

Funciones de aproximación lineal

Uno de los ejemplos más importantes de aproximación de funciones siguiendo la dirección negativa del error empleado para aproximar la función V_t es una función lineal con los parámetros definidos en el vector $\vec{\theta_t}$. Para solucionar el problema de la representación del espacio de estados se puede emplear una representación alternativa transformando estos estados en características. De esta forma, a cada estado s, le corresponde un vector de características $\phi_s = (\phi_s(1), \phi_s(2), ..., \phi_s(n))$ con el mismo número de componentes que θ_t. Estas características son empleadas para aproximar la función estado-valor como se muestra en la ecuación 2.35.

$$V_t(s) = \vec{\theta_t^T}\vec{\phi_s} = \sum_{i=1}^{n} \theta_t(i)\phi_s(i) \tag{2.35}$$

Uno de los métodos que se emplean siguiendo esta aproximación es la denominada *Codificación gruesa* (*Coarse coding*). Si suponemos un ejemplo de tarea en el que el conjunto de estados es continuo y bidimensional, un tipo de características podría definirse como la pertenencia o no a un círculo, de entre un conjunto de círculos que pueden estar solapados (figura 2.3).

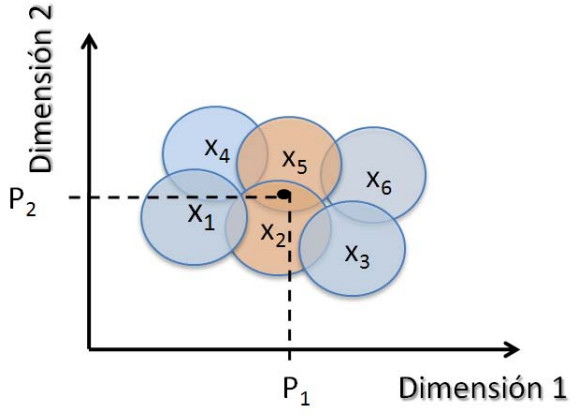


Figura 2.3: Ejemplo de *Codificación gruesa* para un espacio bidimensional.

Si el punto del espacio de entrada se encuentra dentro del círculo se puede decir que la característica asociada a ese círculo toma el valor de 1 mientras que si no, toma el valor de 0. Por lo tanto se trata de características binarias. De esta forma, a partir del vector de características se puede dar una localización aproximada de la posición donde se encuentra realmente ese punto en el espacio. El vector de características asociado a la figura 2.3 para el punto P de componentes (P_1, P_2) sería $\phi(P) = (0, 1, 0, 0, 1, 0)$. El método *Codificación gruesa* más ampliamente utilizado es la *Codificación en teja* (*Tile coding*), también conocido como *CMAC* [Albus, 1975]. En esta codificación el espacio de características se divide en particiones denominados *tejas* (*tiles*). La forma, el tamaño y la densidad de las tejas determina la capacidad de generalización (Figura 2.4).

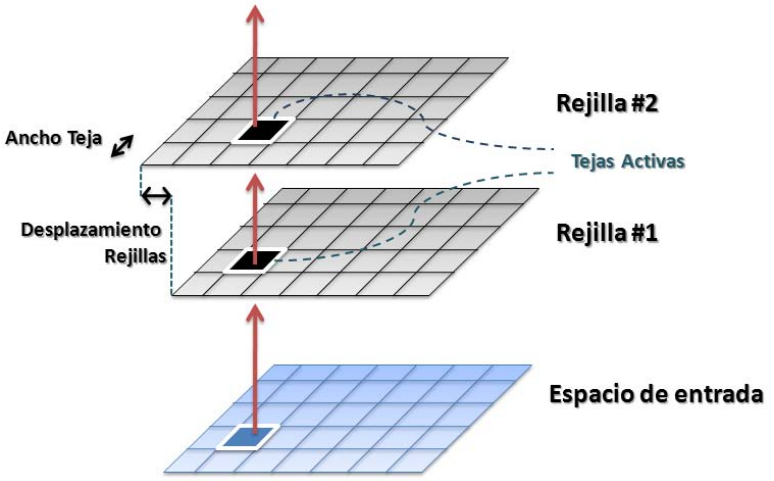


Figura 2.4: Ejemplo de *CMAC* para un espacio de entrada de dos dimensiones.

De estas ecuaciones se puede obtener fácilmente una versión libre del modelo sin más que aproximar, en vez de la función de valor V, la función $Q(s,a)$, o si suponemos un conjunto reducido de acciones, $D_a \subseteq A$, con $|D_a|$ acciones posibles $a_1, a_2, ..., a_{|D_a|}$, tendríamos $|D_a|$ aproximadores diferentes identificados por $\hat{Q}_{a_i}(s)$ para $i = 1, 2, ..., |D_a|$. En la tabla 2.3 se puede ver la combinación de *CMAC* con aprendizaje por refuerzo. En este algoritmo tenemos tantos aproximadores, $Q_a(s)$, como acciones definidas en el conjunto discreto de acciones, $D_a \subseteq A$.

En dicho algoritmo se utiliza un vector de parámetros $\vec{\theta}_a$ y un vector de características $\vec{\phi}_a$ diferente por cada acción a perteneciente al conjunto discreto de acciones $D_a \subseteq A$ que es capaz de ejecutar el agente. En el algoritmo, el vector de características asociado a cada acción se representa como $\vec{\phi}_a$. Al comienzo de un episodio, se inicializan las trazas de elegibilidad a 0, $\vec{e} \leftarrow 0$, y se selecciona el estado inicial, s_0. Después se iteran sobre todas las acciones disponibles, $a \in D_a$, en el estado actual s_0 en el que se encuentra el agente, de forma que para cada acción a, y para cada *rejilla* (*tiling*) de cada atributo que describe el estado, se calculan el conjunto de *tejas* activas, $\vec{\phi}_a$. Acto seguido el valor de la función de valor-acción es calculada como la suma de los pesos de las *tejas* activas almacenadas en $\vec{\phi}_a$. Entonces, se elige una acción a ejecutar por el agente del conjunto de acciones disponibles siguiendo una estrategia de exploración $\epsilon - greedy$. Además, en cada paso del episodio se disminuye el valor de la traza de elegibilidad, la traza para la acción seleccionada es puesta a 1, y las trazas para el resto de acciones disponibles se establece a 0. Este esquema de actualización

Algoritmo de aprendizaje por refuerzo y CMAC para acciones discretas $(K, H, \alpha, \gamma, D_a) \rightarrow Q_a(s)$

Dados:
  Un número máximo de episodios a ejecutar, K.
  Un número máximo de pasos por episodio, H.
  Los parámetros de aprendizaje, α y γ.
  Un conjunto discreto de acciones D_a.
1. Inicializar:
  Inicializar $Q_a(s)$ y $\vec{\theta}_a$ arbitrariamente, y $\vec{e}_a \leftarrow 0$.
2. Ejecutar:
  Para $k = 1$ a K **hacer** ;; Para cada episodio
    $\vec{e} \leftarrow 0$
    Seleccionar el estado inicial del episodio, s_0.
    Para cada $a \in D_a$ **hacer**
      $\vec{\phi}_a \leftarrow$ Conjunto de características presentes en s_0,a.
      $Q_a(s_0) \leftarrow \sum_{i \in \vec{\phi}_a} \theta_a(i)$
    $a \leftarrow \arg\max_a Q_a$
    Con probabilidad $1 - \epsilon$: $a \leftarrow$ Acción aleatoria $a \in D_a$
    Para $h = 1$ a H **hacer** ;; Para cada paso del episodio
      $\vec{e} \leftarrow \gamma\lambda\vec{e}$;; Se disminuyen las trazas de todas las acciones
      Para cada acción $a' \neq a$: ;; Opcional
        Para todo $i \in \vec{\phi}_a$: $e_{a'}(i) \leftarrow 0$
      Para todo $i \in \vec{\phi}_a$: $e_a(i) \leftarrow 1$
      Ejecutar la acción a, obtener la recompensa r y el próximo estado s_1.
      $\delta \leftarrow r - Q_a(s_0)$
      Para cada $a \in A$ **hacer**
        $\vec{\phi}_a \leftarrow$ Conjunto de características presentes en s_1,a.
        $Q_a(s_1) \leftarrow \sum_{i \in \vec{\phi}_a} \theta(i)$
      $a' \leftarrow \arg\max_a Q_a(s_1)$
      Con probabilidad $1 - \epsilon$: $a' \leftarrow$ Acción aleatoria $a' \in D_a$.
      $\delta \leftarrow \delta + \gamma Q_{a'}(s_1)$
      $\vec{\theta}_a \leftarrow \vec{\theta}_a + \alpha\delta\vec{e}_a$
      $a \leftarrow a'$
3. Devolver $Q_a(s)$

Tabla 2.3: Algoritmo de aprendizaje utilizando CMAC para un conjunto discreto de acciones.

de las trazas de elegibilidad es conocido como trazas reemplazantes. Actualizadas las trazas de elegibilidad, se ejecuta la acción que hará transitar al agente a un nuevo estado s_1 que le llegará al agente junto con una señal de refuerzo r. Después, se empieza a calcular el valor del error en la estimación de la función valor-acción realizando la diferencia entre r, el refuerzo recibido, y Q_a. Luego, se encuentran el conjunto de *tejas* activas, $\vec{\phi}_a$. Los pesos asociados a estas *tejas* activas serán utilizados para calcular la función de valor-acción para cada acción en el estado actual s_1. Entonces, se selecciona la próxima acción a ser ejecutada por el agente siguiendo, como en el caso anterior, una estrategia de exploración $\epsilon - greedy$. Por último, se finaliza el cálculo del error en la estimación de la función valor-acción que se inició anteriormente, y se ajustan los pesos contenidos en el vector de parámetros $\vec{\theta}$.

Hasta ahora se ha asumido un conjunto discreto de acciones, aunque esto muy raras veces ocurre en la realidad. Hay veces que el conjunto de acciones también es demasiado grande, e incluso continuo, produciéndose la misma problemática que con los estados. Con conjuntos de acciones discretos, como en el caso anterior, puede plantearse un aproximador (e.g. una red neuronal) por cada acción para aproximar el valor de Q. En cambio, con

espacios de acciones continuos puede plantearse un aproximador, $Q(s, a)$, que reciba como entradas el estado y la acción. La tabla 2.4 muestra la utilización de CMAC cuando el espacio de acciones es continuo [Santamaría *et al.*, 1998].

Algoritmo de aprendizaje por refuerzo y CMAC para acciones continuas $(K, H, \gamma, \Delta_a) \rightarrow Q(s, a)$	
00	**Dados:**
01	Un número máximo de episodios a ejecutar, K.
02	Un número máximo de pasos por episodio, H.
03	Los parámetros de aprendizaje, α y γ.
04	El incremento para las acciones, Δ_a.
05	**1. Inicializar:**
06	Inicializar Q(s,a) y $\vec{\theta}$ arbitrariamente, y $\vec{e} \leftarrow 0$.
07	**2. Ejecutar:**
08	**Para** $k = 1$ a K **hacer** ;; Para cada episodio
09	$\vec{e} \leftarrow 0$
10	Seleccionar el estado inicial del episodio, s_0.
11	$a \leftarrow one\_step\_search(Q, s_0, \Delta_a)$
12	Con probabilidad $1 - \epsilon$: $a \leftarrow$ Acción aleatoria
13	$\vec{\phi} \leftarrow$ Conjunto de características presentes en s_0, a.
14	$Q(s_0, a) \leftarrow \sum_{i \in \vec{\phi}} \theta(i)$
15	**Para** $h = 1$ a H **hacer** ;; Para cada paso del episodio
16	$\vec{e} \leftarrow \gamma\lambda\vec{e}$
17	Para todo $i \in \vec{\phi}$: $e(i) \leftarrow 1$
18	Ejecutar la acción a, obtener la recompensa r y el próximo estado s_1.
19	$\delta \leftarrow r - Q(s_0, a)$
20	$a' \leftarrow one\_step\_search(Q, s_1, \Delta_a)$
21	Con probabilidad $1 - \epsilon$: $a' \leftarrow$ Acción aleatoria
22	$\vec{\phi} \leftarrow$ Conjunto de características presentes en s_1,a'.
23	$Q(s_1, a') \leftarrow \sum_{i \in \vec{\phi}} \theta(i)$
24	$\delta \leftarrow \delta + \gamma Q(s_1, a')$
25	$\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
26	$a \leftarrow a'$
27	**3. Devolver** $Q(s, a)$

Tabla 2.4: Algoritmo de aprendizaje utilizando CMAC para acciones continuas.

La diferencia fundamental con respecto al algoritmo anterior es que la red CMAC consiste en varias rejillas solapadas discretizando tanto el espacio estados como el espacio de acciones, produciendo una representación de características para la totalidad del espacio. El inconveniente que presenta el algoritmo de la tabla 2.4, radica en encontrar dentro de la red la política de comportamiento óptima (i.e., no es un problema trivial el encontrar la acción que corresponda al valor máximo de la función de valor-acción para un estado dado). En este caso, Santamaría et al. [Santamaría *et al.*, 1998] utilizan un algoritmo llamado *One-Step Search* (tabla 2.5).

En este algoritmo, se busca la acción óptima para un estado s probando acciones de un conjunto discreto. La forma de obtener este conjunto discreto consiste en tomar el rango de la acción y probar desde el valor mínimo, a_{min}, al máximo, a_{max}, en intervalos dados por un incremento Δ_a. Al final, la acción con mayor valor de la función $\hat{Q}$ es devuelta, $\hat{a}^*$.

En los últimos años también se han empleado con éxito para la aproximación de funciones en aprendizaje por refuerzo las máquinas de vectores de soporte (*support vector machines*) y los métodos basados en funciones *Kernel* (*Kernel-based methods*) [Schölkopf and Smola, 2002; Smola and Schölkopf, 2004]. La idea básica que se encierra tras estas

one_step_search $(Q, s, \Delta_a) \rightarrow \hat{a}^*$	
00	**1. Inicializar:**
01	$\hat{a}^* = a_{min}$
02	$maxQ = Q(s, a_{min})$
03	**Para** $a = a_{min}$ a a_{max} **hacer**
04	**Si** $Q(s, a) > maxQ$ **hacer**
05	$\hat{a}^* = a$;; Se guarda la acción que produce el máximo valor
06	máx $Q = Q(s, a)$
07	$a \leftarrow a + \Delta_a$
08	**3. Devolver** $\hat{a}^*$

Tabla 2.5: Algoritmo de One-Step Search para la búsqueda de acciones óptimas.

técnicas se describe brevemente a continuación. Supongamos que tenemos un conjunto de entrenamiento formado por $(x_1, y_1), \ldots, (x_l, y_l) \in \mathcal{X} \times \Re$, donde $\mathcal{X}$ denota el conjunto de patrones de entrada. El objetivo es encontrar una función $f(x)$ que tiene como mucho una desviación de ϵ con respecto a los y_i del conjunto de entrenamiento y a su vez que sea lo más plana posible [Smola and Schölkopf, 2004]. La función $f(x)$ que se trata de aproximar es lineal y se muestra en la ecuación 2.36.

$$f(x) = \theta^T x = \sum_{k=1}^{n} \theta(i)x(k) \tag{2.36}$$

La función que se muestra en la ecuación 2.36 tiene la misma forma que la función para aproximar la función de valor que se introducía al principio del apartado en la ecuación 2.35. En esta técnica en cambio, el problema de entrenamiento se plantea como un problema de optimización cuadrática. "Plana" en el contexto de la ecuación 2.36 significa que se buscan valores pequeños para el vector θ. Esto se traduce a tratar de minimizar la norma del vector θ, es decir, $||\theta||^2 = \langle \theta, \theta \rangle$. Entonces, este problema puede ser escrito como se muestra a continuación.

Minimizar:

$$\tfrac{1}{2}||\theta||^2$$

sujeto a:

$$\begin{aligned} y_i - \theta^T x_i &\leq \epsilon \\ -y_i + \theta^T x_i &\leq \epsilon \end{aligned}$$

Las restricciones aseguran que la función $f(x)$ aproxima todos los pares (x_i, y_i) con una precisión ϵ. Este problema puede ser resuelto más fácilmente utilizando multiplicadores de Lagrange [Fletcher, 1987], que poseen además la llave para extender los métodos de máquinas de vectores de soporte a funciones no lineales mediante la utilización de funciones *Kernel*. La ecuación 2.36 puede ser reescrita como se muestra en la ecuación 2.37 considerando que $\theta = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)x_i$.

$$f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\langle x_i, x \rangle \tag{2.37}$$

Hasta ahora se han supuesto los métodos de máquinas de vectores de soporte como métodos de aproximación lineal. Para aproximar funciones no lineales utilizando esta técnica se suele utilizar el truco del *Kernel* (*kernel trick*) [Smola and Schölkopf, 2004]. Este truco consiste en mapear los datos de entrenamiento a un espacio con un número mayor de dimensiones donde los datos sí que son linealmente separables (Figura 2.5). De esta forma, el producto escalar $\langle x_i, x \rangle$ en la ecuación 2.37 se sustituye por una función *Kernel* como queda reflejado en la ecuación 2.38.

$$f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) K(x_i, x) \tag{2.38}$$

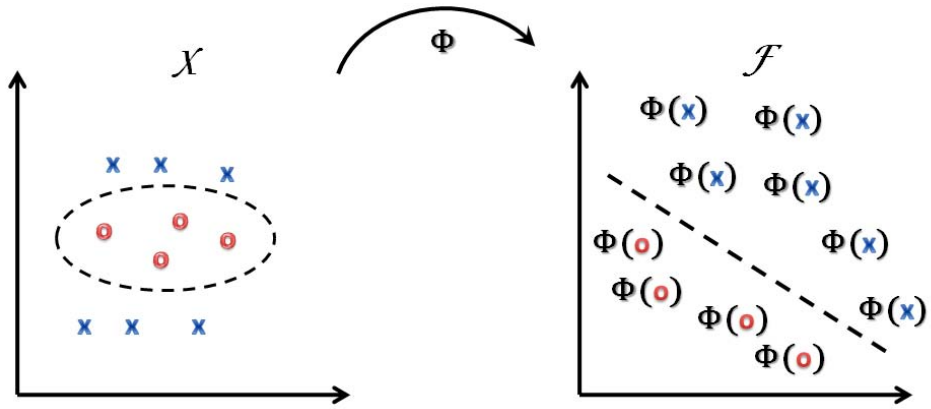


Figura 2.5: Ejemplo de mapeo de los datos originales de entrenamiento no separables linealmente en el espacio $\mathcal{X}$ a un espacio $\mathcal{F}$ donde se pueden separar linealmente.

donde $K(x_i, x) = \langle \phi(x_i), \phi(x) \rangle$, y ϕ es una función de mapeo $\mathcal{X} \rightarrow \mathcal{F}$ donde $\mathcal{F}$ tiene mayor dimensión que $\mathcal{X}$ (por ejemplo $\phi : \Re^2 \rightarrow \Re^3$). La función 2.38 puede ser utilizada para aproximar la función de valor-estado en aprendizaje por refuerzo como se indica en la ecuación 2.39 [Goto *et al.*, 2002; Xu *et al.*, 2005; Hu *et al.*, 2006; B. Bethke and J. How and A. Ozdaglar, 2008].

$$V(s) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) K(s_i, s) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) \langle \phi(s_i), \phi(s) \rangle \tag{2.39}$$

Las técnicas de máquinas de vectores de soporte combinadas con las funciones *Kernel* para aproximar funciones no lineales, poseen una serie de ventajas sobre las técnicas basadas en redes de neuronas y funciones de base radial. Por un lado se evita el problema de tener que seleccionar la topología adecuada de la red de neuronas y las funciones de activación (parámetros de los que depende en gran medida el rendimiento final del algoritmo). Por otro lado, permiten trabajar con espacios de estados muy grandes sin complicar el modelo ni los tiempos de computación. En cualquier caso, su utilización plantea el problema de seleccionar la función *Kernel* adecuada que puede ser diferente de un dominio a otro, y el problema de extraer conocimiento del aproximador sobre cuál es la mejor acción a ejecutar en cada momento [B. Bethke and J. How and A. Ozdaglar, 2008].

Funciones de aproximación no lineal

Debido a la dificultad de encontrar buenas características para las funciones de aproximación lineal, las funciones de aproximaciones lineales suelen ser las que con más frecuencia se utilizan para abordar el problema de generalización en aprendizaje por refuerzo. No obstante, este tipo de aproximaciones tiene el gran inconveniente de que en general pueden quedar estancados en mínimos locales, aunque suelen encontrar mejores soluciones que las aproximaciones lineales. Cuando se utilizan funciones de aproximación no lineal, la función de valor queda definida como se muestra en la ecuación 2.40.

$$V_t(s) = f(\vec{\theta}_t, \vec{\phi}_s) \tag{2.40}$$

Donde $\vec{\theta}_t$ es el vector de parámetros que no tiene necesariamente la misma dimensión que el vector de características $\vec{\phi}_s$. En la mayoría de los casos, f es una red de neuronas y el vector de parámetros $\vec{\theta}_t$ es un vector con todos los pesos de la red de neuronas en el instante t. El vector de parámetros se va adaptando durante el proceso de aprendizaje utilizando la técnica de descenso de gradiente, que en el contexto de las redes de neuronas suele llamarse *backpropagation* [Bryson and Ho, 1969; Werbos, 1974; Rumelhart *et al.*, 1988].

En este caso, como en el anterior, con conjuntos de acciones discretos puede plantearse un aproximador (e.g. una red neuronal) por cada acción para aproximar el valor de Q. En cambio, con espacios de acciones continuos puede plantearse un aproximador de la función Q que reciba como entradas el estado y la acción. El inconveniente en este caso, es encontrar dentro de la red (en el caso de que el aproximador sea una red de neuronas) la política de comportamiento óptima (i.e., no es un problema trivial el encontrar la acción que corresponda al valor máximo de la función de valor-acción para un estado dado). En estos casos, se suelen utilizar dos aproximadores diferentes, lo que se conoce como **métodos actor-crítico**. Los métodos de actor-crítico son métodos TD que tienen una estructura separada independiente para representar la función de valor [van Hasselt and Wiering, 2007] (o la función de valor-acción [Baird and Klopf, 1993; Prokhorov and Wunsch, 1997]) y otra para representar la política. Para este último aproximador también es común utilizar una red de neuronas [van Hasselt and Wiering, 2007].

La figura 2.6 representa la arquitectura de esta aproximación. En esta arquitectura el crítico normalmente almacena la función de valor, mientras que el actor selecciona las acciones que se llevan a cabo en cada momento. Los métodos actor-crítico son una extensión natural del aprendizaje por refuerzo aplicado a espacio de estados continuo, que permite trabajar también con espacio de acciones de naturaleza continua.

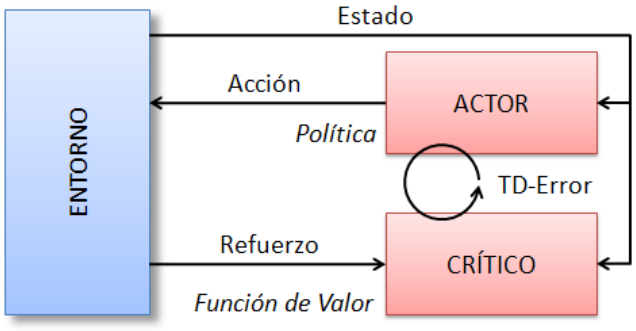


Figura 2.6: Arquitectura Actor-Crítico.

El funcionamiento de esta arquitectura se detalla en la tabla 2.6. Después de cada selección de acción del actor, el crítico evalúa el nuevo estado y determina si las cosas se están llevando a cabo mejor o peor de lo esperado. Esta evaluación se lleva a cabo utilizando el TD-Error, $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, dónde V es la función de valor actual implementada por el crítico. Si el TD-Error es positivo, significa que la tendencia a seleccionar la acción a_t debería ser reforzada en el futuro, mientras que si el TD-Error es negativo, significa que la tendencia a seleccionar la acción a_t debería ser penalizada.

Algoritmo Actor-Crítico $(K, H, \alpha, \gamma, \beta) \rightarrow (V(s), A(s,a))$	
00	**Dados:**
01	Un número máximo de episodios a ejecutar, K.
02	Un número máximo de pasos por episodio, H.
03	El ratio de aprendizaje para el crítico, α.
04	El factor de descuento, γ.
05	El ratio de aprendizaje para el actor, β.
06	**1. Inicializar:**
07	Inicializar el aproximador del actor $A_1(s,a)\ \forall s, a$
08	Inicializar el aproximador del crítico $V_1(s)\ \forall s$
09	**2. Ejecutar:**
10	**Para** $k = 1$ a K **hacer** ;; Para cada episodio
11	Seleccionar s_1
12	**Para** $h = 1$ a H **hacer** ;; Para cada paso del episodio
13	Seleccionar acción a_h derivada de $A(s_h, a)$ con exploración
14	Ejecutar a_h y observar r_h, s_{h+1}
15	**Si** $s_{h+1} = terminal$ **hacer**
16	$A_{h+1}(s_h, a_h) \leftarrow A_{h+1}(s_h, a_h) + \beta_h(r_h - V_h(s_h))$
17	$V_{h+1}(s_h) \leftarrow V_{h+1}(s_h) + \alpha_h(r_h - V_h(s_h))$
18	Seleccionar un nuevo s_{h+1} como estado de inicio del próximo episodio
19	**sino**
20	Calcular el TD-Error definido como $\delta_h = r_h + \gamma V_h(s_{h+1}) - V_h(s_h)$
21	$A_{h+1}(s_h, a_h) \leftarrow A_{h+1}(s_h, a_h) + \beta_h \delta_h$
22	$V_{h+1}(s_h) \leftarrow V_{h+1}(s_h) + \alpha_h \delta_h$
23	**3. Devolver** $V(s)$, $A(s,a)$

Tabla 2.6: Algoritmo de aprendizaje basado en métodos actor-crítico.

El algoritmo *Continuous Actor Critic Learning Automaton* (CACLA) puede ser visto como la versión continua del algoritmo anterior. La diferencia radica en que en lugar de almacenar valores estado-acción, el actor en CACLA almacena un único valor por cada estado: una aproximación de la acción óptima. En problemas con espacio de acciones multi-dimensionales, este valor podría ser un vector en lugar de un único valor. El algoritmo CACLA se describe en la tabla 2.7.

En CACLA, la acción a_h se obtiene realizando exploración gaussiana alrededor de la acción generada por el actor en el estado s_h, $A(s_h)$. Por tanto, la probabilidad de seleccionar la acción a_h siendo la acción $A(s_h)$ la sugerida por el actor queda definida mediante la ecuación 6.4.

$$P(s_h, a_h) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(a_h - A(s_h))^2/2\sigma^2}. \tag{2.41}$$

A diferencia del algoritmo anterior, el algoritmo CACLA sólo se actualiza cuando $\delta_h > 0$, puesto que una actualización cuando el TD-Error es negativo no garantiza que se vaya a actualizar a una acción que es mejor que la que actualmente está generando el actor. Por

Algoritmo Continuous Actor Critic Learning Automaton (CACLA) $(K, H, \alpha, \gamma, \beta) \rightarrow (V(s), A(s))$

Dados:
  Un número máximo de episodios a ejecutar, K.
  Un número máximo de pasos por episodio, H.
  El ratio de aprendizaje para el critic, α.
  El factor de descuento, γ.
  El ratio de aprendizaje para el actor, β.
1. Inicializar:
  Inicializar el aproximador del actor $A_1(s)$ $\forall s$
  Inicializar el aproximador del critic $V_1(s)$ $\forall s$
2. Ejecutar:
  Para $k = 1$ a K **hacer** ;; Para cada episodio
    Seleccionar s_1
    Para $h = 1$ a H **hacer** ;; Para cada paso del episodio
      Seleccionar acción a_h con exploración alrededor de $A(s_h)$ (i.e., exploración gaussiana)
      Ejecutar a_h y observar r_h, s_{h+1}
      Si $s_{h+1} = terminal$ **hacer**
        $V_{h+1}(s_h) \leftarrow V_{h+1}(s_h) + \alpha_h(r - V_{h+1}(s_h))$
        Seleccionar un nuevo s_{h+1} como estado de inicio del próximo episodio
      sino
        Calcular el TD-Error definido como $\delta_h = r_h + \gamma V_h(s_{h+1}) - V_{h+1}(s_h)$
        $V_{h+1}(s_h) \leftarrow V_{h+1}(s_h) + \alpha_h \delta_h$
      Si $\delta_h > 0$ **hacer**
        $A_{h+1}(s_h) \leftarrow \beta_h a_h$
3. Devolver $V(s)$, $A(s)$

Tabla 2.7: Algoritmo de aprendizaje CACLA basado en métodos Actor-Crítico.

tanto, CACLA sólo utiliza el símbolo del TD-Error para realizar actualizaciones en lugar de su valor concreto.

También existen trabajos en los cuales se trata de aproximar la función de valor-acción en lugar de la función de valor [Baird and Klopf, 1993; Prokhorov and Wunsch, 1997]. No obstante, el algoritmo CACLA se ha presentado superior a ellos en diferentes dominios [van Hasselt and Wiering, 2007]. En cualquier caso, como detallan Sutton y Barto [Sutton and Barto, 1998], los algoritmos basados en esta arquitectura están recibiendo cada vez más atención debido a que presentan dos ventajas significativas:

- Estos métodos requieren en general de *poca* computación para seleccionar las acciones que serán ejecutadas en cada momento. Si imaginamos un problema con un espacio de acciones continuo, es imposible utilizar un método que busque sobre este espacio infinito la acción óptima. En este método la política es explícitamente almacenada, por lo tanto la búsqueda intensiva de esta acción no es necesaria.

- Estos métodos pueden aprender una política explícitamente estocástica, es decir, pueden aprender las probabilidades óptimas de seleccionar varias acciones. Esta ventaja se muestra muy útil en entornos competitivos y no Markovianos [Singh and Jordan, 1994].

Además, mantener una arquitectura separada para representar la política (el actor) hace que sea en cierta medida más respetuosa con los modelos psicológicos y biológicos.

Aproximadores de funciones basados en memoria

Los aproximadores de funciones basados en memoria (*memory-based function approximators*) requieren almacenar en memoria pares *estado-acción* que el agente ha experimentado previamente. A estos pares *estado-acción* se añade habitualmente el valor Q formando así casos de la forma $C_i = (s_i, a_i, Q_i)$. La aproximación de la función de valor acción para un estado s se hace tomando un subconjunto reducido del conjunto de casos almacenados en memoria (normalmente se toman los k vecinos más cercanos de s, NN_s), y aplicando algún tipo de promedio ponderado de los valores Q_i que pertecen a los casos que conforman el vecindario NN_s.

Santamaría et al. [Santamaría *et al.*, 1998] presenta dos tipos de aproximadores basados en memoria que se distinguen en la complejidad de los casos que se almacenan. En el primero de estos aproximadores, cada caso C_i está compuesto por $C_i = (s_i, a_i, Q_i, e_i)$, donde s_i se corresponde con el estado, a_i se corresponde con la acción, Q_i se corresponde con el valor asociado al caso y e_i es la elegibilidad. Cuando llega un nuevo par (s_q, a_q) se calcula su distancia d_i a cada uno de los casos C_i que hay en memoria. Entonces, se calcula el conjunto de los vecinos más cercanos a (s_q, a_q), NN_q, utilizando la regla que se muestra en la ecuación 2.42.

$$NN_q = \{C_i \in Memoria \mid d_i \leq \tau_k\} \tag{2.42}$$

El parámetro τ_k es un parámetro de densidad (*density threshold*) que determina cuando un nuevo caso debe ser añadido a la memoria. Para calcular el valor $\hat{Q}(s_q, a_q)$ se utiliza una función kernel, $K()$, para calcular la contribución relativa de cada caso en NN_q al valor $\hat{Q}(s_q, a_q)$. Santamaría utiliza para este propósito una función gaussiana $K(d_i) = exp(-d_i^2/\tau_k^2)$. El valor final $\hat{Q}(s_q, a_q)$ será la media ponderada de cada de los valores Q_i de los casos en el vecindario NN_q como se muestra en la ecuación 2.43.

$$\hat{Q}(s_q, a_q) = \sum_{C_i \in NN_q} \frac{K(d_i)}{\sum_j K(d_j)} Q_i \tag{2.43}$$

El aprendizaje se realiza sobre cada caso C_i en memoria utilizando la ecuación 2.44.

$$\triangle Q_i = \alpha(r_{t+1} + \hat{Q}_{t+1} - \hat{Q}_t)e_i \; \forall C_i \in \; Memoria \tag{2.44}$$

Las trazas de elegibilidad de todos los casos almacenados en memoria también son actualizadas siguiendo la ecuación 2.45.

$$e_i \leftarrow \begin{cases} \frac{K(d_i)}{\sum_j K(d_j)} & \text{si } C_i \in NN_q, \\ \lambda\gamma e_i & \textit{en otro caso.} \end{cases} \tag{2.45}$$

Un nuevo caso se añade a memoria cuando la distancia al vecino más cercano supera el valor de cierto umbral, τ_k. En el segundo de estos aproximadores, Santamaría et al. [Santamaría *et al.*, 1998] proponen complicar la estructura de los casos añadiendo nuevos elementos. Ahora un caso almacenará, además, un conjunto de N_i acciones diferentes, a_{ij}, los valores Q asociados a cada una de estas acciones, Q_{ij}, y sus respectivas trazas de elegibilidad e_{ij} $(j = 1, \ldots, N_i)$ De esta forma, un caso quedará definido como $C_i = (s_i, Q_i, e_i, \{a_{ij}\}, \{Q_{ij}\}, \{e_{ij}\} | j = 1, \ldots, N_i)$, donde Q_i es el valor que generaliza el valor Q a través de todas las acciones almacenadas para el estado s_i. En esta nueva aproximación, los vecinos más cercanos asociados a un nuevo estado s_q se calculan teniendo

en cuenta la distancia, d_i^s, entre el estado s_q y cada estado s_i asociado a un caso C_i en memoria. Por lo tanto, en esta nueva versión únicamente se tiene en cuenta la similitud entre estados y no entre estados y acciones como en el caso anterior. Para determina el valor estimado $\hat{Q}(s_q, a_q)$ se utilizan dos funciones kernel: $K^s()$ y $K^a()$. La primera se utiliza para determinar la contribución relativa de cada caso C_i al valor de $\hat{Q}(s_q, a_q)$. La segunda se emplea para determinar la contribución relativa de cada acción, a_{ij}, dentro de un caso para el evaluar el valor de $Q(a_q)$ tal como se muestra en la ecuación 2.46.

$$Q(a_q) = (1-\rho)Q_i + \rho\left(\sum_{\forall a_{ij}\in C_i} \frac{K^a(d_{ij}^a)}{\sum_j K^a(d_j^a)} Q_{ij}\right) \ \forall C_i \in NN_q \tag{2.46}$$

En la ecuación 2.46, el parámetro ρ pondera la combinación que se realiza entre el valor Q_i asociado al estado s_i y los valores Q_{ij} asociados a cada acción en el estado s_i. Para calcular el valor de $\hat{Q}(s_q, a_q)$, se emplea la distancia d_i^s y la función de kernel $K^s()$ para determinar la contribución relativa de cada caso en NN_q, como queda reflejado en la ecuación 2.47.

$$\hat{Q}(s_q, a_q) = \sum_{C_i \in NN_q} \frac{K^s(d_i)}{\sum_j K^s(d_j)} Q_i(a_q) \tag{2.47}$$

Las actualizaciones en este caso, se llevan a cabo sobre cada caso en memoria y sobre cada acción dentro de cada caso como se muestra en las ecuaciones 2.48 y 2.49.

$$\triangle Q_i = \alpha(r_{t+1} + \hat{Q}_{t+1} - \hat{Q}_t)e_i \ \forall C_i \in \ Memoria \tag{2.48}$$

$$\triangle Q_{ij} = \alpha(r_{t+1} + \hat{Q}_{t+1} - \hat{Q}_t)e_{ij} \ \forall a_{ij} \in \ C_i \tag{2.49}$$

Las trazas de elegibilidad se actualizan siguiendo las ecuaciones 2.50 y 2.51.

$$e_i \leftarrow \begin{cases} (1-\rho)\frac{K^s(d_i^s)}{\sum_j K(d_j^s)} & \text{si } C_i \in NN_q, \\ \lambda\gamma e_i & \textit{en otro caso.} \end{cases} \tag{2.50}$$

$$e_{ij} \leftarrow \begin{cases} \rho\frac{K^a(d_i^a)}{\sum_j K(d_j^a)} & \text{si } C_i \in NN_q, \\ \lambda\gamma e_{ij} & \textit{en otro caso.} \end{cases} \tag{2.51}$$

La selección de la mejor acción a ejecutar en cada momento para estas dos aproximaciones propuestas por Santamaría et al. se realiza utilizando el algoritmo *One Step Search* mostrado en la tabla 2.5 y que evalúa el valor-acción de un número finito de acciones, y devuelve la acción con un valor más elevado. Sharma et al. [Sharma *et al.*, 2007] ofrecen una versión simplificada de esta última aproximación en la que los casos estarían formados por $C_i = (s_i, \{a_{ij}\}, \{Q_{ij}\}, \{e_{ij}\}|j = 1, \ldots, N_i)$, es decir, se han suprimido el componente Q_i que generaliza el valor de Q a través de todas las acciones y el valor de la traza de elegibilidad e_i.

Existen otras aproximaciones que se basan en este mismo principio. Smart y Kaelbling [Smart and Kaelbling, 2000] presentan un algoritmo de aprendizaje basado en casos que utiliza regresión localmente ponderada (*locally weighted regression, LWR*), llamado HEDGER. Los puntos del conjunto de casos son ponderados de acuerdo a una función de su distancia con respecto al punto que se pretende aproximar. Esta función se corresponde a una gaussiana de forma que cada peso se calcula como $\kappa_i = exp(-(\vec{q} - \vec{k})^2/h^2)$, donde

$\vec{q}$ es la concatenación del estado s_q y la acción a_q y $\vec{k_i}$ es el i-ésimo vecino en NN_q. El parámetro h determina el ancho de la función gaussiana. Para calcular la mejor acción en cada momento utilizan un algoritmo iterativo que muestrea n acciones diferentes cercanas al estado s_q y se predicen sus valores Q. Se aproxima la función cuadrática que mejor se adapta a estos puntos y se calcula el máximo.

Molineaux et al. [Molineaux *et al.*, 2008], en cambio, utilizan una aproximación basada en casos para tratar de aproximar la función de valor estado en lugar de la función de valor acción, pero siguiendo una estrategia similar que en los trabajos anteriores. No obstante, Molineaux et al. mantienen dos bases de casos en lugar de una sólo. En la primera base de casos, los casos tienen la forma $C_i = (s_i, a_i, s_{i+1})$, donde s_{i+1} representa el siguiente estado al que se transita. En la segunda base de casos, un caso está definido mediante $C_i = (s_i, v)$, dónde v representa el valor de la función de valor para el estado s_i. Los valores v asociados a los casos en la segunda base de casos, se actualizan utilizando una función de kernel gaussiana, $K(d) = exp(-d^2)$, que pondera la contribución de cada caso en el vecindario al cálculo final de la función de valor. Los casos de la primera base de casos se utilizan para determinar cuál es la mejor acción a ejecutar en cada momento. Para ello, para cada caso del vecindario predice cuál será el valor de la función de valor $V(s_{i+1})$ resultante de ejecutar la acción a_i en s_i. A partir de estos valores estimados se aproxima, mediante regresión cuadrática, la función que dada una acción a_i del vecindario nos devuelve el valor $V(s_i)$ del siguiente estado que se visita, y se calcula el máximo de esta función.

Gabel y Riedmiller [Gabel and Riedmiller, 2005] también aproximan la función de valor estado utilizando casos de la forma $C_i = (s_i, v)$, y utilizando la distancia euclídea para ponderar la importancia relativa de cada caso en el vecindario a la hora de computar el valor estimado de la función de valor, $\hat{V}(s)$, para un nuevo estado s. El artículo describe como la acción mejor acción en cada momento se selecciona explotando avariciosamente la función de valor V, pero no se dan más detalles acerca de cómo se lleva a cabo esta selección.

2.3.2. Discretización del espacio

El problema de la generalización es un problema que crece conforme crecen el número de atributos que definen cada estado y cada acción. Una posible solución es buscar una representación alternativa tanto del conjunto de estados y acciones. Las técnicas basadas en el vecino más cercano suelen emplearse por proporcionar una manera simple de dividir el espacio de estados en regiones, tal y como se muestra en la figura 2.7 [Duda and Hart, 1973; Gersho and Gray, 1991]. En este caso, cada punto del espacio continuo puede aproximarse mediante alguna medida de distancia a uno de entre los n centroides que definen las regiones en las que se ha particionado el espacio. En el ejemplo mostrado en la figura 2.7, cualquier punto perteneciente a la nube de puntos azul se aproxima al centroide y_1, cualquier punto en la nube naranja se aproxima al centroide y_2, y cualquier punto de la nube roja se aproxima al centroide y_3.

Estas regiones pueden obtenerse a través de discretizaciones uniformes del espacio de estados [Fernández and Borrajo, 1999; Munos, 2000]. En la discretización uniforme el rango de cada variable de estado se divide en intervalos iguales, y a cada uno de estos intervalos se le asigna un valor representativo. El valor representativo se escoge de forma que minimice la distorsión por cada intervalo por lo que dependerá de la medida de distorsión empleada. De esta forma, a cualquier valor de entrada que pertenezca a un determinado intervalo se le aproximará por el valor representativo de dicho intervalo. Supongamos una variable de estado $X \in [0, 2]$, y que el rango de esta variable se divide en 8 intervalos iguales, quedando

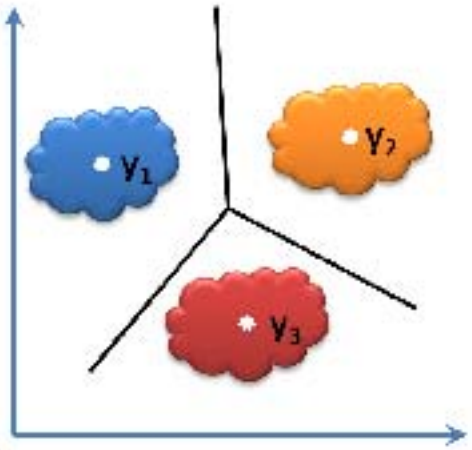


Figura 2.7: Ejemplo de partición del espacio de estados.

intervalos de 0.25. Los centroides en este cuantificador serían $y_1, y_2, ..., y_8$ donde cada uno de estos centroides corresponden al punto medio dentro de cada intervalo, es decir, $y_1 = 0{,}125$, $y_2 = 0{,}250$, ...,$y_8 = 1{,}875$. Así, podemos decir que para una entrada $x = 0{,}240$ la salida obtenida sería $y_1 = 0{,}125$, para una entrada $x = 0{,}275$ la salida obtenida sería $y_2 = 0{,}250$, y así sucesivamente.

No obstante, uno de los enfoques que ha sido utilizado con éxito para obtener esta división del espacio es la cuantificación vectorial o *Vector Quantization* (VQ), y que ha demostrado comportarse mejor que la cuantificación uniforme [Fernández and Borrajo, 1999]. La cuantificación puede ser considerada como un paso en el proceso que permite la digitalización de una señal analógica. Estrictamente hablando, es más correcto referirse a este proceso como la conversión de datos analógicos a digitales. Aunque para el caso que nos ocupa una definición más acertada de cuantificación se da en la definición 2.7.

DEFINICIÓN 2.7 ***Cuantificación***. *Proceso de representación de un conjunto de valores extenso (que podría ser infinito) con un número mucho menor de valores.*

La cuantificación vectorial consiste en agrupar n muestras de la señal de entrada, formando vectores *n-dimensionales* donde cada uno de estos vectores es representado por un vector representativo denominado centroide. Más formalmente, un cuantificador vectorial es un mapeo desde un vector en un espacio Euclídeo n-dimensional en un conjunto finito C, conteniendo N puntos de salida (llamados vectores de código o palabras de código). Es decir, se trata de una tranformación de la forma que se indica en la ecuación 2.52.

$$Q : \mathbb{R}^k \rightarrow C \tag{2.52}$$

donde $C = y_1, y_2, ..., y_N$ e $y_i \in \mathbb{R}^k$.

Al conjunto C se le llama alfabeto y tiene tamaño N, es decir, consta de N elementos diferentes, siendo cada uno de ellos un vector en $\mathbb{R}^k$. También se define la dimensión del espacio de los vectores que lo componen, k, como la dimensión del alfabeto. La resolución (o *code rate* o simplemente *rate*) de un cuantificador vectorial se define mediante la ecuación 2.53.

$$r = \frac{log_2 N}{k} \tag{2.53}$$

La ecuación 2.53 mide el número de bits por cada componente de vector usado para representar el vector de entrada. El *code rate* nos da una indicación de la precisión que es alcanzable si el cuantificador está bien diseñado. Asociado a cada uno de los N puntos del alfabeto tenemos una región o celda. La *i-ésima* celda está definida por la ecuación 2.54:

$$R_i = \{x \in \mathbb{R}^k : Q(x) = y_i\} \tag{2.54}$$

De la definición de celda se sigue que $\cup_i R_i = \mathbb{R}^k$ y $R_i \cap R_j = \phi$ para $i \neq j$ de forma que las celdas forman una partición de $\mathbb{R}^k$.

Una clase muy importante de cuantificadores vectoriales son los llamados cuantificadores Voronoi o de vecino más cercano. Un cuantificador de vecino más cercano es aquel cuantificador cuya partición en celdas viene dada por la ecuación 2.55.

$$R_i = \{x/d(x, y_i) \leq d(x, y_j), \forall j \in J\} \tag{2.55}$$

Donde el alfabeto está dado por $C = \{(y_j)\}$. Para medir el concepto de cercanía o lejanía se debe definir una medida de distancia entre cualquier par de vectores. Una medida de distorsión es una asignación de coste $d(x, y_j)$ asociada a la cuantificación del vector $x \in \mathbb{R}^k$ sobre un vector de reproducción y_j que pertenece al alfabeto. Por lo tanto, se dice que el cuantificador es bueno si la medida de distorsión es baja. Existen diversas formas de calcular la distancia $d(x, y_j)$ aunque es común emplear el error cuadrático definido en la ecuación 2.56.

$$d(x, y) = \sum_{i=1}^{K} (x_i - y_i)^2 \tag{2.56}$$

Para un alfabeto dado, se dice que una partición es óptima cuando para cada i, todos los puntos más cercanos al vector del alfabeto y_i que a cualquier otro vector del alfabeto y_j deben ser asignados a la región R_i. Para un conjunto de niveles de salida C, la partición óptima satisface que $Q(x) = y_i$ sólo si $d(x, y_i) \leq d(x, y_j) \forall j$. Esta condición es la misma que en los cuantificadores escalares. Si un vector de entrada, x, es equidistante a dos o más vectores del alfabeto se asigna a aquel vector con menor subíndice.

Se define el centroide de una partición dada R, $y^* = cent(R)$ como el vector que minimiza la distorsión media entre los puntos que pertenecen a R e y^*. Esto es, $y^* = cent(R)$ si:

$$E(d(x, y^*)|x \in R) \leq E(d(x, y)|x \in R) \forall y \in R \tag{2.57}$$

El cálculo del centroide se reduce al promedio aritmético que queda expresado en la ecuación 2.58.

$$cent(R) = \frac{1}{\|R\|} \sum_{i=1}^{\|R\|} x_i \tag{2.58}$$

Una vez obtenido el cuantificador vectorial que permite la representación alternativa del espacio de estados, es posible aplicar diferentes algoritmos de aprendizaje por refuerzo. Tal es el planteamiento del modelo *VQQL* [Fernández and Borrajo, 1999] que primero realiza una cuantificación vectorial del espacio de estados para después poder aplicar el algoritmo de aprendizaje por refuerzo *Q-Learning*. En el funcionamiento *en línea* de este modelo pueden distinguirse dos tareas bien diferenciadas que se describen brevemente a continuación:

- La primera tarea consiste en construir los cuantificadores vectoriales que posteriormente se emplearán en el proceso de aprendizaje. Para llevar a cabo esta tarea se recogen un conjunto de estados por los que ha transitado el agente en una iteración aleatoria con el entorno. Después, este conjunto es tomado como entrada por el algoritmo que obtiene el cuantificador vectorial para esos datos.

- En la segunda tarea, se utiliza el algoritmo *Q-Learning* para que el agente logre aprender una política de comportamiento óptima. Este algoritmo utiliza la llamada *tabla Q* para aproximar la función de valor-acción para cada par *(s, a)*. Esta *tabla Q* tendrá tantas filas como estados y tantas columnas como acciones posibles pueda ejecutar el agente. En este modelo, hay tantas filas en la *tabla Q* como componentes tenga el alfabeto construido a partir del Algoritmo de Lloyd Generalizado [Lloyd, 1957; Zanuy, 2000]. De esta forma, se ha pasado de la imposibilidad de almacenar la *tabla Q* en memoria debido al enorme número de estados a poder hacerlo con un número relativamente pequeño de estados.

El Algoritmo de Lloyd Generalizado (GLA), descrito en la tabla 2.8 necesita un conjunto de datos de entrenamiento, T, con los cuales construirá el cuantificador vectorial asociado. El algoritmo comienza con un alfabeto inicial. Entonces se genera un nuevo alfabeto (utilizando la condición de centroide) que sea óptima para la nueva partición creada. El nuevo alfabeto generado tiene una distorsión menor o igual al alfabeto de la iteración anterior. Esto implica que el algoritmo GLA es un método de descenso puesto que cada iteración reduce o no modifica la distorsión promedio obtenida en la iteración anterior. Además en cada iteración se produce un cambio local en el alfabeto, es decir, en cada iteración no se obtiene un alfabeto muy diferente del anterior. Esto muestra que una vez que el alfabeto inicial es elegido, el algoritmo nos llevará al mínimo local más cercano a éste en el espacio de todos los alfabetos posibles. Debido a que la función de distorsión es compleja seguramente tendrá un gran número de mínimos locales por lo que es difícil que el algoritmo GLA localice el óptimo global. De aquí se desprende que para obtener un buen resultado es muy importante un buen punto de partida.

Por lo tanto, uno de los problemas que se planteaban con el algoritmo GLA es la elección de un buen alfabeto inicial puesto que es crucial para después obtener unos buenos resultados. Se han planteado varias soluciones para resolver este problema que se describen brevemente a continuación:

- ***Random Coding***. Es el método más simple que consiste en inicializar de forma aleatoria los vectores que componen el alfabeto de acuerdo con la distribución de probabilidad de los vectores de entrada. Una de las opciones más simples consiste en seleccionar los N primeros vectores de entrada como componentes del alfabeto.

- ***Pruning (Podar)***. Este método se basa en la idea de que inicialmente todos los vectores de entrada son considerados como candidatos para integrar el alfabeto. Después, son eliminados uno o uno según cierto criterio hasta obtener el alfabeto deseado.

- ***Diseño de vecino más cercano dos a dos o clustering***. Los k vectores del conjunto de entrenamiento son considerados como vectores del alfabeto, cada uno de los cuales cuenta con su celda asociada en la partición del espacio de entrada. El objetivo de este algoritmo es agrupar los vectores hasta obtener el número adecuado de grupos. El alfabeto final estará compuesto por los centroides de estos grupos.

Algoritmo de Lloyd Generalizado (GLA) $(C_1, T, \theta) \rightarrow C_{t+1}$

Dados:

Un alfabeto inicial, $C_1 = \{y_j, j \in 1, ..., N\}$.

Un conjunto de datos de entrenamiento, T.

Un parámetro de parada, θ.

Inicializar:

Distorsión promedio en el periodo anterior, $D_{t-1} = 0$.

Contador de iteraciones, $t = 1$.

Ejecutar:

1. Dado un alfabeto, $C_t = \{y_j, j \in 1, ..., N\}$, se particiona el conjunto de entrenamiento T en celdas R_j usando la condición de vecino más cercano.

$$R_j = \{v \in T \; dist(v, y_j) \leq dist(v, y_i) \forall j \neq i\}$$

2. Usando la condición de centroide se calculan los centroides para la partición anterior para obtener el nuevo alfabeto C_{t+1}.

$$cent(R) = \frac{1}{\|R\|} \sum_{\vec{x}_i \in R} \vec{x}_i$$

3. Se calcula la distorsión promedio para C_{t+1}.

$$D_t = \frac{1}{M} \sum_{j=1}^{M} min(d(\vec{x}_j, \vec{y})), \vec{y} \in C_{t+1}$$

4. **Si** $(D_t - D_{t-1}) > \theta$ **hacer**

$D_{t-1} = D_t$

$t = t + 1$

Se vuelve al paso 1.

5. Devolver C_{t+1}

Tabla 2.8: Algoritmo de Lloyd Generalizado (GLA).

- ***Splitting***. Inicialmente se elige el centroide de la secuencia de entrada (y_0) como alfabeto de resolución 0 y con un solo nivel. Esta única palabra puede ser divida en dos palabras de código: $y_0 - e$ e $y_0 + e$ donde e es un vector de módulo pequeño. Este nuevo alfabeto tendrá dos elementos por lo cual no puede ser peor que el original. Como siguiente paso de divide cada uno de los vectores del alfabeto en dos, repitiendo lo anterior. Se continua de esta manera, hallando un alfabeto de resolución *r+1* partiendo de un buen alfabeto de resolución r.

Por lo tanto el cuantificador que se genera después del algoritmo de *Splitting* tendrá un número de componentes en el alfabeto de potencia 2. El algoritmo de *Splitting* es el utilizado también en esta Tesis. En realidad esta propuesta aparece en el mismo artículo en el cual aparece el algoritmo GLA, presentándose como una variante de éste [Linde *et al.*, 1980]. Este método de división va obteniendo alfabetos de tamaño creciente (cada uno el doble del anterior) de la siguiente manera:

- El alfabeto inicial $C_1 = \{y_1\}$, de tamaño 1, se corresponde con el centroide de todo el conjunto de datos de entrenamiento, T.
- Cada centroide del alfabeto anterior se divide en dos centroides diferentes.
- Se aplica el algoritmo GLA descrito en la tabla 2.8 utilizando como alfabeto inicial el alfabeto anterior.
- Se repite el proceso de división y relocalización (GLA) anterior hasta que el tamaño del alfabeto sea el deseado.

Este proceso de división integrado en el algoritmo GLA sirve para solucionar el problema de la selección del alfabeto inicial particionando el espacio en divisiones sucesivas. Por

último, aunque el modelo VQQL asocia la cuantificación vectorial al algoritmo de aprendizaje por refuerzo Q-Learning, se ha demostrado que también es posible asociarla con éxito a algoritmos $TD(\lambda)$, dando lugar al modelo $VQ - TD(\lambda)$ [García *et al.*, 2007].

Aunque la literatura es muy extensa en cuanto a la discretización del espacio de estados, no se puede decir lo mismo en cuanto a la discretización del espacio de acciones. En el algoritmo *Parti-game* [Moore and Atkeson, 1995], el espacio de estados se divide utilizando un árbol *kd-tree*. El espacio de acciones también se discretiza, de forma que en cada región definida por el árbol *kd-tree* las acciones disponibles consisten en cómo llegar a las regiones vecinas. El algoritmo construye un grafo de transiciones de estado, que puede ser resuelto por caminos mínimos. Además, utiliza un criterio *minimax* para determinar cuándo un grupo de regiones es demasiado grueso como para prevenir movimientos entre obstáculos o evitar ciclos. Estos grupos de regiones pueden ser divididos o refinados, obteniendo una mayor resolución de esa zona del espacio. El algoritmo *Binary Action Search* [Pazis and Lagoudakis, 2009], discretiza el espacio de acciones dividiéndolo en cada momento mediante preguntas binarias. Concretamente busca la mejor acción en el rango continuo $[a_{min}, a_{max}]$, usando un número finito de pasos de búsqueda binarios. La primera pregunta que se hace el algoritmo es: *Cuando estoy en el estado s, ¿debería incrementar o disminuir el valor de la acción* $a = (a_{max} + a_{min})/2$*?* La respuesta a esta pregunta eliminará la mitad del rango de la acción del espacio de búsqueda. La siguiente pregunta situará la acción en la mitad del intervalo del rango restante, y así sucesivamente. En general, cada paso de búsqueda eliminará la mitad del resto de posibles selecciones. El número de preguntas que se deben realizar depende de un parámetro N introducido por el usuario. El algoritmo *HOOT* ofrece otro ejemplo de discretización del espacio de acciones [Mansley *et al.*, 2011]. La búsqueda en este algoritmo selecciona cada vez más y más pequeñas particiones del espacio hasta que encuentra una hoja en el árbol de búsqueda y selecciona una acción.

Millán et al. [Millán *et al.*, 2002] en cambio, discretizan de forma separada el espacio de estados y el espacio de acciones. En su algoritmo *Continuous-Action Q-Learning* utilizan una aproximación llamada *topología incremental preservando mapas* [Millán, 1997] (*incremental topology preserving maps*) para particionar el espacio de estados. En esta aproximación una *unidad* representa a una región bien definida del espacio de estados. Para cada *unidad* se almacenan los valores Q de M acciones discretas. Las *unidades* se van añadiendo incrementalmente hasta cubrir las necesidades de representación del espacio de estados. La acción continua resultante en cada paso para un estado s percibido, se calcula como una media entre las acciones discretas de la *unidad ganadora* (la más cercana al estado s) ponderadas mediante sus valores Q. Este cálculo, tal como se explica en [Millán *et al.*, 2002], se lleva a cabo de la siguiente forma. Se considera a la *unidad* i la más cercana al estado s y a la acción a_l la acción con mayor valor Q correspondiente a la unidad ganadora i. Las acciones vecinas a la izquierda y la derecha de a_l se ponderan calculando la diferencia entre sus valores Q y el valor Q de la acción a_l. Por simplicidad, solo se considera una acción a cada lado de a_l. El peso correspondiente a la acción de la izquierda de a_l, a_{l-1}, se calcula como se muestra en la ecuación 2.59, mientras que el peso correspondiente a la acción de la derecha de a_l, a_{l+1}, se calcula mediante la ecuación 2.60

$$left = \frac{1}{2 + (Q(i,l) - Q(i,l-1))^2} \tag{2.59}$$

$$right = \frac{1}{2 + (Q(i,l) - Q(i,l+1))^2} \tag{2.60}$$

La acción continua resultante se calcula utilizando los pesos calculados en las ecuaciones 2.59 y 2.60 tal como se muestra en la ecuación 2.61.

$$a = a_l + right \times (a_{l+1} + a_l) + left \times (a_{l-1} - a_l) \tag{2.61}$$

De esta manera el agente explora siempre acciones que se mantienen cerca de la acción óptima, de forma que cuanto más peso tenga una acción discreta más *atraerá* hacia sí misma el valor resultante de la acción continua final, a. El valor Q de la acción a se calcula como se muestra en la ecuación 2.62.

$$Q(i,a) = \frac{Q(i,l) + right \times Q(i,l+1) + left \times Q(i,l-1)}{1 + right + left} \tag{2.62}$$

La ecuación 2.62 se utiliza para calcular el valor Q de una acción continua a. Este valor será utilizando posteriormente para actualizar todos los valores Q de todas las *unidades* y todas las acciones M correspondientes a cada unidad.

2.3.3. Métodos de búsqueda directa de la política

Los métodos de búsqueda directa de la política (*Direct Policy Search Methods*), forman una importante rama dentro de los algoritmos de aprendizaje por refuerzo [Kaelbling *et al.*, 1996]. Estos métodos de resolución buscan directamente en el espacio de comportamientos (políticas) en lugar de buscar en el espacio de funciones de valor (métodos de diferencia temporal) [Powell, 1998; Peshkin, 2002; Taylor *et al.*, 2007]. Por lo tanto, la diferencia fundamental entre estos métodos y los métodos de diferencia temporal es que estos últimos definen una función de valor de la forma $f(s,a) \rightarrow value$ y los métodos de búsqueda directa de la política aprenden directamente una política de decisión $f(s) \rightarrow a$, que mapea de forma directa el estado s con la acción a (Figura 2.8).

Figura 2.8: Búsqueda directa de la política (a) frente a métodos de diferencia temporal (b).

La política, en este caso, se puede entender como una *caja negra* cuya entrada es el estado donde se encuentra el agente y la salida la acción asociada a ese determinado estado. Existen diferentes formas de representación de la política dentro de esta *caja negra*, así como diferentes métodos de búsqueda de la política óptima. Entre estos métodos destacan los que se basan en la búsqueda en el espacio de codificaciones de la estructura de una red de neuronas enmarcados dentro del aprendizaje por refuerzo evolutivo [Moriarty and Miikkulainen, 1997; Moriarty *et al.*, 1999; Martín and de Lope Asiaín, 2009]. Estos métodos representan otra forma de optimizar un vector de parámetros mediante el uso de algoritmos evolutivos [Holland, 1962; Holland, 1992; Bäck and Schwefel, 1993]. En esta Tesis no se pretende entrar en más

Aprendizaje por refuerzo evolutivo $(n^i, n^h, n^o, K, P_1, N, M, A, B, S, \sigma, \varsigma) \rightarrow P_e$

Dados:
  Dada la topología de las redes de neuronas, n^i, n^h y n^o.
  Un número máximo de episodios a ejecutar, K.
  La población inicial, $P_1 = \emptyset$.
  El número de individuos de la población, N.
  El número de los mejores individuos para la correlación, M.
  Los pesos para la correlación, A, B.
  El número de individuos para la mutación suave, S.
  Los factores de mutación suave σ y fuerte ς, con $\varsigma > \sigma$.
1. Inicializar:
  El número de individuos para la mutación fuerte, $F = N - (M + S)$.
  Contador con el número de generaciones, $e = 1$
  Contador con el individuo seleccionado, $i = 1$.
  Para $n = 1$ a N **hacer**
    Crear la red de neuronas $NN(k)$ con $n^i + bias, n^h$ y n^o.
    Inicializar los pesos de la red de neuronas $NN(k)$ de forma aleatoria
    Incluir $NN(k)$ en P_e
2. Ejecutar:
  Para $k = 1$ a K **hacer**
    Elegir la red de neuronas $NN(i) \in P_e$.
    Ejecutar el episodio de aprendizaje k utilizando la política generada por $NN(i)$.
    Recibir el refuerzo acumulado en ese episodio obtenido por $NN(i)$, $R(i)$
    Si $i = N$ **hacer** ;; Se evoluciona la población
      $P_{e+1} = evolucionar(P_e, M, S, F, A, B, \sigma, \varsigma)$
      $e \leftarrow e + 1$.
      $i \leftarrow 0$.
    $i \leftarrow i + 1$.
3. Devolver la última población generada P_e

Tabla 2.9: Algoritmo para aprendizaje por refuerzo evolutivo.

detalles sobre los algoritmos evolutivos. Textos introductorios sobre algoritmos evolutivos se pueden encontrar en [Bäck, 1996] y en [Eiben and Smith, 2008].

Whiteson y Stone [Whiteson and Stone, 2006] utilizan una aproximación que selecciona automáticamente la función de aproximación mediante la utilización de técnicas evolutivas. En el método propuesto combinaban técnicas neuro-evolutivas de optimización [Volná, 2010], con el algoritmo Q-learning. Existen tres formas fundamentales de evolución de redes de neuronas: la evolución de los pesos de las conexiones, la evolución de las arquitecturas, y la evolución de las reglas de aprendizaje. Sin embargo, Whiteson y Stone utilizan la técnica NEAT (**N**euro**E**volution of **A**ugmenting **T**opologies) [Stanley and Miikkulainen, 2002b] para automáticamente buscar la topología apropiada y los pesos iniciales de la función de aproximación. La combinación de esta técnica con Q-learning, da lugar al algoritmo NEAT+Q. Este algoritmo se ha probado con éxito en dominios simples como el *mountain car*, y un dominio poco conocido más complejo, el servidor de planificación de tareas. No obstante, para este último dominio y previo a la utilización de la técnica propuesta, se realiza una discretización tanto del espacio de estados como del espacio de acciones. En otros casos en cambio, sólo se evolucionan los pesos de la red de neuronas [Koppejan and Whiteson, 2009; Martín and de Lope Asiaín, 2009; Koppejan and Whiteson, 2011].

Martín et al. [Martín and de Lope Asiaín, 2009] presentan un algoritmo de aprendizaje por refuerzo evolutivo cuya estrategia de evolución es la adaptación de los pesos de una

red de neuronas, y que servirá de ejemplo para ilustrar el funcionamiento general de los algoritmos de aprendizaje por refuerzo evolutivo. Cada agente maneja una población P_e de N soluciones individuales o redes de neuronas, $NN(i)$ con $i \in [1 \dots N]$. En este caso, las soluciones individuales son redes de neuronas con la misma topología pero con diferentes pesos. Cada red de neuronas es un vector representado por:

- El número de nodos de entrada, $n^i + bias$.
- El número de nodos de la capa oculta, n^h.
- El número de nodos de salida, n^o.

Así, cada red de neuronas $NN(i)$ queda representada por el vector de pesos $w(i)$ de cardinalidad $W = |w^i| + |w^o|$ dónde $|w^i| = (n^i + 1) \times n^h$ son los pesos de la capa de entrada y $|w^o| = n^h \times n^o$ los pesos de la capa de salida.

La tabla 2.9 describe en detalle un algoritmo basado en aprendizaje por refuerzo evolutivo. En el paso de inicialización se crea la población inicial, P_e, con redes de neuronas de tantos nodos de entrada $n^i + bias$, tantos nodos en la capa oculta n^h y tantas salidas n^o como los especificados como parámetros del algoritmo. En la ejecución, en cada episodio se elige un individuo $NN(i) \in P_e$ para proceder a su evaluación. Cada individuo recibe como entradas el estado percibido por el agente del entorno y devuelve como salida la acción a ser ejecutada. En este caso, el fitness del individuo será el refuerzo acumulado $R(i)$ generado por el individuo $NN(i)$ en el episodio k. Cuando se han evaluado todos los individuos de la población, se procede a la evolución de la misma. La evolución se describe en la tabla 2.10.

	evolucionar $(P_e, M, S, F, A, B, \sigma, \varsigma) \rightarrow P_{e+1}$
00	**1. Inicializar:**
01	Ordenar de mayor a menor los individuos $NN(i) \in P_e$ atendiendo a $R(i)$
02	Inicializar el contador $inicio \leftarrow 1$.
03	**2. Correlar:**
04	$inicio \leftarrow M + 1$.
05	**Para** $n = inicio$ a $(inicio + M)$ **hacer**
06	Actualizar los pesos de $NN(n)$, $w(n)$, usando los pesos de $NN(n - M)$ y $NN(n - M + 1)$ siguiendo
07	la ecuación:
08	$w(n) = A \times w(n - M) + B \times w(n - M + 1)$
09	**3. Mutar suave:**
10	$inicio \leftarrow inicio + M$.
11	**Para** $n = inicio$ a $(inicio + S)$ **hacer**
12	Seleccionar un aleatorio $r \in [1, M]$.
13	Seleccionar un aleatorio $r' \in [-\sigma, \sigma]$.
14	Calcular los nuevos pesos del individuo n, $w(n) = w(r) + r'$.
15	**4. Mutar fuerte:**
16	$inicio \leftarrow inicio + S$.
17	**Para** $n = inicio$ a $(inicio + F)$ **hacer**
18	Seleccionar un aleatorio $r \in [1, M]$.
19	Seleccionar un aleatorio $r' \in [-\varsigma, \varsigma]$.
20	Calcular los nuevos pesos del individuo n, $w(n) = w(r) + r'$.
21	**5. Devolver** la población P_{e+1}

Tabla 2.10: Evolución de la población en el algoritmo de aprendizaje por refuerzo evolutivo.

La evolución, tal y como se muestra en la figura 2.9, consta de tres pasos fundamentales: correlación, mutación suave de los pesos y mutación fuerte. La correlación se produce sobre

los M mejores individuos de la población. De esta forma se asegura que se producen nuevos individuos en la población muy similares a los M mejores individuos. Esto produce una cadena de correlaciones que se propaga rápidamente sobre toda la población. La mutación fuerte y suave se produce nuevamente sobre los M mejores individuos de la población y sirve para introducir variedad en la nueva población generada.

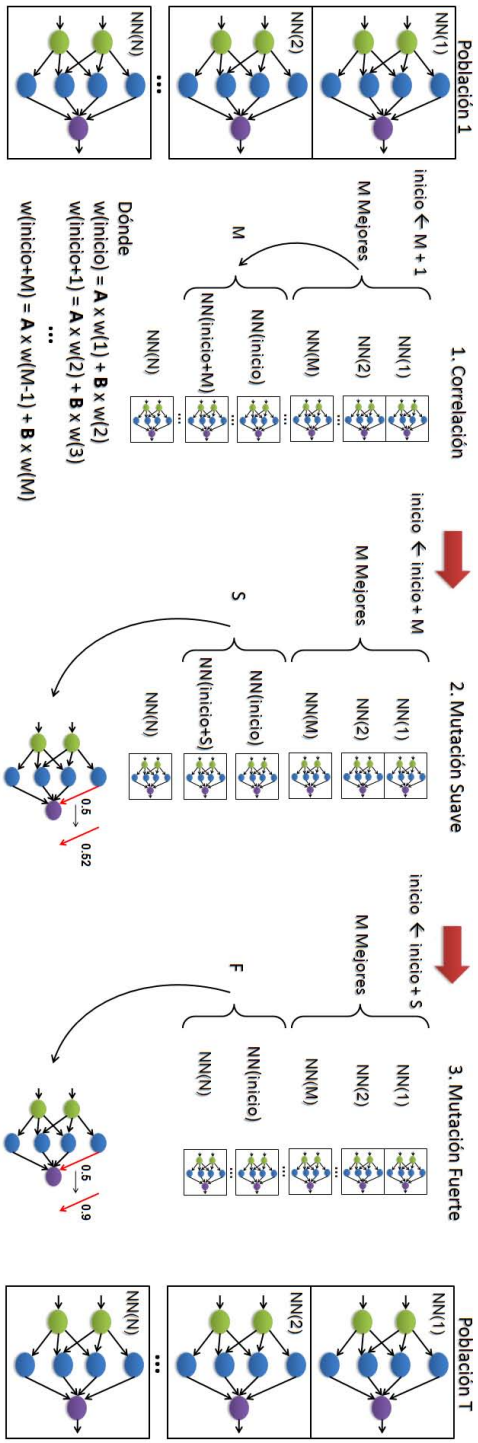


Figura 2.9: Aprendizaje por refuerzo evolutivo con redes de neuronas.

2.3.4. Discusión

La revisión exhaustiva de la literatura pone de manifiesto que todavía existen ciertas deficiencias para el aprendizaje por refuerzo en dominios con espacios de estados y acciones continuos y de grandes dimensiones. Uno de los grandes desafíos en aprendizaje por refuerzo consiste en tratar de abandonar los problemas de *juguete* a los que a menudo se enfrenta y dar el salto a la resolución de problemas reales. En la mayor parte de las ocasiones estos problemas reales contienen un gran número estados y acciones, a menudo continuos, inabordables con las técnicas tradicionales de aprendizaje por refuerzo, que asumen unos espacios de estados y acciones discretos. La revisión literaria indica que existe una gran cantidad de trabajos que resuelven con éxito el problema de la generalización del espacio de estados, ya sean utilizando técnicas de discretización [García *et al.*, 2007; López-Bueno *et al.*, 2009], o aproximación de funciones [Stone *et al.*, 2005]. En cambio, es mucho menor el número de trabajos que afrontan el problema de la generalización del espacio de acciones [Lazaric *et al.*, 2007], y aún menor la cantidad de trabajos donde se afronta directamente el problema de la generalización de los espacios de estados y acciones de forma conjunta [Santamaría *et al.*, 1998].

En el caso de la generalización del espacio de estado y acciones, es mayoritario el uso de aproximadores de funciones, y más concretamente la utilización de redes de neuronas. En caso de aproximar la función de valor $V(s)$, las entradas de la red de neuronas estarían compuestas por cada una de las características que describen el estado, y la salida sería el valor predicho, $\hat{V}(s)$, para la función de valor $V(s)$ en el estado s [van Hasselt and Wiering, 2007]. En el caso de aproximar la función de valor-acción $Q(s, a)$, además de las entradas a la red correspondientes a las características de los estados, se añade una entrada por cada una de las características que describen las acciones. La salida en este caso, sería el valor predicho, $\hat{Q}(s, a)$, para la función de valor-acción $Q(s, a)$ cuando la entrada es el par $< s, a >$ [Baird and Klopf, 1993; Prokhorov and Wunsch, 1997]. Los pesos de la red de neuronas se van adaptando siguiendo técnicas de descenso de gradiente. Con la utilización de este tipo de aproximaciones, la arquitectura de la red de neuronas crece exponencialmente conforme crecen el número de características que describen los estados y acciones, de ahí que solo se utilicen en dominios donde el número de estas características es muy reducido.

Existen tres problemas fundamentales a la hora de utilizar redes de neuronas para aproximar las funciones de valor. Por un lado, existe el problema de que en muchos casos no se garantiza la convergencia y ofrece rendimientos muy pobres incluso en dominios simples [Baird, 1995; Justin Boyan, 1995]. Por otro lado está el problema de extraer de la red de neuronas conocimiento sobre qué acción es la que maximiza el valor de la función de valor para un determinado estado. En este caso, se suele utilizar otro aproximador de funciones, que nuevamente suele ser una red de neuronas, que predice en función del aproximador de la función de valor, la acción a ejecutar en un determinado estado (siguiendo la arquitectura actor-crítico presentada en la sección 2.3.1 en el algoritmo 2.6) [van Hasselt and Wiering, 2007]. La utilización de este tipo de aproximaciones solo es aplicable en dominios muy simples, puesto que si es complicado garantizar la convergencia de una sola red de neuronas, lo es el doble cuando se utilizan dos redes de neuronas. Por último, se encuentra el problema de elegir la topología adecuada que permita a la red de neuronas enfrentarse al problema con éxito. La elección de la topología y de su dimensión, así como la función que debe implementar cada nodo, es crucial para la correcta convergencia de la red de neuronas, y seleccionar una equivocada puede resultar en un red de neuronas que no es capaz de aprender nada [Weigend *et al.*, 1990; Nolfi and Floreano, 2004]. Existen cuatro

formas de llevar a cabo esta selección. La primera, la más utilizada, consiste en la prueba y error (e.g. empezar con una red de neuronas *pequeña* e ir incrementando su tamaño hasta que se alcanzan los resultados deseados, o viceversa, comenzar con una red de neuronas *grande* e ir eliminando capas y nodos). La segunda consiste en dejar que sea un experto el que decida cuál debería ser la topología en función de su experiencia [Jacobs *et al.*, 1991; Nolfi and Floreano, 2004]. La tercera se basa en la evolución de las conexiones de la red y su topología (se añaden nuevos nodos o se reorganiza la red) mediante técnicas evolutivas [Rudolph, 1996; Yaeger and Sporns, 2008]. La cuarta y última forma de selección se basa en la eliminación de elementos redundantes y poda de redes de neuronas grandes [Hassibi *et al.*, 1993]. Puede comprobarse por tanto, que la correcta selección de la topología de una red de neuronas constituye un problema lo suficientemente complejo por sí solo.

Santamaría et al. [Santamaría *et al.*, 1998] utilizan dos aproximadores también, uno para aproximar el valor de la función de valor-acción y otro para determinar cuál es la política de comportamiento a seguir. En el caso del segundo aproximador emplea el algoritmo *One-Step Search* (Tabla 2.5) para buscar la acción óptima entre un conjunto discreto de acciones. En este caso hay que decidir qué incremento Δ_a utilizar. Incrementos muy pequeños puede tener un coste computacional muy alto debido a la gran cantidad de pruebas que se realizan. En cambio, incrementos muy grandes pueden conllevar probar pocas acciones y por lo tanto no probar acciones con valores de la función Q elevados.

En cambio, la utilización de técnicas de máquinas de vectores de soporte (*support vector machines*) y su combinación con funciones *Kernel* para aproximar la función de valor-estado [Goto *et al.*, 2002; Hu *et al.*, 2006] o la función de valor-acción [Xu *et al.*, 2007; Xu *et al.*, 2011], presenta el problema de seleccionar la función *Kernel* adecuada, de la cual depende el rendimiento del algoritmo, y que muchas veces se obtiene mediante prueba y error [B. Bethke and J. How and A. Ozdaglar, 2008], probando de un conjunto muy amplio de posibles funciones *Kernel* (capítulo 4, [Rasmussen and Williams, 2006]). En muchos casos, las técnicas de máquinas de vectores de soporte se emplean para generalizar el espacio de estados considerando un espacio de acciones discreto y reducido. Goto et al. [Goto *et al.*, 2002] por ejemplo, recogen primero un conjunto significativo de tuplas (s_i, a_k, s_{i+1}), donde a_k es la acción que se ejecuta cuando el agente se encontraba en el estado s_i haciéndole transitar al estado s_{i+1}. Después, y mediante la utilización de este tipo de técnicas, a cada estado s_i lo etiquetan como positivo si s_{i+1} es un estado meta o si te permite alcanzar la meta, y como negativo en caso contrario. El método selecciona una acción aleatoria con probabilidad ϵ, y la acción que clasifica al estado s de entrada como positivo con probabilidad $1-\epsilon$. No obstante, solo se presentan resultados del algoritmo para un dominio reducido con un espacio de estados de 4 dimensiones y con 4 posibles acciones discretas de 1 dimensión. La utilización de este método queda por tanto reducida a dominios con espacios de acciones discretos, y que además tengan estados meta para poder realizar la clasificación entre estados positivos y negativos (algo que no ocurre en todos los dominios). Hu et al. [Hu *et al.*, 2006] también utilizan las máquinas de vectores de soporte para generalizar el espacio de estados considerando, igual que en el caso anterior, un conjunto discreto de acciones. No obstante, también se utilizan estas técnicas para la aproximación de funciones en espacios de estados y acciones continuos. En estos casos, se tiene el mismo problema que se describía anteriormente sobre cómo extraer conocimiento del aproximador acerca de cuál es la mejor acción a ejecutar en un determinado momento. En el método propuesto por Xu et al. [Xu *et al.*, 2011] la acción óptima se calcula en cada momento basándose en la información que proporcionan los gradientes de la función de valor-acción aproximada, un cálculo que requiere un mayor coste computacional a medida que crece el número de dimensiones de las

acciones. En cualquier caso, para este método solo se presentan resultados para problemas de dimensionalidad reducida, como el *mountain-car* (2 dimensiones para el estado, la posición del coche y la velocidad, y 1 dimensión para la acción, la aceleración), y el *cart-pole* (4 dimensiones para el estado, y 1 dimensión para la acción). En otros trabajos se emplean otros métodos de selección de la mejor acción utilizando técnicas de iteración de la política en un algoritmo llamado *KLSPI* (***K****ernel-based* ***L****east* ***S****quare* ***P****olicy* ***I****teration*) [Xu *et al.*, 2007]. En este algoritmo, se selecciona de forma avariciosa la mejor acción de acuerdo a la última política calculada, aunque no se dan detalles de los cálculos necesarios para llevar a cabo esta selección. Además, solo se proporcionan resultados en dominios con espacios de estados y acciones continuos de baja dimensionalidad como son el *acrobot* (4 dimensiones para el estado, correspondientes a los ángulos y la velocidad angular de los brazos, y 1 dimensión para la acción, la fuerza de torsión), y el problema del control de la dirección de un barco (4 dimensiones para el estado, correspondientes a la velocidad y la posición del barco, y 1 dimensión para la acción, la dirección del barco). En otros casos, previamente a la utilización del algoritmo de aprendizaje, se discretiza de forma uniforme el espacio de acciones de naturaleza continua con el fin de evaluar más fácilmente la mejor acción en cada momento [Röttger and Liehr, 2009].

Por otro lado, los aproximadores basados en técnicas basadas en memoria o CBR, también presentan algunos problemas [Santamaría *et al.*, 1998; Smart and Kaelbling, 2000; Molineaux *et al.*, 2008; Gabel and Riedmiller, 2005]. En el caso de Santamaria et al. [Santamaría *et al.*, 1998], se propone una técnica que permite considerar al espacio de estados como un espacio continuo mediante la utilización de funciones kernel. Sin embargo, la búsqueda de la mejor acción a ejecutar en cada momento se realiza utilizando nuevamente el algoritmo *One-Step Search* 2.5 con los problemas que se comentaban anteriormente. En el caso del algoritmo HEDGER [Smart and Kaelbling, 2000], la búsqueda de la acción óptima se realiza aproximando la función de valor-acción para n acciones discretas diferentes mediante regresión cuadrática, y buscando el máximo de la función aproximada. En este caso, como los propios autores comentan en su artículo, el valor del máximo resultante depende de una manera crítica de las n acciones discretas seleccionadas al inicio para llevar a cabo esta aproximación. Además, este método solo se ha probado en dominios de baja dimensionalidad como el *mountain car*, y el problema de recorrer un pasillo (*following-corridor task*). Molineaux et al. [Molineaux *et al.*, 2008] utilizan, en cambio, dos bases de casos en lugar de solo una, lo que complica significativamente el modelo de aprendizaje en cuanto a tiempo de recuperación de los casos y a requerimientos de almacenamiento. Su aplicación quedará por tanto limitada a dominios simples donde no se requieran unas bases de casos excesivamente grandes. Por último, Gabel y Riedmiller [Gabel and Riedmiller, 2005] aproximan la función de valor ponderando la contribución de cada vecino utilizando la distancia euclídea, y explotando de forma avariciosa esta la función de valor aproximada. No obstante, no se dan detalles de cómo se lleva a cabo esta explotación que permita seleccionar la acción óptima en cada momento.

En cuanto a las técnicas de discretización existentes, la mayoría aborda problemas con un espacio de estados continuo y acciones discretas [Fernández and Borrajo, 1999; García *et al.*, 2007] y son muy pocos los trabajos que tratan de discretizar tanto el espacio de estados como el de acciones. Los métodos de discretización uniforme requieren de un proceso de prueba y error con el fin de encontrar la discretización adecuada [Fernández and Borrajo, 2002], proceso que se complica si además de discretizar el espacio de estados hay que discretizar el espacio de acciones. Además, una 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]. El algoritmo *Parti-game* de Munos [Munos, 2000] discretizan los espacios de estados y acciones utilizando un árbol *kd*, de tal forma que se va incrementando la resolución de la discretización en aquellas regiones donde hay gran varianza en la función de valor para los estados de dicha región. Estos métodos, no obstante, 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]. Además, estas técnicas también presentan la necesidad de incluir un parámetro que define la resolución que se debe alcanzar, lo que requiere de un proceso de refinamiento de este parámetro. Pazis et al. muestran los resultados de su algoritmo *Binary Action Search* [Pazis and Lagoudakis, 2009] en dominios pequeños donde las acciones son solamente de una dimensión, haciendo difícil creer que esta técnica es generalizable a dominios donde las acciones tengan más dimensiones. Además, la principal dificultad de este algoritmo radica en decidir si *incrementar* o *disminuir* el valor de una acción cuando se está en cierto estado s. En cuanto al algoritmo *HOOT* [Mansley *et al.*, 2011], además de aportar resultados en dominios sencillos de no más de 4 variables para la acción, el espacio de estados se discretiza de forma uniforme en 20 intervalos iguales por cada variable de estado. Por lo tanto, se centra exclusivamente en ofrecer una técnica eficiente para la discretización del espacio de acciones en dominios sencillos, pero no aborda de forma conjunta el problema de la discretización de los espacios de estados y acciones. El algoritmo *Continuous Q-Learning* [Millán *et al.*, 2002] es un claro ejemplo de algoritmo que utiliza la discretización de los espacios de estados y acciones. No obstante, la acción ejecutada en cada momento es una acción continua calculada a partir de los valores Q de las M acciones discretas correspondientes a la *unidad* ganadora. Los valores Q se utilizan para ponderar la contribución de cada acción discreta en torno a la acción óptima en ese momento tal como se indicaba en la ecuación 2.61. No obstante, el valor Q es el único que se tiene en cuenta para calcular el peso de una acción discreta. No se tiene en cuenta la distancia entre la acción discreta y la acción óptima, de forma que si la acción discreta tiene un valor Q alto pero está muy alejada de la acción óptima tendrá menos peso que otra acción con un valor Q menor pero que se encuentra más cerca de la acción óptima. Por lo tanto, esta técnica sólo es válida cuando se tienen discretizaciones uniformes del espacio de acciones.

El aprendizaje por refuerzo evolutivo, exponente de las técnicas de búsqueda directa de la política (Sección 2.3.3), también ha sido utilizado en la literatura como una forma de sortear el problema de la *maldición* de la dimensionalidad (*curse of dimensionality*) en aprendizaje por refuerzo. No obstante, se suelen utilizar en problemas donde los espacios de estados y acciones son reducidos, porque las probabilidades de una convergencia exitosa se disminuyen conforme aumenta la complejidad del problema. Koppejan et al. [Koppejan and Whiteson, 2011] y Martín et al. [Martín and de Lope Asiaín, 2009] son un ejemplo de este hecho. Además, Koppejan et al. utilizan una topología propuesta por un experto, donde, además, se particionan los espacios de estados y acciones reduciendo de esta forma la complejidad del problema (se calcula el valor de cada dimensión de una acción por separado, y este valor sólo depende de un subconjunto de características del estado). En otros casos, otro tipo de aproximaciones de aprendizaje por refuerzo evolutivo, donde no solo se evolucionan los pesos de la red de neuronas sino también la topología (conocidas como ***N****euro****E****volution of* ***A****ugmenting* ***T****opologies* [Stanley and Miikkulainen, 2002b]), también han tenido resultados exitosos en dominios de baja dimensionalidad como el *Cart-Pole* [Stanley and Miikkulainen, 2002a] (acción con una sola dimensión), o *keepaway* [Taylor *et al.*, 2006; Taylor *et al.*, 2007] (acción con una sola dimensión de naturaleza discreta).

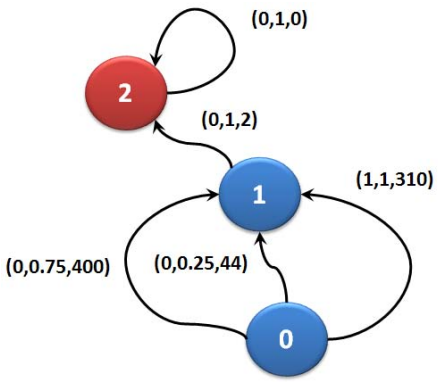
2.4. Aprendizaje por Refuerzo con Riesgo

Uno de los problemas que plantea el aprendizaje por refuerzo es el de la exploración segura, donde la exploración del entorno se realiza tratando de minimizar los daños en el sistema de aprendizaje. Las técnicas tradicionales de exploración como $\epsilon - greedy$ o Boltzmann son no seguras debido a la componente aleatoria que tienen a la hora se seleccionar acciones. La noción de riesgo en aprendizaje por refuerzo está relacionada con el hecho de que incluso una política óptima podría tener un rendimiento pobre en algunos casos, debido a la naturaleza estocástica del problema.

En el caso del aprendizaje seguro, existen fundamentalmente dos formas diferentes de entender el *riesgo*. Por un lado están las aproximaciones que entienden el riesgo como una varianza en el refuerzo acumulado, R, con $R = \sum_0^\infty \gamma^t r_t$, y por otro las que asocian el riesgo con la identificación de estados peligrosos que nunca deben de ser visitados. Otro conjunto de aproximaciones que no hablan explícitamente del concepto de riesgo están basadas en la utilización de la experiencia recogida a partir de demostraciones realizadas por expertos que realizan la tarea de forma segura. A continuación se describen cada una de estas aproximaciones.

2.4.1. Aproximaciones basadas en la Varianza del Refuerzo Acumulado

Muchas aproximaciones *risk-averse* en aprendizaje por refuerzo están basadas en la varianza de $R = \sum_0^\infty \gamma^t r_t$, o con sus peores valores. Un ejemplo de estas aproximaciones son conocidas como las de Control del Peor de los Casos (*Worst Case Control*) donde se trata de optimizar el peor resultado de R [Coraluppi and Marcus, 1999; Heger, 1994]. El MDP que se muestra en la figura 2.10, recogido de [Heger, 1994], sirve para ilustrar en qué consiste esta aproximación. En este MDP, denotado por el grafo de transiciones entre estados, hay tres estados correspondientes a los círculos. Una transición desde el estado i al estado j se representa mediante las flechas etiquetadas. Cada una de estas etiquetas consta de tres valores: la primera corresponde a una acción a admisible para el estado i, la segunda es la probabilidad de transitar al estado j cuando en el estado i se ejecuta la acción a, y el último se refiere al refuerzo recibido en la transición. Hay dos posibles políticas de comportamiento estacionarias π y μ.



s	π(s)	μ(s)
0	0	1
1	0	0
2	0	0

Figura 2.10: Ejemplo de MDP para explicar la aproximación de varianza del refuerzo acumulado en aprendizaje por refuerzo con riesgo.

En las aproximaciones tradicionales de aprendizaje por refuerzo, conocidas en el ámbito

del aprendizaje por refuerzo seguro como de riesgo neutro (*risk-neutral approaches*) [Mihatsch and Neuneier, 2002], el objetivo, se basa en la ecuación 2.63, introducida ya en la sección 2.2.2 para definir el valor de un estado.

$$V^{\pi}(s) = E(R^{\pi}|s_t = s) = E\left(\sum_{t=0}^{\infty} \gamma_t r_t\right) \tag{2.63}$$

Mediante la ecuación 2.63 se trata de maximizar el refuerzo acumulado. En la figura 2.10, se tendría que para la política de comportamiento estacionaria π, $E(R^{\pi}|s_t = 0) = 311 + 2\gamma$, y para la política de comportamiento estacionaria μ, $E(R^{\mu}|s_t = 0) = 310 + 2\gamma$. Es decir, atendiendo al objetivo marcado por la ecuación 2.63, se podría decir que las políticas π y μ tienen un comportamiento similar en cuanto a rendimiento se refiere, siendo la política π la política óptima. En las aproximaciones basadas en el peor de los casos, el objetivo de la ecuación 2.63 se cambia por el definido en la ecuación 2.64.

$$V^{\pi}(s) = minimo\left(\sum_{t=0}^{\infty} \gamma_t r_t\right) \tag{2.64}$$

De esta forma, y atendiendo a este nuevo objetivo, se tendría que para la política π, $V^{\pi}(0) = 44 + 2\gamma$, y para la política μ, $V^{\mu}(0) = 310 + 2\gamma$, decantándose claramente la balanza en favor de la política de comportamiento μ. Es decir, en este tipo de aproximaciones una política es considerada óptima si el retorno R en el peor de los casos es mayor. En cualquier caso esta aproximación es demasiado restrictiva porque tiene en cuenta escenarios raros, aunque estos sean muy poco probables de aparecer.

El parámetro $\alpha - value$ del retorno $\hat{m_{\alpha}}$ introducido por [Heger, 1994] puede ser considerado como una extensión de la aproximación del peor de los casos para MDPs. Este concepto establece que los retornos $R < \hat{m_{\alpha}}$ de una política que ocurren con una probabilidad menor que α no son tenidos en cuenta. El algoritmo, por tanto, es menos pesimista que las aproximaciones basadas en el peor de los casos puras, porque escenarios extremos, que raramente ocurren, no tienen efecto en la política. Heger et al. [Heger, 1994] también introduce la idea de ponderar el retorno y el riesgo, llamándolo *expected value-variance criterion*. De esta forma, la política seleccionada es aquella que maximiza $E(R) - K \times V(R)$, donde $E(R)$ es la esperanza del retorno, $V(R)$ es la varianza del retorno, es decir, el riesgo (cuanto mayor es la varianza mayor es el riesgo), y K es una constante referida al grado de aversión al riego (valores grandes de K implica una mayor aversión al riesgo).

En otros casos, el parámetro R es transformado para reflejar una medida subjetiva de utilidad (e.g. una función exponencial de utilidad), donde en vez de maximizar el valor esperado de R, ahora el objetivo es maximizar $U = \beta^{-1} log E(e^{\beta R})$, donde β es un parámetro y R es el retorno habitual [Liu *et al.*, 2003]. Dependiendo del valor del parámetro β, políticas con una gran varianza $V(R)$ son penalizadas ($\beta < 0$) o reforzadas ($\beta > 0$). Mihatsch y Neuneier [Mihatsch and Neuneier, 2002] también consideran la peor salida posible de una política, y asocian el concepto de riesgo a la variabilidad de R. En su trabajo, demuestran que su algoritmo interpola entre un comportamiento de riesgo neutro (*risk-neutral*) y el criterio del peor de los casos, y tiene el mismo comportamiento en el límite que las funciones exponenciales de utilidad. El algoritmo de Neuneier and Mihatsch tiene un parámetro $\kappa \in (-1{,}0, 1{,}0)$ que permite alternar entre un comportamiento de aversión al riesgo (*risk-averse behavior*) ($\kappa \to 0$) y un comportamiento de búsqueda de riesgo (*risk-seeking behavior*) ($\kappa \to -1$). Morimura et al. [Morimura *et al.*, 2010b; Morimura *et al.*, 2010a] utiliza una

aproximación sensible al riesgo (*risk-sensitive approach*) estimando la densidad del retorno R, lo cual le permite manejar diferentes criterios de riesgo. Usando esta misma filosofía, Sato et al. [Sato *et al.*, 2002] introducen un algoritmo para la toma de decisiones usando la media y la varianza de las distribuciones del retorno.

Cabe destacar que este tipo de aproximaciones basadas en la varianza del retorno acumulado no son adecuadas para algunos tipos de problemas en los cuales políticas con una pequeña variación pueden conllevar un gran riesgo (e.g. que el robot se choque, que el helicóptero se estrelle). Geibel et al. ponen como ejemplos en este caso el problema del *Grid World* y el *Tank Control Task* [Geibel and Wysotzki, 2005].

2.4.2. Aproximaciones basadas en la Identificación de Estados de Error

Por otro lado están las técnicas que entienden el riesgo de una forma completamente diferente. Tratan de identificar estados, transiciones o acciones considerados peligrosos [Geibel, 2001; Geibel and Wysotzki, 2005; Hans, Alexander *et al.*, 2008], para no llegar nunca a ellos y así minimizar el nivel de riesgo. Geibel et al. [Geibel and Wysotzki, 2005] definen una función de riesgo como la probabilidad de visitar un estado de error (*error state*).

DEFINICIÓN 2.8 ***Estado de error*** *[Geibel and Wysotzki, 2005]. Sea el conjunto $\Phi \subseteq S$ un conjunto de estados de error. Un estado $s \in \Phi$ es un estado terminal, es decir, el control del agente termina cuando alcanza s.*

DEFINICIÓN 2.9 ***Estado de no error*** *[Geibel and Wysotzki, 2005]. Sea el conjunto $\Gamma \subseteq S$ un conjunto de estados de no error. Un estado $s \in \Gamma$ es considerado un estado terminal de no error, con $\Gamma \cap \Phi = \emptyset$.*

En términos de aprendizaje por refuerzo episódico, si el agente alcanza un estado de error, el episodio actual termina con daños en el agente o en el sistema de aprendizaje (e.g. el helicóptero se estrella, el robot ha golpeado una pared). Por el contrario, si el agente alcanza un estado de no error, el episodio se completa normalmente independientemente del refuerzo recibido sin que el agente sufra ningún daño.

DEFINICIÓN 2.10 ***Riesgo*** *[Geibel and Wysotzki, 2005]. El riesgo de un estado s con respecto a la política π, $\rho^\pi(s)$, se define como la probabilidad de que la secuencia de estados $(s_i)_{i\geq 0}$ con $s_0 = s$, que es generada ejecutando la política π, termine en un estado de error $s' \in \Phi$.*

Por definición, $\rho^\pi(s) = 1$ si $s \in \Phi$. Si $s \in \Gamma$, entonces $\rho^\pi(s) = 0$ porque $\Phi \cap \Gamma = \emptyset$. Para estados $s \notin \Phi \cup \Gamma$, el riesgo depende de las acciones elegidas por la política π. A partir de esta función de riesgo, Geibel et al. [Geibel and Wysotzki, 2005] utilizan una nueva función de valor ponderando la función de riesgo y la función de valor como se muestra en la ecuación 6.7.

$$V_\xi^\pi(s) = \xi V^\pi(s) - \rho^\pi(s) \tag{2.65}$$

El parámetro $\xi \geq 0$ determina la influencia de los valores $V^\pi(s)$ comparados con los de la función de riesgo $\rho^\pi(s)$. Para valores $\xi = 0$, V_ξ^π se corresponde a políticas de mínimo riesgo. Para valores elevados de ξ, la función de valor original domina a la función de riesgo.

Otra posibilidad para evitar estados peligrosos podría ser asociar refuerzos muy negativos a las transiciones que conduzcan a estos estados. Una política óptima entonces trataría

de evitar estos estados en general. No obstante, un gran inconveniente de esta aproximación es que se desconoce como de grande es el riesgo (probabilidad) de llegar a uno de estos estados. Por otra parte, es posible que se desee proporcionar un umbral de probabilidad w de entrar a un estado peligroso que no debe ser superado por la política del agente. No obstante, en general es imposible evitar por completo los estados de riesgo, pero deben ser controlados en cierta medida [Geibel and Wysotzki, 2005]. La definición de estados peligrosos varía de un trabajo a otro. Por ejemplo, Hans et al. [Hans, Alexander *et al.*, 2008] consideran un estado s peligroso si no existe una política π que, partiendo desde s, no realice una transición peligrosa, donde una transición (s, a, s', r) se considera peligrosa si el refuerzo r recibido cae por debajo de cierto umbral τ. Por la tanto, el buen funcionamiento de este algoritmo radica en una buena elección del parámetro τ, para lo cual, nuevamente, se requerirá cierto conocimiento del dominio.

2.4.3. Aproximaciones basadas en la Utilización de Expertos

Este tipo de aproximaciones se basan en el uso de expertos (*teachers*) capaces de hacer demostraciones de la tarea que se trata de resolver [Atkeson and Schaal, 1997]. Normalmente la utilización de estos expertos tiene como principal objetivo acelerar el proceso de aprendizaje reduciendo el tiempo dedicado a la exploración del entorno, aunque esto conlleva, a su vez, una reducción en las situación de riesgo que se producen. Estos expertos son utilizados de tres maneras diferentes: para inicializar el algoritmo de aprendizaje, para derivar una política a partir de un conjunto de demostraciones ejecutadas por el experto, para guiar el proceso de exploración.

(i) Inicialización del algoritmo de aprendizaje. Driessens y Sžeroski emplean un método de inicialización en aprendizaje por refuerzo relacional en el cual en primer lugar se almacenan un conjunto de demostraciones llevadas a cabo por un experto para posteriormente presentarlas a un algoritmo de regresión [Driessens and Džeroski, 2004]. Esto permite al algoritmo de regresión construir de manera parcial la función de valor, la cual puede utilizarse posteriormente para explorar el espacio empleando un estrategia de exploración Boltzmann. Smart y Kaelbling también utilizan ejemplos para inicializar una aproximación basada en Q-learning en su algoritmo HEDGER [Smart and Kaelbling, 2000]. El conocimiento inicial suministrado al algoritmo permite al agente aprender más efectivamente ayudando a reducir el tiempo dedicado en la exploración aleatoria del entorno. Los comportamientos expertos también se emplean para inicializar la población inicial en aproximaciones neuro-evolutivas [Yao, 1999; Siebel and Sommer, 2007] como la mostrada en el algoritmo 2.9). De esta forma, ya en la población inicial existen individuos capaces de completar la tarea con éxito, y el resto del proceso de aprendizaje trata de mejorar el comportamiento de estos individuos mediante cruzamientos y mutaciones. Este proceso de introducir expertos en la población inicial se puede considerar una forma de inicialización de la población (*population seeding*), donde no todos los pesos de los individuos son inicializados de forma aleatoria (y por tanto tienen un comportamiento aleatorio), sino que algunos tienen un comportamiento definido. Usando esta técnica, Martín et al. [Martín and de Lope Asiaín, 2009] (ganador de la competición de aprendizaje por refuerzo en su edición de 2009 en el dominio del control de un helicóptero), desarrolló el algoritmo mostrado en 2.9, en el cual varios expertos son introducidos en la población inicial. El algoritmo propuesto restringe los cruzamientos y las mutaciones permitiendo solo cambios muy pequeños a las políticas producidas por los expertos. De esta forma, se asegura la rápida convergencia del algoritmo a una política cercana a la política óptima [Lin, 1991] e, indirectamente, el algoritmo minimiza el número de colisiones del he-

licóptero realizando una exploración más segura del entorno [Smart and Kaelbling, 2002; Thomaz and Breazeal, 2006]. No obstante, los expertos introducidos en la población inicial son el resultado de un entrenamiento *ad-hoc*, y el rendimiento de este tipo de algoritmos dependen en gran medida del tipo de comportamiento de los expertos introducidos en la población inicial. Por eso, este tipo de aproximaciones parecen *ad-hoc* y difícilmente se podrían generalizar a otro tipo de dominios de aprendizaje por refuerzo sin volver a *tunear* los parámetros requeridos por el algoritmo, y la población inicial con los comportamientos expertos.

Koppejan et al. [Koppejan and Whiteson, 2009; Koppejan and Whiteson, 2011] utiliza una técnica similar a la comentada anteriormente, evolucionando redes de neuronas y comenzando a partir de redes de neuronas que tienen comportamientos *expertos*. Este tipo de inicialización se ha mostrado ventajosa en numerosas aplicaciones de métodos evolutivos [Hernández-Díaz *et al.*, 2008; Koppejan and Whiteson, 2009]. No obstante, el algoritmo que propone Koppejan parece en unas ocasiones *ad-hoc* y diseñado para un conjunto específico de entornos. Koppejan hace una definición de estado peligroso para el dominio del helicóptero que requiere un fuerte conocimiento del dominio y su dinámica, que es imposible generalizar a otros entornos. Concretamente considera que un estado es peligroso si la posición del helicóptero, la velocidad, o la velocidad angular a lo largo de uno de los ejes de coordenadas cae fuera del rango [-0.5,0.5]. Además, su método evoluciona perceptrones multi-capa con una topología construida manualmente por humanos expertos en la tarea en cuestión [Ng *et al.*, 2004]. Esta topología calcula el valor de cada dimensión de la acción individualmente, a partir de un subconjunto de características del estado. Con este diseño, se reduce considerablemente la complejidad del problema. No obstante, esta división requiere, de nuevo, de un fuerte conocimiento del dominio y su dinámica, y puede que en otros dominios esta división no sea posible, quedando restringida la aplicación del método a un conjunto muy limitado de problemas. Esta idea se ver reforzada con el hecho de que Koppejan solo aporta resultados de su algoritmo en el dominio del helicóptero.

(ii) Aprendizaje de una política a partir de un conjunto finito de demostraciones. Todas estas aproximaciones caen dentro del campo de Aprendizaje por Demostración [Argall *et al.*, 2009]. En este grupo, destaca el trabajo de Abeel et al., basado en el *Aprendizaje del Aprendiz* (*Apprenticeship Learning*) [Abbeel *et al.*, 2010]. En este tipo de aproximaciones, el objetivo es que el agente (*aprendiz*) aprenda una política para el MDP que se trata de resolver, a partir de las demostraciones realizadas por un experto en ese mismo MDP, y que al menos sea tan buena como la política del experto [Abbeel *et al.*, 2007; Abbeel *et al.*, 2008; Abbeel *et al.*, 2010]. Abbel et al. proponen un algoritmo compuesto de tres pasos diferentes: i) seleccionar un experto para demostrar la tarea que se trata de resolver, y guardar las trayectorias estado-acción generadas por las demostraciones llevadas a cabo por el experto en esa tarea, ii) A partir de las trayectorias estado-acción generadas en el paso anterior por el experto, aprender el modelo de la dinámica del sistema, y, para este modelo, encontrar la política cercana al óptimo empleando cualquier algoritmo de aprendizaje por refuerzo, y iii) Probar la política obtenida ejecutándola en el sistema real. En su lugar, Tang et al. [Tang *et al.*, 2010] presentan un algoritmo para la generación automática de trayectorias en complejas tareas de control a partir de las demostraciones de múltiples expertos.

(iii) Guía del proceso de exploración. Driessens and Sžeroski, de nuevo en el contexto de aprendizaje por refuerzo relacional, utilizan un comportamiento experto, en lugar de la política derivada de la aproximación actual de la función de valor-acción (la cuál no se ha aprendido en los primero pasos del aprendizaje), para la selección de acciones [Driessens and Džeroski, 2004]. En esta aproximación, los episodios ejecutados mediante el comportamien-

to experto son intercalados con episodios donde se lleva a cabo una exploración normal. La mezcla del comportamiento experto con exploración normal facilita al algoritmo de regresión distinguir entre buenas y malas acciones. En el contexto de aprendizaje por demostración, hay aproximaciones que utilizan los consejos de un comportamiento experto (*teacher advice*). En esta aproximación, en primer lugar el experto realiza demostraciones sobre la tarea que se trata de aprender. Posteriormente, el agente solicita demostraciones adicionales al experto en estados muy diferentes de los estados con demostración o en estados donde una acción individual no puede ser seleccionada con certeza [Chernova and Veloso, 2007; Chernova and Veloso, 2008]. Esta aproximación se basa en la asunción de que el experto está siempre disponible durante el proceso de aprendizaje, y que éste es suficiente para reconocer errores basados en la observación y para conocer qué demostraciones serían útiles para corregir el comportamiento que el agente está llevando a cabo. No obstante la utilización de esta técnicas sólo se ha demostrado en tareas simples con un número discreto de acciones.

2.4.4. Discusión

La exploración en aprendizaje por refuerzo es un tema clave a la hora de encontrar buenas políticas de comportamiento. En general, estrategias de exploración como $\epsilon-greedy$ o *Boltzmann* se consideran no seguras para dirigir la exploración en dominios considerados de riesgo, debido a la fuerte componente aleatoria que introducen. Estas estrategias podrían requerir la ejecución de políticas que exploran de forma agresiva diferentes partes del espacio de estados, incluyendo aquellas partes que son peligrosas y que pueden provocar daños en el agente.

En cuanto al primer tipo de aproximaciones basadas en la varianza del refuerzo acumulado o en el peor de los casos presentadas en la sección anterior, Geibel et al. [Geibel and Wysotzki, 2005] demuestran que para ciertos problemas, estas técnicas no sirven. El dominio que presentan para afirmar este hecho es el *grid-world* en el cual un agente que parte de una posición inicial debe alcanzar una posición final mediante la ejecución de cuatro acciones posibles: arriba, abajo, derecha, izquierda. Con cierta probabilidad el agente no se transporta a la posición deseada, sino a una de las tres direcciones restantes. Esto añade al dominio una componente estocástica. El agente recibe un refuerzo igual a 1 si consigue alcanzar la meta, y de 0 en cualquier otro caso. En el dominio previamente se han marcado unas posiciones que se identifican con estados de error, y que si el agente alcanza, el episodio se da por finalizado. En este problema, siguiendo el esquema del peor de los casos, **todas** las políticas tienen la misma peor salida posible (debido a la componente estocástica que hace que hasta una política que alcance la meta en un episodio, puede acabar en un estado de error en el siguiente). En cuanto a las aproximaciones basadas en la varianza del refuerzo acumulado, en este dominio, no existe mucha diferencia entre la varianza de las políticas que encuentran el estado meta rápidamente y las que encuentran un estado de error. Por lo tanto, este tipo de aproximaciones tampoco son aplicables a dominios donde una pequeña varianza puede implicar un gran riesgo (en cuanto a la probabilidad de visitar un estado de error). Además, la mayor parte de estas aproximaciones están limitadas a representaciones tabulares de la función de valor (i.e., en la mayoría de los casos solo consideran dominios con unos espacios de estados y acciones discretos y reducidos) [Heger, 1994]. En esta Tesis se aborda por tanto un problema diferente de control comparado con las aproximaciones basadas en la varianza del refuerzo acumulado o el peor de los casos.

En la siguiente línea de trabajo en aprendizaje por refuerzo con riesgo, se trata de

diferenciar entre estados de error y de no error. No obstante, estas aproximaciones también tienen algunos inconvenientes si lo que se quiere es minimizar el número de daños en el agente o en el sistema de aprendizaje (i.e., minimizar el número de veces que se visitan estados de error). En el caso de la aproximación de Geibel et al. [Geibel and Wysotzki, 2005], el aprendizaje de la función de riesgo ρ conlleva visitar repetidamente estados de error (e.g. que el helicóptero se estrelle, que el robot se caiga o colisione contra una pared). Sólo cuando la función de riesgo se haya aproximado se conseguirá, dependiendo del valor ξ seleccionado, evitar las situaciones de peligro. En la aproximación presentada por Hans et al. [Hans, Alexander *et al.*, 2008], consideran que una transición es fatal si el refuerzo obtenido en la misma cae por debajo de cierto umbral τ, y un estado peligroso si no existe una política que partiendo de este estado no realice una transición peligrosa. En este caso, la definición del parámetro τ requerirá de un fuerte conocimiento del dominio (en ciertos problemas un refuerzo por debajo de cierto umbral puede ser considerado admisible mientras que en otros no). Por otro lado, el concepto de riesgo en esta aproximación se vuelve a hacer dependiente del refuerzo. Además, tanto en el trabajo de Geibel como en el de Hans, se asume que las dinámicas son conocidas, se tolera visitar estados de error, o, a diferencia de la aproximación propuesta en esta Tesis, no afrontan problemas espacios de estados y acciones continuos y de grandes dimensiones. Con respecto a esto último, Geibel et al. [Geibel and Wysotzki, 2005] escriben que su aproximación puede ampliarse fácilmente para tratar con problemas con espacio de acciones continuos, utilizando por ejemplo arquitecturas actor-crítico, pero no dan más detalles sobre cómo llevar a cabo esta ampliación.

La última línea de investigación que se basa en la utilización de una u otra forma de las demostraciones llevadas a cabo por expertos, también presenta algunos inconvenientes. La inicialización de las aproximaciones utilizadas en la aproximación de las funciones de valor no evita que durante el posterior proceso de exploración se visiten estados de error [Driessens and Džeroski, 2004; Smart and Kaelbling, 2000]. En el caso de las aproximaciones basadas en aprendizaje por refuerzo evolutivo donde se añaden comportamientos expertos en la población inicial [Martín and de Lope Asiaín, 2009; Koppejan and Whiteson, 2009; Koppejan and Whiteson, 2011], además de presentar los problemas inherentes a la utilización de redes de neuronas, presentan otro tipo de problemas. En primer lugar, que la solución que presentan parece en muchos casos modelada *ad-hoc* que la hace difícilmente generalizable a otros dominios sin un fuerte *tuneado* de los parámetros del algoritmo y de la obtención de la población inicial con expertos. Un ejemplo de este hecho se demuestra en que Koppejan et al. [Koppejan and Whiteson, 2009; Koppejan and Whiteson, 2011] tratan de distinguir los estados peligrosos en el dominio del helicóptero utilizando un fuerte conocimiento acerca de la dinámica del sistema. Concretamente consideran un estado peligroso si la posición del helicóptero, la velocidad, o la velocidad angular a lo largo de uno de los ejes de coordenadas cae fuera del rango [-0.5,0.5]. Para reforzar aún más este punto, y como se comentó anteriormente, la topología de redes de neuronas están definidas por un experto en el dominio. Mediante esta topología además, se consigue reducir la complejidad del problema puesto que se decide el valor de cada dimensión de la acción individualmente considerando sólo un subconjunto de las variables de estado. Por otro lado, si se considera complicado obtener un sólo comportamiento experto capaz de realizar demostraciones de la tarea que se trata de aprender, lo es aún más conseguir un conjunto de estos comportamientos expertos que aporten la suficiente variedad a la población inicial. Además, solo se presentan resultados de esta técnica en el dominio del helicóptero de la competición de aprendizaje por refuerzo. En el caso de las aproximaciones basadas en el *Aprendizaje del Aprendiz* (*Apprenticeship Learning*) [Abbeel *et al.*, 2007; Abbeel *et al.*, 2008; Abbeel *et al.*, 2010;

Tang *et al.*, 2010], el modelo aprendido de la dinámica del entorno está fuertemente condicionado por el comportamiento del experto utilizado, pudiendo obtenerse resultados muy diferentes si se utiliza un experto u otro. Por último, la intercalación de episodios guiados mediante el comportamiento experto con episodios de exploración normales, no evita que durante estos episodios de exploración normales se visiten estados de error [Driessens and Džeroski, 2004]. En cualquier caso, en todas las aproximaciones descritas en esta línea de investigación no se hace mención explícita del concepto de riesgo, ni de cómo tratar de evitar situaciones peligrosas que conduzcan a daños en el agente y en el sistema de aprendizaje.

Además, existe una total desconexión entre los métodos de generalización y la exploración segura del entorno en dominios realistas y complejos. Los trabajos de aprendizaje por refuerzo con riesgo utilizan como dominios de experimentación dominios simples y en la mayor parte de las ocasiones con unos espacios de estados y acciones discretos.

Parte II

Objetivos y Evaluación de la Tesis Doctoral

Capítulo 3

Objetivos de la Tesis Doctoral

En este capítulo se describen detalladamente los objetivos que se pretenden alcanzar con la realización de esta Tesis. Estos objetivos pretenden cubrir algunas de las deficiencias descubiertas durante la revisión exhaustiva de la literatura existente, y que se detallan a modo de resumen en la sección siguiente.

3.1. Motivación de la Tesis Doctoral

En cuanto a las técnicas de aprendizaje por refuerzo para su aplicación en dominios totalmente continuos y de grandes dimensiones, las deficiencias encontradas en la literatura actual, y que se discutían en la sección 2.3.4, se resumen a continuación.

1. La mayor parte de los trabajos se concentran exclusivamente en la generalización del espacio de estados [Fernández and Borrajo, 2002; Goto *et al.*, 2002; Stone *et al.*, 2005; García *et al.*, 2007] considerando dominios con un conjunto de acciones discretas y reducido. Son menos los trabajos que afrontan el problema de la generalización del espacio de acciones [Lazaric *et al.*, 2007; Mansley *et al.*, 2011], y son aún menos los trabajos que abordan de forma conjunta el problema de la generalización de los espacios de estados y acciones.

2. La mayor parte de los trabajos donde se abordan problemas con espacios de estados y acciones continuos utilizan aproximación de funciones para aproximar el valor de la función de valor-estado (o la función de valor-acción). Como función de aproximación es mayoritario el uso de redes de neuronas [Prokhorov and Wunsch, 1997; van Hasselt and Wiering, 2007], aunque también existen trabajos que utilizan técnicas CBR para aproximar funciones [Santamaría *et al.*, 1998; Smart and Kaelbling, 2000; Gabel and Riedmiller, 2005; Molineaux *et al.*, 2008], o técnicas basadas en las máquinas de vectores de soporte (*support vector machines*) [Xu *et al.*, 2007; Röttger and Liehr, 2009; Xu *et al.*, 2011]. **Estas técnicas conllevan el problema de extraer conocimiento sobre cuál es la acción óptima a ejecutar en cada momento.**

3. En el caso de las propuestas del punto anterior que utilizan redes de neuronas, se suele utilizar otro aproximador, también una red de neuronas, que representa la política de comportamiento [Prokhorov and Wunsch, 1997; van Hasselt and Wiering, 2007]. El uso de redes de neuronas presenta problemas de convergencia [Justin Boyan, 1995] (que se ve agravado por el hecho de que se tratan de aproximar dos funciones en lugar de una), y conlleva además, problemas para seleccionar la topología adecuada.

4. En el caso de las propuestas anteriores que utilizan CBR como técnica de aproximación de funciones, se suele hacer un barrido sobre un conjunto finito de acciones calculando su valor Q, con el fin de determinar el mejor valor, y por tanto la mejor acción (algoritmo *One Step Search* 2.5). No obstante, un barrido sobre un número elevado de acciones conllevará un alto coste computacional. En otras ocasiones se suele aproximar la función de valor-estado [Molineaux *et al.*, 2008] o la función de valor-acción [Smart and Kaelbling, 2000] para n acciones diferentes mediante técnicas de regresión local, con el fin de buscar el valor máximo de la función. Realizar esta aproximación requiere en algunos casos almacenar una base de casos adicional [Molineaux *et al.*, 2008] aumentando el tiempo de acceso y los requisitos de almacenamiento, y en otros casos requiere el muestreo de n acciones diferentes, de cuya selección depende el posterior rendimiento del algoritmo [Smart and Kaelbling, 2000].

5. Por último, en el caso de utilizar técnicas de máquinas de vectores de soporte como técnica de aproximación de funciones, la acción óptima se calcula en cada momento basándose en la información que proporcionan los gradientes de la función de valor-acción aproximada, un cálculo que requiere un mayor coste computacional a medida que crece el número de dimensiones de las acciones [Xu *et al.*, 2011]. En otros casos, se discretiza de forma uniforme el espacio de acciones con el fin de evaluar cada acción de forma individual siguiendo una estrategia similar a la presentada en el algoritmo *One Step Search* (tabla 2.5) [Röttger and Liehr, 2009].

6. La mayoría de los dominios empleados en la experimentación son dominios con unos espacios de estados y acciones continuo aunque de reducido número de dimensiones [Munos, 2000; van Hasselt and Wiering, 2007; Pazis and Lagoudakis, 2009; Koppejan and Whiteson, 2009; Mansley *et al.*, 2011; Xu *et al.*, 2011]. En algunos casos, además, se hace una descomposición de los espacios de estados y acciones basada en el conocimiento de expertos [Koppejan and Whiteson, 2009], reduciendo así la complejidad del problema.

7. Hay escasez de trabajos que aborden el problema de la generalización conjunta de los espacios de estados y acciones mediante técnicas de discretización, o mediante la combinación de técnicas de discretización y aproximación de funciones [Millán *et al.*, 2002].

Además, las deficiencias encontradas en la literatura sobre el aprendizaje por refuerzo con riesgo, y que se detallaban en la sección 2.4.4, quedan resumidas en los siguientes puntos.

1. En algunos trabajos el concepto de riesgo está asociado a la varianza del refuerzo acumulado [Heger, 1994; Mihatsch and Neuneier, 2002]. No obstante, los algoritmos basados en este concepto no son aplicables a dominios donde las políticas de riesgo tienen muy poca varianza [Geibel and Wysotzki, 2005]. Además, este tipo de aproximaciones afrontan un tipo de problema diferente al abordado en esta Tesis.

2. En otros casos, se distinguen estados de *error*, como los estados indeseados que pueden provocar daños en el agente o en el sistema de aprendizaje [Geibel, 2001; Geibel and Wysotzki, 2005]. En este caso, el aprendizaje de la función de riesgo conlleva visitar repetidamente estados de error hasta que se aproxima correctamente. En otros casos se da una definición de estados de riesgo en función de estados/transiciones peligrosas asociados con un refuerzo que cae por debajo de cierto umbral [Hans, Alexander *et*

al., 2008]. El problema de esta aproximación es que se requiere la correcta elección del umbral para identificar adecuadamente los estados de riesgo, y normalmente con esto no se consigue evitar por completo visitarlos. Además, en esta aproximación quedan íntimamente ligados el concepto de riesgo y refuerzo.

3. En el caso de aprendizaje por refuerzo seguro basado en la utilización de *expertos*, las aproximaciones basadas en aprendizaje por refuerzo evolutivo [Martín and de Lope Asiaín, 2009; Koppejan and Whiteson, 2009; Koppejan and Whiteson, 2011] parecen en la mayoría de los casos diseñadas *ad-hoc* y difícilmente generalizables a otros dominios. En las técnicas basadas en el *Aprendizaje del Aprendiz*, el modelo de la dinámica del entorno queda vinculado al comportamiento del experto, que en general, no realizará una exploración en profundidad del entorno. Además, en ninguno de los trabajos anteriores se ofrece explícitamente una definición del concepto de riesgo, y la reducción de los daños en el agente son una consecuencia de la utilización de estos *expertos* más que el objetivo [Abbeel and Ng, 2005; Martín and de Lope Asiaín, 2009]. El objetivo en la mayor parte de los casos es aumentar la velocidad de convergencia de los algoritmos de aprendizaje [Lin, 1991; Martín and de Lope Asiaín, 2009; Koppejan and Whiteson, 2009; Koppejan and Whiteson, 2011].

4. En la mayoría de los trabajos se abordan problemas discretos [Heger, 1994], o continuos pero con dimensionalidad reducida [Geibel and Wysotzki, 2005; Hans, Alexander *et al.*, 2008]. En otros casos, previa a la aplicación del algoritmo, se realiza una reducción de la complejidad del problema sugerida por un experto [Koppejan and Whiteson, 2009; Koppejan and Whiteson, 2011].

5. Desconexión entre los métodos de generalización y los algoritmos de aprendizaje por refuerzo con riesgo en dominios realistas y complejos.

Todos estos motivos dan lugar a los objetivos que se presentan a continuación.

3.2. Objetivos

El primer gran objetivo que se plantea en el desarrollo de esta Tesis doctoral consiste en el **obtención de métodos de aprendizaje por refuerzo libres del modelo aplicables a dominios con espacios de estados y acciones continuos y de una dimensión mayor que la habitualmente encontrada en la literatura, utilizando para ello técnicas de discretización basadas en el método del vecino más cercano** [Cover and Hart, 1967; Macleod *et al.*, 1987; Fernández and Borrajo, 1999; García *et al.*, 2007] y **aproximación de funciones** [Stone *et al.*, 2005]. La consecución de este objetivo plantea:

1. La eliminación de los dos aproximadores que normalmente aparecen en la literatura en las arquitecturas del tipo actor-crítico (Algoritmo 2.6) y que dificultan la convergencia de los algoritmos, considerando un único aproximador de la función Q a partir del cual se pueda extraer de forma directa la política de comportamiento óptima o una cercana a la óptima.

2. Los métodos obtenidos emplearán de forma combinada técnicas de discretización y aproximación de funciones para aprovechar los beneficios que ofrecen ambas técnicas, así como métodos que empleen únicamente técnicas de discretización para generalizar tanto el espacio de estados como el de acciones.

3. La discretización empleada permitirá reducir el enorme número de parámetros que hay que ajustar adecuadamente para aprender una política de comportamiento óptima o cercana a la óptima, y que habitualmente necesitan las técnicas que abordan directamente los problemas continuos de grandes dimensiones sin ningún tipo de simplificación o discretización.

4. Los métodos obtenidos deben poder ser aplicables con independencia del dominio en que se trate de realizar el aprendizaje (i.e., el algoritmo no tendrá parámetros que requerirán para su elección de cierto conocimiento del dominio).

5. Los métodos emplearán, cuando el dominio lo requiera, **comportamientos base o expertos** definidos a priori para realizar una exploración adecuada del espacio de estados y del espacio de acciones. De esta forma, se expone al sistema de aprendizaje a las zonas más relevantes de este espacio sobre el que después llevar a cabo el proceso de aprendizaje.

6. Los métodos obtenidos deben contemplar, en la medida de lo posible, las siguientes capacidades esenciales para un algoritmo de aprendizaje por refuerzo [Gaskett *et al.*, 1999].

   - ***Selección de la Acción:*** Encontrar la acción con el mayor valor esperado rápidamente.
   - ***Evaluación de un Estado:*** Encontrar el valor de un estado rápidamente para actualizar el valor de la función Q.
   - ***Evaluación de la función Q:*** Almacenar o aproximar la función Q completa, como se requiere para realizar las actualizaciones.
   - ***Libre del modelo:*** No requiere conocer o aprender previamente el modelo de las dinámicas del sistema.
   - ***Política Flexible:*** Permite la representación de una amplia gama de políticas.
   - ***Continuidad:*** Las acciones pueden cambiar ligeramente con pequeños cambios en los estados.
   - ***Generalización de Estados:*** Generaliza entre estados similares, reduciendo la cantidad de exploración requerida para el espacio de estados.
   - ***Generalización de Acciones:*** Generaliza entre acciones similares, reduciendo la cantidad de exploración requerida para el espacio de acciones.

7. Los métodos propuestos se probarán en diferentes dominios **markovianos, estocásticos, grandes, episódicos y generalizados** (sección 2.1). Se considera un dominio **grande** cuando los espacios de estados y acciones son continuos y además el número de variables que describen los estados y acciones es mayor que el encontrado en la literatura.

8. Comparación de los métodos obtenidos con algunos algoritmos existentes, donde se espera alcanzar un rendimiento superior, o en el peor de los casos similar, en los diferentes dominios propuestos.

El segundo gran objetivo que se plantea consiste en la **obtención de métodos de aprendizaje por refuerzo libres del modelo aplicables a dominios con riesgo con**

unos espacios de estados y acciones continuos y de una dimensión mayor que la habitualmente encontrada en la literatura, minimizando el riesgo de daño en el agente o en el sistema de aprendizaje. Este objetivo plantea:

1. Definir una nueva función que permita identificar cuándo un estado puede ser considerado seguro, y por tanto se podrían utilizar estrategias de exploración agresivas, y cuándo es un estado peligroso, en el cual se deberán emplear comportamientos conservadores/seguros, que permitan regresar el sistema a una situación segura, minimizando así los daños que se pueden producir en el agente o en el sistema de aprendizaje.

2. Esta función de riesgo no estará basada ni en la varianza del refuerzo acumulado, ni en la definición de estados/transiciones de error, sino en la **distancia** existente entre el espacio de estados conocido por el agente (que almacena de alguna forma en memoria), y el espacio desconocido. De esta forma, no se requiere aproximar primero ninguna función de riesgo [Geibel and Wysotzki, 2005], y ya desde el comienzo del aprendizaje es posible identificar una posible situación de riesgo de la que no lo es, de forma que se pueda actuar de una u otra forma.

3. Los métodos propuestos emplearán **comportamientos base o expertos** definidos a priori que servirán en primer lugar para indicar al agente cómo debe comportarse inicialmente en el entorno, y en segundo lugar para apoyar el proceso de exploración posterior dónde se emplearán para regresar el sistema a una situación segura cuando se visiten situaciones consideradas peligrosas.

4. Para este parámetro de **distancia** y la de otros involucrados en el algoritmo, se dará una guía sobre cómo inicializar sus valores, de forma que no sea necesario realizar un proceso de refinamiento de todos los parámetros para conseguir unos resultados adecuados.

5. Como en el caso anterior, los métodos propuestos se probarán en diferentes dominios **markovianos, estocásticos, grandes, episódicos y generalizados** (Sección 2.1). Además, en estos dominios existirá una componente de riesgo que se traducirá en daños en el agente o en el sistema de aprendizaje (e.g., que el helicóptero se estrelle), o en situaciones indeseables que hay que tratar de evitar (e.g., que una compañía entre en bancarrota).

6. El modelo de aprendizaje propuesto se comparará con otros algoritmos que han demostrado ser competitivos, debiendo alcanzar un resultado superior, o en el peor de los casos, similar en cuanto a rendimiento se refiere, y obteniendo un menor número de daños en el agente (e.g. colisiones, choques).

Capítulo 4

Evaluación de la Tesis Doctoral

La evaluación de los algoritmos propuestos en esta Tesis se llevará a cabo comparando su rendimiento con otros algoritmos existentes en la literatura, y utilizando diferentes dominios de experimentación. En las siguientes secciones se detallan los algoritmos concretos utilizados en las comparaciones para cada uno de los objetivos planteados, y los dominios de evaluación empleados.

4.1. Algoritmos de Evaluación

En la tabla 4.1 se muestran los algoritmos que se utilizarán en las comparaciones para cada uno de los objetivos definidos en la Tesis.

	Objetivo 1	Objetivo 2
CMAC continuo [Santamaría *et al.*, 1998] (Algoritmo 2.4)	✔	
CACLA [van Hasselt and Wiering, 2007] (Algoritmo 2.6)	✔	
Aprendizaje por refuerzo evolutivo [Martín and de Lope Asiaín, 2009] (Algoritmo 2.9)	✔[1]	✔[2]
Aproximación estados de error [Geibel and Wysotzki, 2005] (Sección 2.4.2)		✔[3]

[1] Todos los comportamientos en la población inicial se generan de forma aleatoria

[2] Se introducen en la población inicial varios comportamientos base o *expertos*

[2] Se ha modificado para su aplicación en dominios con espacios de estados y acciones continuos

Tabla 4.1: Utilización de los algoritmos en cada uno de los objetivos planteados en la Tesis.

El *Objetivo 1* se refiere al objetivo de desarrollar algoritmos de aprendizaje por refuerzo aplicables a dominios con espacios de estados y acciones continuos, utilizando técnicas de discretización y aproximación de funciones mientras que el *Objetivo 2* se refiere a desarrollar algoritmos de aprendizaje por refuerzo aplicables a dominios con riesgo con unos espacios de estados y acciones continuos. En el *Objetivo 1* no se introducen comportamientos base en la población inicial del algoritmo evolutivo. Esto es así por varios motivos fundamentales. En primer lugar, se trata de demostrar las ventajas que aportan la utilización de comportamientos base, empleando para ello los algoritmos propuestos en el *Objetivo 1*, frente a algoritmos que no los utilizan. En segundo lugar, en el dominio del *Octopus Arm*, dada su complejidad, no se ha conseguido modelar ningún comportamiento base mediante redes de neuronas. Koppejan et al. [Koppejan and Whiteson, 2009; Koppejan and Whiteson, 2011] y Martín et al. [Martín and de Lope Asiaín, 2009] introducen con éxito comportamientos base en la población inicial, aunque sólo en el dominio

del *Helicopter* donde el comportamiento base está descrito previamente por un conjunto de ecuaciones lineales fácilmente modelables mediante una red de neuronas. Puesto que el *Octopus Arm* tiene 82 características para los estados y 32 para las acciones, las redes de neuronas empleadas en este modelado constan de 82 neuronas de entrada y 32 neuronas de salida, lo que dificulta cualquier tipo de convergencia [1]. Por tanto, no siempre es posible la introducción de este tipo de conocimiento en la población inicial, circunstancia que se ve agravada en problemas de gran tamaño donde la complejidad de las redes de neuronas complica la convergencia de la red. Por último, los algoritmos propuestos en el *Objetivo 1*, no tratan de construir conocimiento para lograr imitar el comportamiento base, sino que simplemente recogen información sobre las regiones del espacio de estados y acciones visitadas por el comportamiento base. La información recogida se utiliza para construir discretizaciones de estas regiones "interesantes" del espacio, que posteriormente son utilizadas para mejorar el rendimiento de los algoritmos. Por lo tanto, la utilización del comportamiento base por parte de los algoritmos en el *Objetivo 1*, es mucho más generalizable a otros dominios que la utilización que hacen algunos algoritmos evolutivos. No obstante, dada la naturaleza del algoritmo propuesto en el *Objetivo 2*, que en primer lugar trata de imitar el comportamiento base, y posteriormente hace uso de éste para apoyar el posterior proceso de exploración del espacio para mejorar su rendimiento (i.e., utiliza de una forma mucho más intensiva el comportamiento base), si se ve necesario que los algoritmos evolutivos con los que se compara tengan en la población inicial varias redes de neuronas modelando comportamientos base.

4.2. Dominios de Evaluación

Varios de estos dominios están extraídos de la competición de aprendizaje por refuerzo (*Helicopter domain* y *Octopus Arm*). Esta competición, que se celebra cada cierto tiempo, permite a los investigadores en aprendizaje por refuerzo comparar rigurosamente sus resultados en una amplia variedad de dominios. Estos dominios se han seleccionado de forma que crezcan en complejidad.

4.2.1. *Automatic Car Parking Problem*

El problema del aparcamiento automático se representa en la figura 4.1. Ha sido utilizado en la literatura de aprendizaje por refuerzo como dominio de experimentación [Cichosz, 1996]. En este problema, un coche (que se representa mediante un rectángulo en la figura 4.1) se encuentra inicialmente en una determinada posición dentro del área de conducción (limitada por la línea solida en la figura 4.1). El objetivo del agente es conducir el coche desde la posición inicial hasta el garaje en la mínima cantidad de pasos posible, de forma que el coche quede completamente dentro del garaje. El coche no se puede mover fuera del área de conducción. Se trata de una tarea episódica donde cada episodio comienza con el coche en una misma posición dentro del área de conducción. Cada episodio finaliza cuando

[1] El entrenamiento se ha realizado utilizando en primer lugar el comportamientos base para generar un conjunto de patrones de *entrada-salida*. Posteriormente, se emplea el método de descenso de gradiente *backpropagation* para entrenar la red. Este método primero presenta un patrón de entrada que se propaga por las distintas capas que componen la red hasta obtener la salida. Esta salida obtenida se compara con la salida deseada y se calcula el error cometido por cada neurona de salida. Estos errores se transmiten hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de las capas intermedias. Basándose en el error recibido, se ajustan los errores de los pesos sinápticos de cada neurona [Werbos, 1994].

el agente aparca el coche en el garaje, o cuando golpea el coche contra una pared o contra el obstáculo definido.

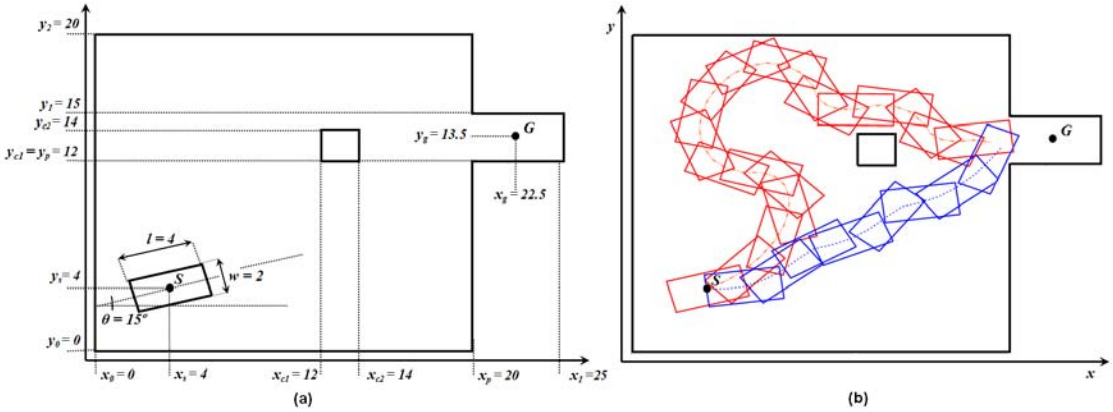


Figura 4.1: Problema de aparcamiento automático: (a) Modelo del problema del aparcamiento automático. (b) Ejemplos de trayectorias seguidas por el agente para tratar de aparcar el coche.

Los estados en este dominio se describen mediante tres variables continuas: las coordenadas del centro del coche, x_t y y_t, y el ángulo θ entre el eje del coche y el eje X del sistema de coordenadas. El coche se dirige con dos entradas también continuas: la velocidad, v, y el ángulo de dirección, ϕ. No obstante, el coche se maneja sólo mediante el ángulo de dirección ϕ puesto que la velocidad v permanece constante. De esta forma, el espacio de acciones se describe mediante una variable continua $a_t \in [-1, 1]$, correspondiente al radio de giro que se utilizará en las ecuaciones con la dinámica del sistema, descritas más adelante. Cuando el coche se encuentra totalmente dentro del garaje, el agente recibe un refuerzo positivo $r = (1 - \varsigma(dist(P_t, P_g))) \times 10$, donde $P_t = (x_t, y_t)$ es el centro del coche, $P_g = (x_g, y_g)$ es el centro del garaje (y la posición meta), y ς es una función de normalización que escala la distancia euclídea $dist(P_t, P_g)$ entre P_t y P_g dentro de un rango $[0, 1]$ cuando el coche se encuentra dentro del garage (i.e., el valor del refuerzo es mayor cuando el coche está aparcado correctamente en el centro del garaje). El agente recibe un refuerzo de -1 siempre que el coche choque contra la pared o el obstáculo. En todos los pasos restantes, el refuerzo es de -0.1. Se trata de un problema difícil no sólo por el refuerzo retardado, sino además, porque los refuerzos negativos son mucho más frecuentes que los positivos (i.e., es mucho más fácil golpear la pared que aparcar el coche correctamente en el garaje). El movimiento del coche queda descrito mediante las siguientes ecuaciones [Tanaka and Sano, 1995; Lee and Lee, 2008]:

$$\theta_{t+1} = \theta_t + v\tau/(l/2)\tan(\phi \times a_t) \tag{4.1}$$
$$x_{t+1} = x_t + v\tau\cos(\theta_{t+1}) \tag{4.2}$$
$$y_{t+1} = y_t + v\tau\sin(\theta_{t+1}) \tag{4.3}$$

donde v es la velocidad lineal del coche que se asume que es constante, ϕ es el máximo ángulo de giro (i.e., el coche puede cambiar su dirección en un ángulo máximo ϕ en ambas direcciones), y τ es la duración de un paso de la simulación. Para añadir una componente estocástica al dominio, se ha añadido ruido tanto a las acciones como a los refuerzos mediante

una gaussiana con una desviación estándar de 0.1. En esta Tesis, $l = 4$ (m), $v = 1{,}0$ (m/s), $\phi = 0{,}78$ (rad), and $\tau = 0{,}5$ (s) (las dimensiones del área de conducción y el obstáculo se detallan en la figura 4.1 (a)). Para transformar el dominio en estocástico y puesto que las interacciones con ruido son inevitables en la mayor parte de las aplicaciones del mundo real, se ha añadido ruido aleatorio a las acciones y a la función de refuerzo siguiendo una distribución Gaussiana con desviación 0.1.

4.2.2. *Cart-Pole*

Se trata del problema del péndulo invertido, en el cual se tiene un carro que puede moverse uni-dimensionalmente por un camino [Barto *et al.*, 1983]. Sobre el carro se tienen un elemento lineal (el péndulo) que puede girar solamente en el plano vertical del carro y el camino como se indica en la figura 4.2.

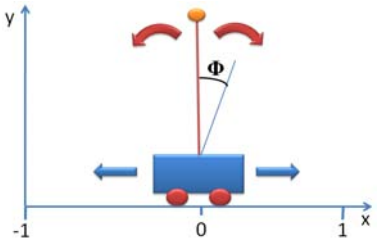


Figura 4.2: Cart-Pole.

El problema radica en encontrar la fuerza horizontal a aplicar en el carro de manera que se consiga que el carro esté estacionado en el centro de la pista y el péndulo balanceado en la posición vertical. El espacio de estados consta de cuatro características que se corresponden con el ángulo ϕ y velocidad radial ϕ' del péndulo, y la posición x y la velocidad x' del carro. El espacio de acciones consta de una única dimensión continua correspondiente a la fuerza, f, que es aplicada para empujar el carro en una dirección u otra. Una fuerza positiva implica mover el carro a la derecha, mientras que si es negativa el carro se mueve a la izquierda. La dinámica del péndulo y el carro quedan descritas por las ecuaciones:

$$x_{t+1} = x_t + \tau x'_t \tag{4.4}$$

$$x'_{t+1} = x'_t + \tau \left(\left(\frac{f + M_p L \phi_t^2 \sin(\phi_t)}{M} \right) - \left(\frac{M_p L \cos(\phi_t)}{M} \right) \right) \tag{4.5}$$

$$\phi_{t+1} = \phi_t + \tau \phi'_t \tag{4.6}$$

$$\phi'_{t+1} = \phi'_t + \tau \left(\frac{G \sin(\phi) - \cos(\phi) \left(\frac{f + M_p L \phi_t^2 \sin(\phi_t)}{M} \right)}{L \left(\frac{\frac{4{,}0}{3{,}0} - M_p \cos(\phi)^2}{M} \right)} \right) \tag{4.7}$$

donde M_p es el peso del péndulo, M es el peso total del péndulo y el carro, G es la fuerza de gravedad, L es la mitad de la longitud del péndulo, y τ es la duración de un paso de la simulación. En los experimentos llevados a cabo en esta Tesis, $M_p = 1{,}0$ (Kg), $M = 1{,}1$ (Kg), $G = 9{,}8$ (m/s), $L = 0{,}5$ (m), y $\tau = 0{,}02$ (s). Un episodio en este dominio finaliza cuando el péndulo supera cierto ángulo $\phi_{\text{máx}} = 0{,}52(rad)$ o el carro se sale fuera

de los límites de la pista de longitud $l = 4{,}8$. La función de refuerzo en este dominio queda descrita mediante la ecuación 4.8.

$$r_t = \begin{cases} 1 - ((\phi/\phi_{\text{máx}})^2 + (x/(l/2))^2)/2 & \text{si } -\phi_{\text{máx}} < \phi < \phi_{\text{máx}} \ y \ -l/2 < x < l/2, \\ -1 & \textit{en caso contrario.} \end{cases} \tag{4.8}$$

Por lo tanto, el refuerzo se calcula como $r_t = 1 - ((\phi/\phi_{\text{máx}})^2 + (x/(l/2))^2)/2$ cuando el péndulo y el carro se encuentran dentro de los límites permitidos (i.e., $-\phi_{\text{máx}} < \phi < \phi_{\text{máx}} \ y \ -l/2 < x < l/2$, y -1 en caso contrario. Con el fin de transformar este dominio en estocástico, se ha añadido ruido aleatorio a las acciones y a los refuerzos siguiendo una distribución Gaussiana con desviación 10^{-4}.

4.2.3. *Helicopter*

El dominio del helicóptero de la competición de aprendizaje por refuerzo está basado en el simulador desarrollado por el grupo de Andrew Ng [Abbeel *et al.*, 2008] (Figura 4.3). El refuerzo es máximo cuando el péndulo se encuentra en posición vertical encima del carro, y el carro se encuentra en el centro de la pista.

Figura 4.3: Imagen del helicóptero por radio control simulado en este dominio.

Los agentes tratan de controlar el helicóptero de forma que se mantenga lo más estable posible en el aire. Es un dominio que incluye varios desafíos:

- Dinámica: Los efectos impredecibles del viento y la complicada dinámica no lineal dificultan el problema.
- Exploración y explotación: Debe llevarse a cabo con mucho cuidado puesto que el helicóptero puede llegar a chocar.

Para llevar a cabo esta tarea, los agentes deben manipular 4 acciones continuas (tabla 4.3) sobre la base de un espacio continuo de estados de 12 dimensiones (tabla 4.2).

En cada paso de tiempo el helicóptero recibe una recompensa inmediata calculada como se muestra en la ecuación 4.9, donde s es el estado en el que se encuentra el helicóptero y t la posición sobre la cual el helicóptero debe permanecer estable.

$$r_t = -\sum_i (s_i - t_i)^2 \tag{4.9}$$

La dinámica que rige el comportamiento del dominio se puede consultar en los trabajos de Abbeel et al. [Abbeel *et al.*, 2006; Abbeel *et al.*, 2008]. Un episodio en este dominio consiste de 6000 pasos, cada uno con una duración de 0.1 segundos. El episodio finaliza

x	posición con respecto al eje x
y	posición con respecto al eje y
z	posición con respecto al eje z
u	velocidad a lo largo del eje x
v	velocidad a lo largo del eje y
w	velocidad a lo largo del eje z
ϕ	rotación alrededor del eje x
θ	rotación alrededor del eje y
w	rotación alrededor del eje z
p	velocidad angular alrededor del eje x
q	velocidad angular alrededor del eje y
r	velocidad angular alrededor del eje z

Tabla 4.2: Características de los estados en el dominio *helicopter*.

a_1	paso cíclico longitudinal (alerones)
a_2	paso cíclico latitudinal (elevadores)
a_3	rotor de cola (timón)
a_4	rotor principal

Tabla 4.3: Características de las acciones en el dominio *helicopter*.

prematuramente si el helicóptero choca. Se trata de un dominio generalizado puesto que diferentes parámetros pueden modificar el comportamiento del dominio (i.e., se pueden generar diferentes *Sequential Decision Problems* (SDP), para el entrenamiento y la evaluación). Los dominios generalizados tratan de evitar los buenos resultados conseguidos a través de sobreajustar los parámetros o del sobreadecuamiento del algoritmo al problema concreto que se está resolviendo.

En este dominio la mayoría de los enfoques utilizados para el aprendizaje de los agentes se basan en la utilización de redes de neuronas [Koppejan and Whiteson, 2009], combinadas con técnicas evolutivas [Martín and de Lope Asiaín, 2009], produciendo una búsqueda directa en el espacio de políticas.

4.2.4. *Octopus Arm*

En este dominio, los agentes tratan de aprender a manejar un brazo robótico (con forma de tentáctulo) a través de un entorno viscoso, para dirigirlo hacia un punto meta desconocido. El brazo está formado por compartimentos que pueden ser contraídos de forma independiente (Figura 4.4).

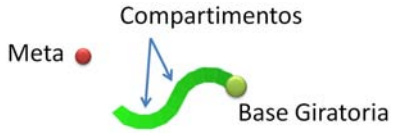


Figura 4.4: Octopus Arm.

Se requiere la coordinación entre los músculos de los diferentes compartimentos para dotar al brazo de movimientos que permitan llegar al objetivo. Este dominio es de gran

interés debido a sus enormes espacios de estados y acciones y a la dinámica no lineal que lo controla. El espacio de estados está formado por 82 variables continuas y el espacio de acciones por 32 variables también de naturaleza continua. Las 82 variables de estado corresponden a los 10 puntos que el brazo tiene en su lado ventral, y los 10 puntos que tiene en su parte dorsal, donde cada punto está descrito por 4 variables continuas correspondientes a su posición y a su velocidad. A estas variables hay que sumarles otras 2 correspondientes a la posición angular de la base, y su velocidad. Las 32 acciones se deben a las 3 fuerzas de contracción dorsal, transversal y ventral de los músculos de los 10 compartimentos, a las que hay que sumar otras 2 referidas a la rotación en sentido antihorario y en sentido horario que se aplica a la base del brazo (Figura 4.5).

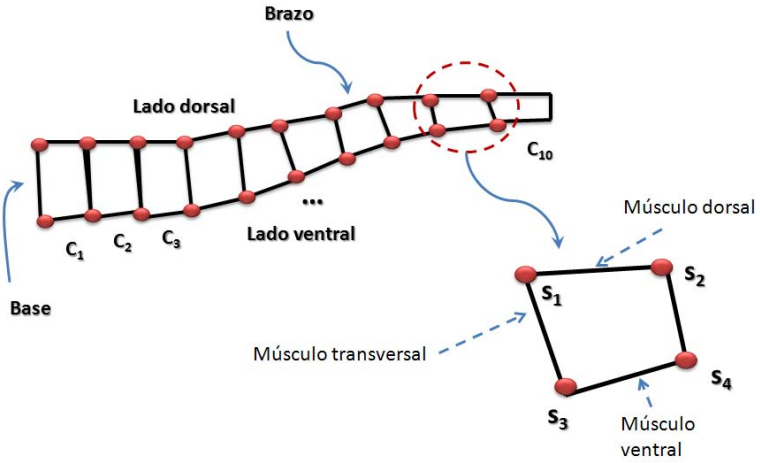


Figura 4.5: Representación de los estados y las acciones en el dominio *Octopus Arm*.

La descripción con la dinámica del dominio se puede encontrar en [Engel *et al.*, 2005]. En este dominio, se asigna un refuerzo $r_t = -1$ si no se alcanza la meta y de 20 en caso contrario.

4.2.5. SIMBA

Se trata de un simulador empresarial que utiliza las mismas variables, relaciones y eventos que ocurren en el mundo real de los negocios. Su propósito es proporcionar a los usuarios una visión integral de una compañía, utilizando las técnicas básicas de gestión empresarial, tratando de simplificar la complejidad que ocurre en el mundo real y centrándose en los principios y contenidos económicos más relevantes, puesto que se trata de un simulador con fines educativos [Borrajo *et al.*, 2009; Borrajo *et al.*, 2010; Borrajo *et al.*, 2011]. No obstante, el simulador presenta una serie de características que lo hacen también interesante para su uso en investigación [García *et al.*, 2012]. A continuación se describe de forma general el simulador, y se dan las pautas para convertirlo en un dominio que puede ser utilizado para la investigación, y más concretamente para la aplicación de técnicas de aprendizaje por refuerzo.

Descripción General

En primer lugar, se trata de un simulador de tipo competitivo donde equipos de par-

ticipantes humanos pueden competir contra otras compañías que pueden estar gestionadas de forma automática por agentes o por otros equipos (Figura 4.6).

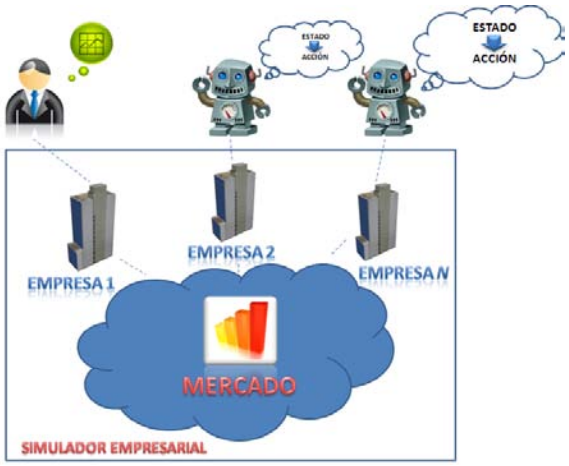


Figura 4.6: SIMBA.

Por otro lado, el simulador es también multi-funcional en el sentido de que permite gestionar las principales áreas funcionales de una compañía. Las actividades de una empresa se dividen en áreas funcionales con el propósito de repartir las actividades y los objetivos.

Área Funcional	**Actividades Gestionadas**
Gestión General	Indicadores de Posicionamiento del Negocio
	$Análisis = \begin{cases} Estratégico \\ de\ la\ Competencia \\ de\ Equilibrio \end{cases}$
Recursos Humanos	Planificación de RRHH
	Análisis de Productividad de RRHH
Operaciones	Planificación de la Capacidad de Producción
	Productividad Laboral
	Previsión de la Demanda
	Gestión de Inventarios
	Análisis de los Costes de la Producción
Finanzas	Balance
	Pérdidas y Ganancias
	Planificación Financiera
	$Análisis = \begin{cases} Ratios\ Económicos \\ Flujo\ Efectivo \\ Préstamos \end{cases}$
	Valoración de Existencias
	Periodo Medio de Maduración
	Valoración de Mercado
Marketing	Análisis de la Demanda
	$Políticas\ de = \begin{cases} Producto \\ Precio \\ Comunicación \\ Distribución \end{cases}$

Tabla 4.4: Diferentes áreas funcionales de una compañía en SIMBA

La Tabla 4.4 muestra las diferentes áreas funcionales presentes en una compañía dentro de SIMBA. Por cada área funcional se muestran ejemplos de actividades que son gestionadas dentro de ese área. Existen cinco áreas funcionales: el área de recursos humanos, el área de operaciones, el área de finanzas, el área de marketing y el área de gestión general. El área de recursos humanos tiene como principal objetivo estipular los operarios o trabajadores que llevarán a cabo las diferentes tareas. Se encarga del reclutamiento y selección del personal capaz, responsable y adecuado para los puestos de trabajo de la empresa. En un restaurante por ejemplo, este área sería la encargada de contratar en función de un proceso de selección la persona encargada de cocinar y las encargadas de atender las mesas. El área de operaciones toma en cuenta todo lo relacionado con el funcionamiento de la empresa, desde la contratación al pago de personal, la firma de cheques, verificar que los empleados cumplen con el horario, el pago a proveedores, control de inventarios, gestión del negocio.

En cambio, el área de finanzas es la encargada de llevar un sistema contable en el que se detallen todos los ingresos y gastos de la compañía. La emisión de facturas, el análisis de los préstamos, las proyecciones de ingresos por ventas, y los costes asociados con el desarrollo del negocio se consideran en este área. El área de marketing se encarga de establecer las funciones de las personas responsables de la estrategia de marketing, publicidad, diseño de la marca del producto, la distribución del mismo. Por último, el área de gestión general es la cabeza de la empresa, la que establece los objetivos de la misma en función de un plan de negocios, y la que se encarga de tomar las decisiones en situaciones críticas [Carysforth, 2006].

El juego en SIMBA se organiza en torno a lo que se denomina una *simulación*. Una simulación consta de un mercado, integrado por diferentes compañías, cada una gestionada por un equipo de humanos o por un agente. Cada simulación a su vez, está dividida en periodos o turnos de juego, de ahí que SIMBA se considere un juego basado en turnos (*turn-based game*) [Andersen *et al.*, 2009]. En cada uno de estos turnos o periodos, los jugadores toman sus decisiones a la vez, haciendo que SIMBA caiga también dentro de la categoría de *juegos ejecutados simultáneamente* (*simultaneously-executed games* o *"We-Go" Games*) [Miller, 2003; Grant *et al.*, 2008]. Cada periodo de decisión es equivalente a un trimestre del ejercicio fiscal en la simulación. Seleccionando esta unidad de tiempo, SIMBA refleja patrones de actividad anuales en la compañía, considerando ciclos largos (correspondientes a cuatro trimestres), pero también considerando ciclos más cortos (trimestres). Para que cada simulación en SIMBA sea diferente, hay un gran número de variables que modifican el entorno de juego y que se asocian a cada uno de los mercados dentro de una simulación. Estas variables se pueden clasificar en dos grandes grupos: el entorno económico y los parámetros internos.

- **Entorno Económico.** El entorno económico está compuesto por 15 variables. La modificación de estos valores se lleva a cabo de acuerdo al comportamiento del mercado y a las decisiones que toman los equipos. Algunas de estas variables incluyen los costes de productividad, el coste de las materias primas, interés sobre los activos y pasivos, y el IPC.

- **Parámetros Internos.** SIMBA contiene además 93 parámetros internos destinados a reflejar las fluctuaciones de comportamiento de ciertos parámetros ambientales o las especificaciones del mercado. Entre estos parámetros se encuentran el valor de la demanda, la tasa de crecimiento de ventas, criterios para la depreciación de activos de activos fijos, los impuestos, etc.

Las variables que componen el entorno económico y los parámetros internos modifican

su valor en cada periodo de la simulación estableciendo la dinámica de evolución y las fluctuaciones de comportamiento de los mercados. La creación de una simulación en SIMBA involucra los pasos que se describen a continuación.

1. En primer lugar se crea el mercados que forma parte de la simulación. También se asigna el entorno económico de partida y los parámetros internos descritos anteriormente. El número de periodos de decisión que se van a ejecutar y la planificación temporal de las distintas decisiones también se establecen en este paso.

2. Se define el número de compañías que lo van a integran el mercado. Se asignan las situaciones de partida (i.e., los datos históricos) a cada una de las compañías. Cada compañía comienza a partir de unos datos históricos que abarcan dos años de su vida útil. Estos datos históricos reflejan los resultados que obtuvo la compañía y las decisiones que tomó en periodos anteriores. En total son necesarias 170 variables por cada periodo histórico. Este número de variables hay que multiplicarlo por el número de compañías compitiendo en un mercado.

3. Los participantes humanos o agentes se asignan a las compañías creadas. De esta forma se decide si una compañía va a estar gestionada de forma manual o automática.

En cada periodo y mercado de una simulación en SIMBA se llevan a cabo cuatro pasos fundamentales tal y como se detalla en la figura 4.7.

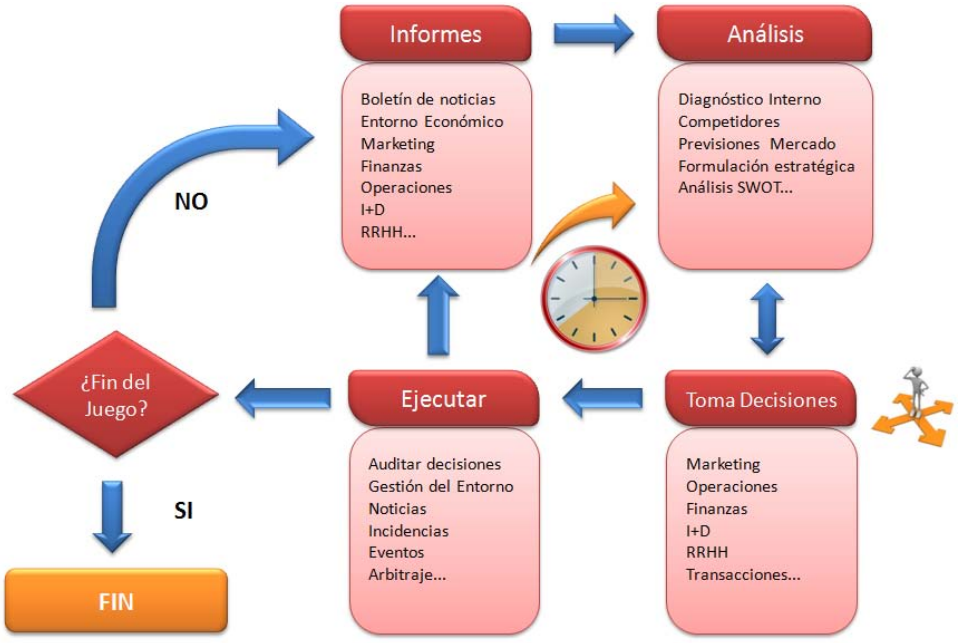


Figura 4.7: Flujo de Trabajo en SIMBA.

Estos pasos se describen brevemente a continuación.

1. **Informes.** En este paso, SIMBA reporta a los diferentes equipos información sobre el estado económico de la compañía, las compañías competidoras y el mercado. En total se proporcionan 170 variables para cada uno de los equipos en cada mercado.

2. **Análisis.** Los información proporcionada en el paso anterior es analizada por los equipos, que establecen una estrategia en función de los resultados obtenidos.

3. **Toma Decisiones.** Los equipos toman las decisiones en las diferentes áreas funcionales de la compañía de acuerdo al análisis de los resultados y la estrategia seleccionada en el paso anterior. Para cada mercado, se debe aproximar el valor de 25 variables, una cifra que puede aumentar si se entran en operaciones de compra-venta entre compañías.

4. **Ejecutar.** Una vez que todos los equipos han tomado sus decisiones, SIMBA calcula los resultados del periodo de acuerdo a las decisiones tomadas por los equipos, y a los valores de las variables del entorno económico y los parámetros internos que hay en ese momento.

Una vez que todos los periodos establecidos en la simulación se han ejecutado, la simulación finaliza quedando a disposición de los participantes los resultados finales conseguidos. En el caso de que la simulación tenga más periodos, se vuelve a comenzar el proceso descrito anteriormente. El tiempo estimado para que un equipo analice la información inicial, desarrolle un diagnóstico y tome las decisiones del periodo correspondiente es de al menos 2 horas, considerando que cada miembro del equipo sea responsable de una de las áreas funcionales de la compañía. Este tiempo estimado es un buen indicador acerca de la complejidad del proceso de toma de decisiones en SIMBA [Borrajo *et al.*, 2009; García *et al.*, 2012].

SIMBA: Un Dominio Generalizado para Aprendizaje por Refuerzo

Un problema de decisión generalizado se caracteriza por los cambios en el entorno que significan que la política de decisión puede cambiar a lo largo del tiempo. De hecho, el SDP subyacente puede cambiar en los diferentes pasos o episodios del proceso de aprendizaje. Formalmente, un dominio generalizado queda definido mediante la tupla $\mathcal{G} =< \Theta, \mathcal{P} >$, donde Θ es el conjunto de todos los posibles problemas de decisión secuencial, y $\mathcal{P}$ es la distribución sobre el conjunto de posibles problemas. En esta Tesis, SIMBA se ha definido como un dominio generalizado. De hecho, el conjunto de posibles problemas Θ es extremadamente grande en el sentido de que cualquier modificación que se produzca en alguna de las variables del entorno económico o de los parámetros internos, hacen que el comportamiento del mercado cambie completamente de un periodo a otro.

A continuación se describe SIMBA como un dominio que permite la aplicación de técnicas de aprendizaje por refuerzo. SIMBA puede considerarse un dominio markoviano, estocástico, episódico, grande y generalizado (Sección 2.1). Una simulación en SIMBA se considera un *episodio* puesto que establece una forma de división en la interacción entre del agente y el simulador. Además, un periodo en SIMBA se considera un paso de aprendizaje por refuerzo, en el sentido de que nos proporciona una forma de división para las simulaciones. Para concluir la descripción de SIMBA como un Proceso de Decisión de Markov, a continuación se describen el espacio de estados S, el espacio de acciones A, la función de transición T, y la señal de refuerzo r.

- **Espacio de estados.** El estado, $s \in S$, obtenido en cada paso en un episodio está compuesto por las 170 variables continuas que recibe cada participante.

- **Espacio de acciones.** Los participantes agentes o humanos deben aproximar el valor de 25 variables de naturaleza continua relativas a las diferentes áreas funcionales de la compañía que están gestionando. Por lo tanto, una acción $a \in A$ está compuesta por estas 25 variables continuas.

- **Función de transición.** $T : \Psi \times U \rightarrow \Psi$, donde $\Psi = \{s_1, s_2, \ldots, s_p\}$ son los estados correspondientes a los p participantes en una simulación, y $U = \{a_1, a_2, \ldots, a_p\}$ son las decisiones correspondientes a los p participantes. Cuando un paso del episodio finaliza, SIMBA integra los estados de los participantes $\Psi_t = \{s_1, s_2, \ldots, s_p\}$, las decisiones $U_t = \{a_1, a_2, \ldots, a_p\}$, los valores de las variables del entorno económico y de los parámetros internos, para generar la información correspondiente a los nuevos estados, Ψ_{t+1}.

- **Señal de refuerzo.** El refuerzo inmediato $r \in \mathbb{R}$ que se va a considerar en esta Tesis es el resultado del ejercicio (i.e., los beneficios) que una compañía obtiene cuando ejecuta una acción a en un estado s. Desde el punto de vista de aprendizaje por refuerzo, el objetivo es maximizar el refuerzo total recibido. En este dominio, como en otros dominios clásicos como *Keepaway* [Stone *et al.*, 2005], los refuerzos inmediatos recibidos en cada paso son distintos de cero. No obstante, el problema es *delayed* en la medida en que es necesario hacer sacrificios a corto plazo con el fin de maximizar el refuerzo total recibido a lo largo del tiempo.

SIMBA como Marco de Investigación Multi-Agente

SIMBA puede ser visto como un entorno multi-agente donde cada agente gestiona una compañía de forma diferente. En SIMBA, los agentes coexisten, tomando sus acciones de forma simultanea e independiente. Los agentes no son informados de las acciones que han tomado o que van a tomar el resto de agentes involucrados en la simulación. Es decir, un agente no intenta emular las acciones de un agente competidor ni tampoco trata de predecir sus valores. En cualquier caso, conviene señalar que la forma en que el mundo cambia cuando el agente i toma sus decisiones en el estado s depende también de las acciones que el resto de agentes toma en sus respectivos estados.

La figura 4.8 muestra a SIMBA como un marco para la investigación multi-agente utilizando RL-Glue [Tanner and White, 2009].

RL-Glue es un estándar que permite la conexión de agentes de aprendizaje por refuerzo, entornos y programas de experimentos. En la versión actual, RL-Glue solo permite la conexión de un único agente de aprendizaje por refuerzo, pero en esta Tesis se han propuesto una serie de modificaciones para permitir la conexión de múltiples agentes. En la versión original de RL-Glue hay cuatro programas: el agente, el entorno, el experimento y RL-Glue. El agente implementa los algoritmos de aprendizaje por refuerzo y la política de comportamiento que se ejecuta en cada momento. El programa del entorno implementa la dinámica del mundo y genera los estados y los refuerzos de acuerdo a las acciones que se hayan ejecutado. El programa de experimentos dirige la ejecución del experimento. La propuesta que se hace en esta Tesis es incluir dos módulos intermedios entre los agentes y RL-Glue (i.e., *Control de Agentes*) y entre el entorno y RL-Glue (i.e., *SIMBA Virtual*).

Los equipos humanos gestionan sus compañías a través de un servidor Web. Los agentes representan una alternativa a los equipos humanos (i.e., no será necesario un equipo de personas para gestionar una compañía, sino que ésta podrá estar gestionada por un único

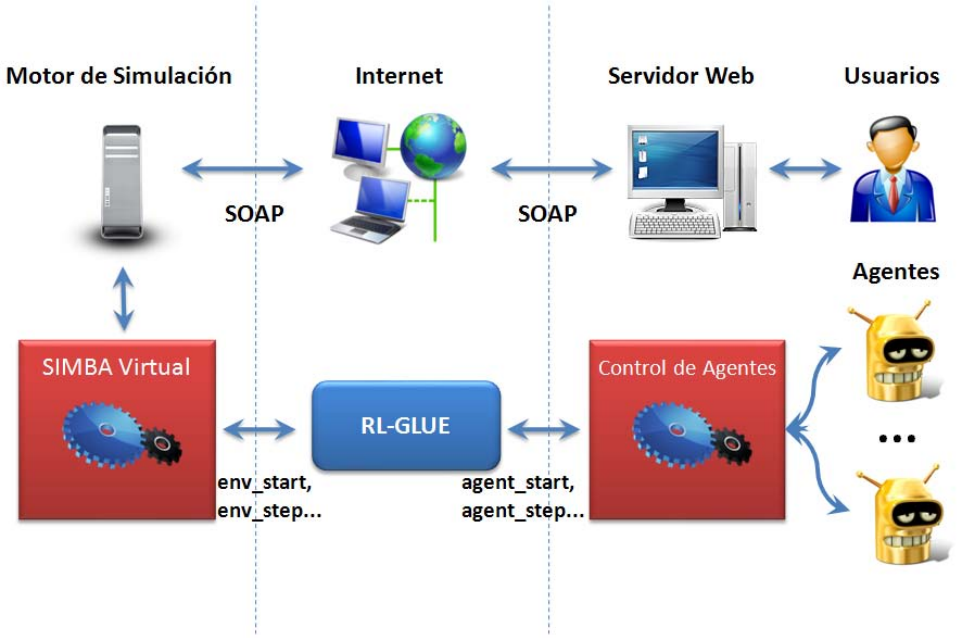


Figura 4.8: SIMBA como marco multi-agente.

agente). No obstante, la arquitectura propuesta permite que convivan en una misma simulación compañías gestionadas por equipos humanos, y compañías gestionadas por agentes. En cada paso, los agentes reciben los resultados obtenidos por sus compañías del *Motor de Simulación*. Los agentes utilizan esta información para tomar la acción correspondiente al periodo actual. El módulo *Control de Agentes* coordina los diferentes agentes conectados al sistema. En una dirección, el módulo *Control de Agentes* recoge las acciones de todos los agentes y las combina en una única acción, $U = \{a_1, a_2, \ldots, a_p\}$. En la otra dirección, distribuye la información de los estados a los agentes adecuados. El módulo *SIMBA Virtual* recibe las acciones de los agentes y las envía al *Motor de Simulación*. Cuando el *Motor de Simulación* recibe todas las acciones de los participantes (tanto humanos como agentes), calcula los nuevos estados, así como el refuerzo que es enviado de vuelta a cada uno de los participantes. Los participantes utilizan la información contenida en los estados para decidir qué acción tomar en el siguiente paso.

4.2.6. Resumen de los dominios

Los dominios descritos que se utilizarán en cada objetivo de la Tesis se indican de forma resumida en la tabla 4.5. En esta tabla también se indica para el segundo objetivo de aprendizaje por refuerzo en entornos con riesgo qué se entiende por riesgo en cada uno de los dominios. En el caso del *Automatic Car Parking Problem* el riesgo viene dado por la posibilidad de que el coche colisione contra la pared o el obstáculo, en el caso del *Cart-Pole* que el péndulo se desbalancé hasta superar cierto ángulo ϕ, en el dominio del *Helicopter* que el helicóptero colisione y, por último, en el caso del simulador empresarial SIMBA, que la empresa entre en bancarrota.

Además, se indican si los dominios son generalizados o no, y el número de características que describen los estados y las acciones. Así, se ve cómo los dominios evolucionan en

	Objetivo 1	**Objetivo 2**	**Generalizado**	**Características Estados**	**Características Acciones**
Car Parking		✔[1]		3	1
Cart-Pole	✔	✔[2]		4	1
Helicopter		✔[3]	✔	12	4
Octopus Arm	✔			82	32
SIMBA	✔	✔[4]	✔	170	25

[1] Riesgo de que el coche colisione contra la pared o el obstáculo

[2] Riesgo de que el péndulo supere cierto ángulo ϕ

[3] Riesgo de que el helicóptero colisione

[4] Riesgo de que la empresa quiebre

Tabla 4.5: Utilización de los dominios en cada uno de los objetivos.

complejidad, siendo el *Automatic car parking problem* el dominio más sencillo y SIMBA el dominio con mayor complejidad, en cuanto al número de características se refiere.

Parte III

Métodos

Capítulo 5

Modelos de aprendizaje para la toma de decisiones en espacios de estados y acciones continuos. G-VQQL y CMAC-VQQL.

En este capítulo se presentan dos modelos de aprendizaje por refuerzo aplicables a dominios con espacios de estados y acciones de grandes dimensiones y de naturaleza continua, y que pretenden dar respuesta, por tanto, al primer objetivo que se persigue en esta Tesis [García *et al.*, 2010; García *et al.*, 2012].

La motivación principal del desarrollo de los modelos propuestos es la necesidad de generalización cuando el problema que intentamos solucionar mediante técnicas de aprendizaje por refuerzo, no es un dominio de *juguete* como los que habitualmente se encuentran en la literatura, sino que se trata de un problema con variables de estado y de acciones continuas y de alta dimensionalidad. La figura 5.1 muestra el esquema de aprendizaje por refuerzo tradicional basado en la utilización de una representación tabular completa (Figura 5.1 (a)), y un esquema que utiliza representaciones alternativas de los espacios de estados y acciones para problemas con espacios de estados y acciones continuos (Figura 5.1 (b)).

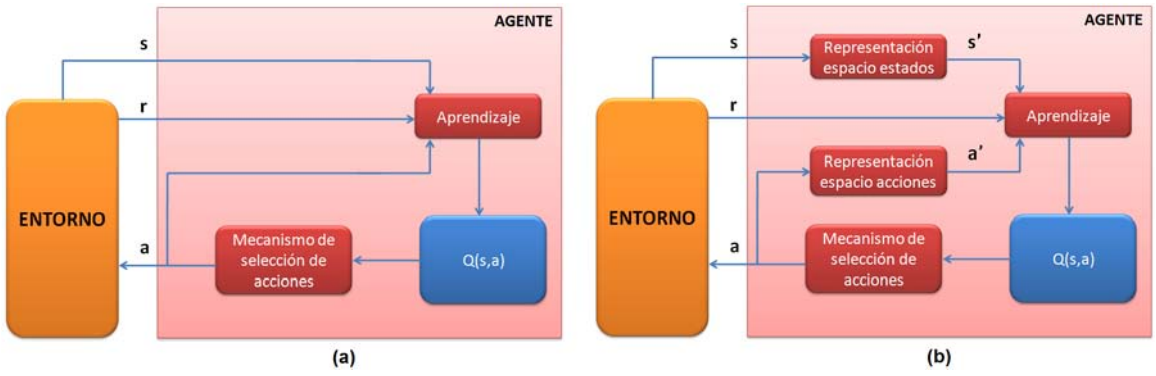


Figura 5.1: (a) Esquema de aprendizaje por refuerzo tradicional. (b) Esquema de aprendizaje por refuerzo con representaciones alternativas para el espacio de estados y de acciones para el tratamiento de problemas con espacios de estados y acciones continuos.

El esquema de la figura 5.1 (a) no es aplicable directamente a problemas con espacios de estados y acciones continuos. Por ello, una de las áreas más activas en aprendizaje por refuerzo es la elaboración de técnicas aplicables a este tipo de problemas. Cuando el agente tiene que tratar con estados y acciones de naturaleza continua es necesaria alguna forma de procesamiento de los estados y acciones percibidos del entorno, bien sea mediante el uso de técnicas de discretización o mediante el uso de alguna técnica de aproximación de funciones. En otras palabras, en estos problemas el agente opera con una representación de los estados y las acciones según alguna técnica en particular. En esta Tesis se proponen dos métodos diferentes para obtener estas representaciones alternativas.

- El método ***G**eneralized **V**ector **Q**uantization* para **Q-L**earning (G-VQQL) utiliza la cuantificación vectorial como técnica de generalización tanto del espacio de estados como del espacio de acciones. En este algoritmo, los módulos *Representación espacio estados* y *Representación espacio acciones* de la figura 5.1 (b) se corresponderían con un cuantificador vectorial: uno para el espacio de estados y otro para el espacio de acciones. En el módulo azul de la figura 5.1 (b), el que se corresponde con la representación de la función $Q(s, a)$, la función $Q(s, a)$ se representaría en forma de tabla manteniendo la forma de representación tradicional.

- El método ***C**erebellar **M**odel **A**rticulation **C**ontroller and **V**ector **Q**uantization* para **Q-L**earning (CMAC-VQQL) combina la utilización de técnicas de aproximación de funciones para generalizar sobre el espacio de estados, CMAC, con la utilización de técnicas de discretización basadas en la cuantificación vectorial para generalizar sobre el espacio de acciones, VQ. Por lo tanto, en este algoritmo, el módulo *Representación espacio estados* de la figura 5.1 (b) estaría integrado por una red CMAC, mientras que el módulo *Representación espacio acciones* se correspondería con un cuantificador vectorial. En este caso, la función $Q(s, a)$ del módulo azul de la figura 5.1 (b) se representaría en forma de función de parámetros.

En las siguientes secciones se describen en detalle las particularidades de cada uno de estos algoritmos propuestos. La sección 5.1 describe en detalle el algoritmo G-VQQL, y la sección 5.2 describe el algoritmo CMAC-VQQL. La sección 5.3 presenta los experimentos que se han llevado a cabo y los resultados que se han obtenido empleando estos dos algoritmos en diferentes dominios de evaluación. Por último, en la sección 5.4 se discuten los resultados obtenidos y el rendimiento y la aplicabilidad de los algoritmos propuestos.

5.1. Algoritmo *Generalized Vector Quantization* for Q-Learning (G-VQQL)

El modelo G-VQQL surge a partir de las ideas planteadas en el modelo VQQL [Fernández and Borrajo, 1999] para espacios de estados continuos, y también utiliza el Algoritmo de Lloyd Generalizado (***G**eneralized **L**loyd **A**lgorithm*, GLA) [Lloyd, 1957] descrito en la sección 2.3.2 para obtener un conjunto de prototipos representativos para el espacio de estados y el espacio de acciones. Una vez que se obtiene una representación *discreta* del problema que se trata de resolver, se emplea el algoritmo Q-Learning para aprender la función $Q(s, a)$ en forma tabular (Figura 5.2).

La cuantificación apareció en el campo de las comunicaciones digitales [Haykin, 1988]. A partir de la necesidad de transmitir y almacenar gran cantidad de información surge

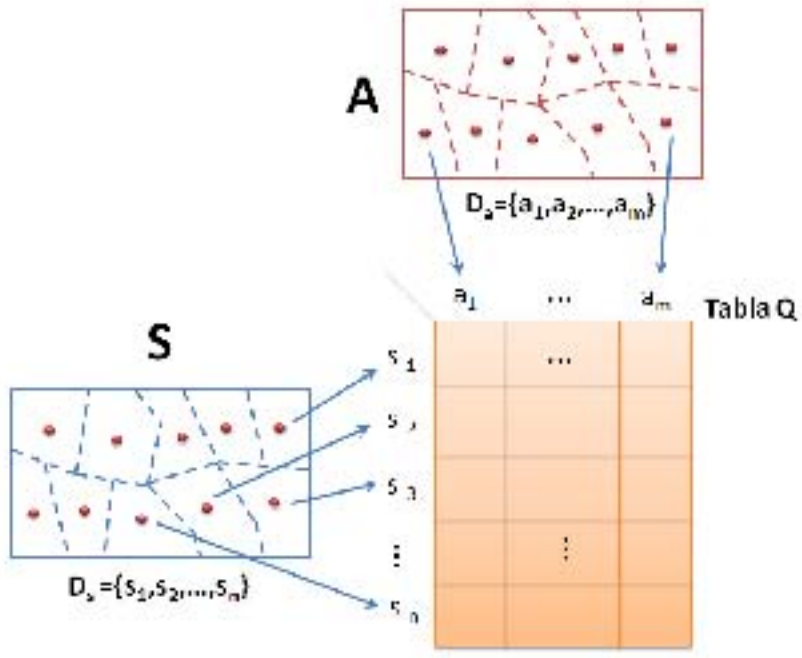


Figura 5.2: Representación tabular de la función $Q(s, a)$ en el algoritmo G-VQQL.

la idea de representarla de otra forma, una forma más económica que además permita transmitirla digitalmente. Este variar de aspecto la información para conseguir unos determinados objetivos (i.e., la transmisión digital y la compresión de la cantidad de información a transmitir), se le conoce como cuantificación. Esta cuantificación permite convertir una representación de la información en otra sin alterar substancialmente su significado, una propiedad importante puesto que el receptor de la señal cuantificada debe ser capaz de reconstruir con la mayor exactitud posible la señal original transmitida. Además de su aplicación al campo de las comunicaciones digitales, la cuantificación vectorial también ha sido utilizada con éxito como técnica de compresión de imágenes [Asari, 2005; Sathappan and Pannirselvam, 2011] y para *clustering* [Linde *et al.*, 1980; Kekre *et al.*, 2012]. A partir de esta descripción de la cuantificación vectorial, se deriva su aplicación directa en la discretización de los espacios de estados y acciones continuos en problemas de aprendizaje por refuerzo. La cuantificación vectorial permite pasar de tener conjuntos infinitos o muy grandes de estados y acciones a conjuntos reducidos que permiten posteriormente aproximar la función de valor mediante la utilización de algún algoritmo de aprendizaje por refuerzo como Q-Learning.

El algoritmo G-VQQL consta de dos pasos fundamentales.

- **Aprendizaje de los cuantificadores vectoriales.** En primer lugar, hay que construir los cuantificadores vectoriales para el espacio de estados y para el espacio de acciones. Tal como se muestra en la figura 5.2, G-VQQL *discretiza* el problema continuo mediante el algoritmo GLA obteniendo dos conjuntos de prototipos diferentes: un conjunto de prototipos para el espacio de estados, y otro conjunto para el espacio de acciones. Cada uno de estos prototipos representa una región de Voronoi del espacio de estados o acciones original. La construcción de los cuantificadores vectoriales se realiza a partir de un número suficiente de tuplas de experiencia. La tabla Q tiene

tantas filas como prototipos resultantes en el espacio de estados, y tantas columnas como prototipos resultantes en el espacio de acciones.

- **Aprendizaje de la política de comportamiento.** Una vez que se tiene una representación tabular de la política de comportamiento, ésta se aprende utilizando el algoritmo Q-Learning [Sutton and Barto, 1998].

Las siguientes secciones describen en detalle cada uno de los pasos de los que se compone el algoritmo G-VQQL. La sección 5.1.1 describe en detalle el primer paso del algoritmo, es decir, la construcción de los cuantificadores. La sección 5.1.2 describe el segundo paso del algoritmo, que realiza el aprendizaje de la política de comportamiento a partir de la versión discreta del problema obtenida en el paso anterior.

5.1.1. Primer Paso: Aprendizaje de los Cuantificadores Vectoriales

El primer paso para el algoritmo G-VQQL es recoger un número suficiente de tuplas de experiencia que le permita posteriormente construir los cuantificadores. La figura 5.1 muestra cómo se lleva a cabo esta recogida de experiencia y la posterior construcción de los cuantificadores.

Recogida de experiencia $(K, H, T = \emptyset, \pi_T) \rightarrow (T, \Gamma, VQ^A, D_a, VQ^{S'}, D_{s'})$
Dados:
Un número máximo de episodios, K.
Un número máximo de pasos, H.
Un conjunto vacío de tuplas de experiencia, $T = \emptyset$.
La política de comportamiento, π_T.
1. Recoger tuplas de experiencia.
Para $k = 1$ a K **hacer** ;; Para cada episodio
Seleccionar el estado inicial, s_1
Para $h = 1$ a H **hacer** ;; Para cada paso del episodio
Seleccionar la acción a_h mediante la política π_T.
Ejecutar la acción a_h, observar el refuerzo r_{h+1} y el nuevo estado s_{h+1}
Generar la tupla de experiencia $< s_h, a_h, s_{h+1}, r_{h+1} >$ e insertarla en T
2. Reducir la dimensión del espacio de estados (Opcional).
Construir de manera supervisada una función de selección de características, Γ, a partir de T.
Sea T_s el conjunto de estados en T.
Aplicar selección de características mediante la proyección $\Gamma : S \rightarrow S'$.
Sea $T'_s = \Gamma(T_s)$.
3. Discretizar el espacio de estados.
Usar GLA para discretizar el espacio de estados, $D_{s'} = \{s'_1, s'_2, ..., s'_n\}$, $s'_i \in S'$, a partir de T'_s.
Sea $VQ^{S'} : S' \rightarrow D_{s'}$ la función, que dado cualquier estado en S', devuelve el vecino más cercano en $D_{s'}$.
4. Discretizar el espacio de acciones.
Sea T_a el conjunto de acciones en T.
Usar GLA para discretizar el espacio de acciones, $D_a = \{a_1, a_2, ..., a_m\}$, $a_i \in A$, a partir de T_a.
Sea $VQ^A : A \rightarrow D_a$ la función, que dada cualquier acción en A, devuelve el vecino más cercano en D_a.
5. Devolver T, Γ, VQ^A, D_a, $VQ^{S'}$, $D_{s'}$

Tabla 5.1: Recogida de experiencia para el algoritmo *Generalized Vector Quantization* para Q-Learning.

La recogida de experiencia se llevará a cabo siguiendo la política de comportamiento π_T especificada en la línea 04 de la tabla 5.1, que en la mayoría de los casos será aleatoria. No obstante, dado que el cometido de esta primera fase del algoritmo es explorar la mayor parte de los espacios de estados y acciones (o aquellas zonas más prometedoras), en algunos dominios, donde la exploración aleatoria solo conduce a explorar de una manera pobre el espacio [Martinez-Gil *et al.*, 2011], es necesario utilizar otras políticas de comportamiento. Por ejemplo, la exploración aleatoria puede producir una exploración parcial del espacio en aquellos dominios donde no hay refuerzos positivos salvo en un estado meta. La exploración aleatoria debería ser capaz de llegar a la meta (o a sus inmediaciones) en alguna ocasión, aunque es fácil comprobar que en dominios realistas con varias dimensiones para el espacio de estados y de acciones es extremadamente difícil alcanzar el estado meta en alguna ocasión mediante un comportamiento aleatorio. El dominio del *Octopus Arm* (sección 4.2.4) de la competición de aprendizaje por refuerzo es un ejemplo claro de esta situación. La exploración aleatoria en este dominio sólo permite explorar de forma parcial regiones muy alejadas del estado meta. Por lo tanto, el posterior diseño que se haga de los cuantificadores vectoriales a partir de los ejemplos recogidos en esta interacción aleatoria, no servirá para aprender la política óptima (que consiste en llegar al estado meta en la menor cantidad de pasos posibles) puesto que no contamos con algún prototipo que represente de forma adecuada la región a la que pertenece el estado meta.

En estos casos se suelen utilizar diferentes estrategias para explorar los espacios de estados y acciones. Una de las estrategias que ha sido utilizada con éxito consiste en guiar al agente de aprendizaje mediante el uso de comportamientos "razonables" (no óptimos), provistos por un experto (*teacher*) [Driessens and Džeroski, 2004]. La exploración aleatoria puede necesitar mucho tiempo para alcanzar aquellas zonas "interesantes" de los espacios de estados y acciones. La utilización de estos comportamientos "razonables" permite encontrar zonas "interesantes" del espacio de estados que pueden incrementar enormemente la velocidad de convergencia del algoritmo de aprendizaje. Por "interesantes" se consideran aquellas zonas del espacio que resultan útiles para posteriormente aprender la función de valor óptima [Smart and Kaelbling, 2000; Smart and Kaelbling, 2002]. En nuestro caso, utilizamos estos comportamientos expertos para recoger tuplas de experiencia de las zonas más relevantes del espacio que nos permitan obtener una buena discretización del espacio de estados y de acciones, para posteriormente aprender una política óptima (o cercana a la óptima).

Por otro lado, en dominios donde el número de características que describen los estados es muy elevado, se pueden emplear técnicas de selección de características con el fin de reducir el número de las mismas, reduciendo al mismo tiempo la complejidad del problema. Esta selección de características se lleva a cabo en el paso 2 (líneas 13 a 16 del algoritmo en la tabla 5.1). Los algoritmos de aprendizaje por refuerzo funcionan mejor cuanto menor es el número de dimensiones de los estados y las acciones [Coons *et al.*, 2008]. Los métodos de selección de características en aprendizaje por refuerzo se pueden dividir en dos: los métodos de envoltura (*wrapper methods*) y los métodos de filtro (*filter methods*) [Guyon and Elisseeff, 2003]. En los métodos de envoltura las características se seleccionan dependiendo del proceso de aprendizaje posterior. Se basan en probar subconjuntos de características en el algoritmo de aprendizaje dado y observar el rendimiento obtenido. Se trata, por tanto, de un tipo de aprendizaje no supervisado que require de un meta-aprendizaje para buscar en el espacio de características qué subconjunto de estas características es el que tiene un buen rendimiento. El algoritmo de aprendizaje es una subrutina en torno a la cual se envuelve el selector de características [Whiteson, 2010]. No obstante, estos métodos

tienen un gran coste computacional a la hora de evaluar los conjuntos de características candidatas, e incrementan además el número de interacciones requeridas con el entorno para aprender una política óptima (i.e., *sample complexity*) [Kakade, 2003]. Puesto que los métodos de aprendizaje por refuerzo son en su mayoría evaluados *on-line*, es esencial tratar de minimizar el número de interacciones para alcanzar un rendimiento óptimo.

En los métodos de filtro, en cambio, las características son seleccionadas independientemente del proceso de aprendizaje posterior. En este caso, el proceso de selección de características es supervisado puesto que se considera que los ejemplos están *etiquetados*, y se trata de filtrar aquellas características que son irrelevantes. Kroon y Whiteson [Kroon and Whiteson, 2009] tratan de establecer la dependencia implítica que existe entre los estados y los refuerzos durante las interacciones con el entorno realizando una evaluación recursiva en un MDP factorizado, lo que requiere un alto coste computacional por un lado, y conocer el modelo por otro.

El método de selección de características que se propone en esta Tesis se enmarca dentro de los métodos de filtro, y es similar en espíritu al propuesto por Kroon y Whiteson [Kroon and Whiteson, 2009], pero sin realizar la evaluación recursiva, lo que aligera la carga computacional, ni factorizar el MDP, el cual se desconoce *a priori* puesto que estamos asumiendo aprendizaje por refuerzo libre del modelo. En nuestro caso, aplicamos un algoritmo de aprendizaje automático al conjunto de tuplas de experiencias T para establecer la dependencia existente entre las características de los estados y los refuerzos. En otras palabras, sólo se seleccionan aquellas características de estados que más influyen sobre la función de refuerzo. De esta forma, se construye la función de selección de características Γ a partir de los estados y refuerzos contenidos en T. En cualquier caso, este paso es opcional y si no se ejecuta, se puede considerar la función de proyección como una función identidad $\Gamma : S \rightarrow S$, dejando el espacio de estados original.

Por último, en los pasos 3 y 4 (líneas 17 a 25), se usa el algoritmo GLA para discretizar tanto el espacio de estados como el de acciones respectivamente. Las discretizaciones se construyen a partir del conjunto de tuplas de experiencia, T, obtenido en el paso 1. El conjunto de estados S', T'_s, se utiliza como entrada al algoritmo GLA [Linde *et al.*, 1980] para obtener la discretización del espacio de estados, $D_{s'}$ (línea 18). La función $VQ^{s'}$ permite realizar un mapeo entre los estados en S' a $D_{s'}$, donde $D_{s'}$ es el espacio de estados discretizado $D_{s'} = s'_1, s'_2, ..., s'_n$ para $s'_i \in S'$ (línea 19). El conjunto de acciones A, T_a, se emplea como entrada a GLA para obtener la discretización del espacio de acciones, D_a (línea 23). La función VQ^A permite mapear acciones de A a D_a, donde D_a es el espacio de acciones discretizado $D_a = a_1, a_2, ..., a_m$ para $a_i \in A$ (línea 24).

5.1.2. Segundo Paso: Aprendizaje de la Política de Comportamiento

Una vez que se ha recogido la experiencia necesaria y se han construido los cuantificadores vectoriales, se ejecuta el algoritmo G-VQQL (Tabla 5.2). En la versión del algoritmo G-VQQL mostrado en la tabla 5.2, la versión en línea, se realiza una nueva exploración para aprender la política de comportamiento utilizando las discretizaciones realizadas y los cuantificadores obtenidos en el paso anterior.

En el paso 1 del algoritmo (líneas 9 a 12 en el algoritmo de la tabla 5.2), se inicializan los valores de la tabla Q que se emplearán para representar la función $Q(s, a)$. Esta tabla tendrá tantas filas como estados discretos o prototipos hayan resultado de aplicar en el paso anterior el algoritmo GLA a los estados T'_s, y tantas columnas como acciones discretas o prototipos hayan resultado de aplicar el algoritmo GLA a las acciones T_a. En el paso 2

del algoritmo (líneas 13 a 20), se aprende la tabla Q mediante una nueva exploración con el entorno que puede ser conducida mediante alguna estrategia de exploración/explotación como $\epsilon - greedy$. La acción que se selecciona en cada momento se extrae del conjunto de acciones discreto, $D_a = \{a_j, j \in 1, \ldots, m\}$, construido mediante el algoritmo GLA en el paso anterior. En cada paso se mapean la proyección del estado actual, $\Gamma(s_h)$, y la proyección del nuevo estado, $\Gamma(s_{h+1})$ a los estados discretizados correspondientes utilizando para ello el cuantificador vectorial $VQ^{S'}$ (línea 19). El estado actual discretizado, $VQ(\Gamma(s_h))$, la acción ejecutada, $a_h \in D_a$, el nuevo estado discretizado resultante, $VQ(\Gamma(s_{h+1}))$, y el refuerzo recibido, r_{h+1}, son utilizados para realizar una actualización mediante Q-Learning en la función $Q(s, a)$ (línea 20).

G-VQQL en línea $(\Gamma, D_a, VQ^{S'}, D_{s'}, K, H, \alpha, \gamma) \rightarrow Q$

00 **Dados:**
01 El cuantificador vectorial del espacio de estados, $VQ^{S'}$.
02 La función de selección de características, Γ.
03 El conjunto de prototipos para los estados, $D_{s'} = \{s'_i, i \in 1, \ldots, n\}$.
04 El conjunto de prototipos para las acciones, $D_a = \{a_j, j \in 1, \ldots, m\}$.
05 Una tabla $Q(s, a)$ con n filas, y m columnas.
06 Un número máximo de episodios de aprendizaje a ejecutar, K.
07 Un número máximo de pasos por episodio de aprendizaje, H.
08 Los parámetros de aprendizaje, α y γ.
09 **1. Inicializar la tabla Q.**
10 **Para** $i = 1$ a n **hacer** ;; Para cada $s' \in D_{s'}$
11 **Para** $j = 1$ a m **hacer** ;; Para cada $a \in D_a$
12 $Q(i, j) = 0$
13 **2. Aprender la tabla Q.**
14 **Para** $k = 1$ a K **hacer** ;; Para cada episodio
15 Seleccionar el estado inicial, s_1
16 **Para** $h = 1$ a H **hacer** ;; Para cada paso del episodio
17 Seleccionar la acción $a_h \in D_a$ siguiendo alguna estrategia de exploración/explotación (e.g. $\epsilon - greedy$)
18 Ejecutar la acción a_h, observar el refuerzo r_{h+1} y el nuevo estado s_{h+1}
19 Transformar: $s_h \leftarrow VQ^{S'}(\Gamma(s_h))$, $s_{h+1} \leftarrow VQ^{S'}(\Gamma(s_{h+1}))$.
20 $Q(s_h, a_h) \leftarrow Q(s_h, a_h) + \alpha(r_{h+1} + \gamma \max_a Q(s_{h+1}, a) - Q(s_h, a_h))$
21 **3. Devolver** Q.

Tabla 5.2: Algoritmo *Generalized Vector Quantization* para Q-Learning en su versión en línea.

No obstante, puesto que el primer paso del algoritmo *Recogida de experiencia* (tabla 5.1) ya lleva a cabo una extensa exploración de los espacios de estados y acciones, esa misma exploración puede ser utilizada para realizar actualizaciones en la función $Q(s, a)$ tal como se muestra en el algoritmo de la tabla 5.3. En esta versión del algoritmo G-VQQL, la versión fuera de línea, se sustituye el paso 2 del algoritmo G-VQQL en línea en el cual se lleva a cabo un nuevo proceso de exploración del entorno (líneas 13 a 20 en el algoritmo de la tabla 5.2), por la exploración realizada en el paso anterior, es decir, las actualizaciones en la función $Q(s, a)$ se llevan a cabo a partir de las transiciones de estados recogidas en T (líneas 13 a 16 en el algoritmo de la tabla 5.3). No es necesaria una estrategia de exploración/explotación como $\epsilon - greedy$ para que guíe el nuevo proceso de exploración como en el caso anterior, sino que la exploración se conducirá nuevamente por las mismas regiones por las que se llevó a cabo para obtener el conjunto de tuplas de experiencias, T. Esto hace que, a diferencia del caso anterior, la acción que se tome para realizar la actualización, a_h, no pertenezca

directamente al conjunto de acciones discretas $D_a = \{a_j, j \in 1, \ldots, m\}$. En este caso la acción a_h extraída de T_a puede tomar cualquier valor continuo dentro del rango definido para la acción, y que se ha obtenido siguiendo la política de comportamiento π_T utilizada para explorar en el paso anterior. De ahí que en esta versión del algoritmo sea necesario, además, el cuantificador vectorial para el espacio de acciones, $VQ^A : A \rightarrow D_a$, que se empleará para determinar qué acción dentro del conjunto $D_a = \{a_j, j \in 1, \ldots, m\}$ le corresponde a la acción a_h extraída de la tupla de experiencia en T (línea 15).

G-VQQL Fuera de Línea $(T, \Gamma, VQ^A, D_a, VQ^{S'}, D_{s'}, \alpha, \gamma) \rightarrow Q$	
00	**Dados:**
01	El conjunto de tuplas de experiencia, T.
02	El cuantificador vectorial del espacio de estados, $VQ^{S'}$.
03	El cuantificador vectorial para el espacio de acciones, VQ^A.
04	La función de selección de características, Γ.
05	El conjunto de prototipos para los estados, $D_{s'} = \{s'_i, i \in 1, \ldots, n\}$.
06	El conjunto de prototipos para las acciones, $D_a = \{a_j, j \in 1, \ldots, m\}$.
07	Una tabla $Q(s, a)$ con n filas, y m columnas.
08	Los parámetros de aprendizaje, α y γ.
09	**1. Inicializar la tabla Q.**
10	**Para** $i = 1$ a n **hacer** ;; Para cada $s' \in D_{s'}$
11	**Para** $j = 1$ a m **hacer** ;; Para cada $a \in D_a$
12	$Q(i, j) = 0$
13	**2. Aprender la tabla Q.**
14	**Para cada** $< s_h, a_h, s_{h+1}, r_{h+1} > \in T$ **hacer** ;; Para cada tupla de experiencia en T
15	Transformar: $s_h \leftarrow VQ^{S'}(\Gamma(s_h))$, $a_h \leftarrow VQ^A(a_h)$, $s_{h+1} \leftarrow VQ^{S'}(\Gamma(s_{h+1}))$.
16	$Q(s_h, a_h) \leftarrow Q(s_h, a_h) + \alpha(r_{h+1} + \gamma \max_a Q(s_{h+1}, a) - Q(s_h, a_h))$
17	**3. Devolver** Q.

Tabla 5.3: Algoritmo *Generalized Vector Quantization* para Q-Learning en su versión fuera de línea.

El principal inconveniente que plantea el algoritmo G-VQQL en su versión fuera de línea, es que el aprendizaje quedará fuertemente sesgado por la exploración que se haya llevado a cabo en el primer paso del algoritmo. Por ejemplo, en el caso de que se utilice un comportamiento experto, π_T, en la obtención del conjunto de tuplas de experiencia T, la política aprendida por el algoritmo G-VQQL en su versión fuera de línea estará limitada por la calidad de las trayectorias que es capaz de producir el comportamiento, π_T, y como se comentó en la anterior sección, el comportamiento π_T es un comportamiento "razonable" aunque sub-óptimo.

5.2. Algoritmo *Cerebellar Model Articulation Controller and Vector Quantization* for Q-Learning (CMAC-VQQL)

El algoritmo CMAC-VQQL hace uso de técnicas basadas en codificación gruesa, CMAC, que se describieron en la sección 2.3.1 para generalizar sobre el espacio de estados y en el algoritmo GLA para discretizar el espacio de acciones. Por lo tanto, CMAC-VQQL utiliza de manera combinada técnicas de aproximación de funciones y técnicas de discretización para abordar problemas continuos y de grandes dimensiones (Figura 5.3).

Como se describió en la sección 2.3.1, CMAC extrae las características asociadas a un estado s formando un vector de características, $\phi_s = (\phi_s(1), \phi_s(2), \ldots, \phi_s(v))$. Cada una

de estas características establece su valor dependiendo de si el estado pertenece o no a una determinada región del espacio de estados. Si el estado se encuentra dentro de una de estas regiones, la característica asociada a esa determinada región tomará el valor de 1 y tomará el valor de 0 en caso contrario. Se trata, por tanto, de características binarias. De esta forma, a partir del vector de características, ϕ_s, puede establecerse de una forma *gruesa* cuál es la localización del estado s en el espacio de estados. Cada vector de características tiene asociado un vector de parámetros, θ_a, de forma que la función de valor, $Q(s,a)$, no se representa en forma de tabla, sino como una función parametrizada usando el vector de parámetros $\vec{\theta}_a$.

$$Q_a(s) = \vec{\theta}_a^T \phi_s = \sum_{i=1}^{v} \theta_a(i)\phi_s(i) \tag{5.1}$$

Como se puede observar en la figura 5.3, el algoritmo CMAC-VQQL maneja m aproximadores diferentes, $\{Q_{a_1}, Q_{a_2}, Q_{a_3}, \ldots, Q_{a_m}\}$, uno por cada prototipo resultante de aplicar el algoritmo GLA sobre el espacio de acciones.

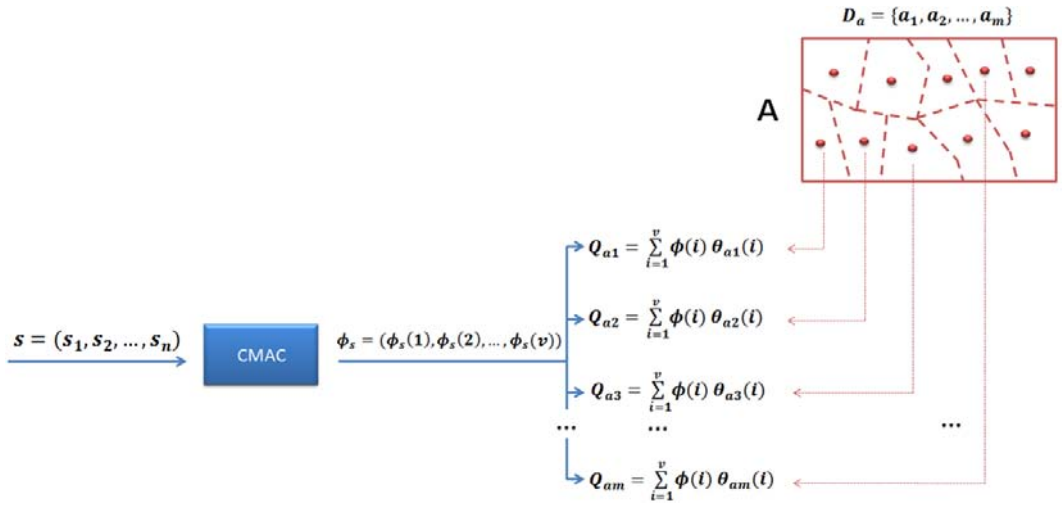


Figura 5.3: Representación de forma paramétrica de la función $Q(s,a)$ en el algoritmo CMAC-VQQL.

El algoritmo CMAC-VQQL consta de dos pasos fundamentales:

- **Aprendizaje del cuantificador vectorial asociado a las acciones.** En primer lugar, como en el caso anterior, se recogen un número lo suficientemente grande de tuplas de experiencia. A partir de este conjunto de tuplas se diseña el cuantificador vectorial asociado a las acciones. Cada uno de los prototipos resultantes en el proceso de construcción del cuantificador vectorial representa una región de Voronoi en el espacio de acciones.

- **Aprendizaje de la política de comportamiento.** En primer lugar, a partir de las regiones de Voronoi para el espacio de acciones definidas en el paso anterior se diseña la función que mapea estados a vectores de características. En este paso, se contará con un aproximador diferente por cada región de Voronoi tal y como queda reflejado en la figura 5.3. A diferencia del algoritmo G-VQQL, el aprendizaje del comportamiento

óptimo en CMAC-VQQL consiste en ajustar en cada uno de los pasos de aprendizaje los vectores de parámetros asociados a cada una de las acciones, $\{\theta_{a_1}, \theta_{a_2}, \theta_{a_3}, \ldots, \theta_{a_m}\}$ mediante el algoritmo de aprendizaje por refuerzo Q-Learning.

En las siguientes secciones se presentan de forma detallada cada uno de los pasos de los que está compuesto el algoritmo CMAC-VQQL. La sección 5.2.1 describe el primer paso del algoritmo en el cual se explora el entorno con el fin de recoger tuplas de experiencia que nos permitan diseñar el cuantificador vectorial para el espacio de acciones. A continuación, la sección 5.2.2 describe el segundo paso en el cual se diseña CMAC, los vectores de parámetros y se aprende la política de comportamiento.

5.2.1. Primer Paso: Aprendizaje del Cuantificador Vectorial asociado a las Acciones.

La descripción del primer paso del algoritmo se encuentra recogida en la tabla 5.4.

	Recogida de experiencia $(K, H, T = \emptyset, \pi_T) \rightarrow (T, \Gamma, VQ^A, D_a)$
00	**Dados:**
01	Un número máximo de episodios, K.
02	Un número máximo de pasos, H.
03	Un conjunto vacío de tuplas de experiencia, $T = \emptyset$.
04	La política de comportamiento, π_T.
05	**1. Recoger tuplas de experiencia.**
06	**Para** $k = 1$ a K **hacer** ;; Para cada episodio
07	Seleccionar el estado inicial, s_1
08	**Para** $h = 1$ a H **hacer** ;; Para cada paso del episodio
09	Seleccionar la acción a_h mediante la política π_T.
10	Ejecutar la acción a_h, observar el refuerzo r_{h+1} y el nuevo estado s_{h+1}
11	Generar la tupla de experiencia $< s_h, a_h, s_{h+1}, r_{h+1} >$ e insertarla en T
12	**2. Reducir la dimensión del espacio de estados (Opcional).**
13	Construir de manera supervisada una función de selección de características, Γ, a partir de T.
14	Aplicar selección de características mediante la proyección $\Gamma : S \rightarrow S'$.
15	**3. Discretizar el espacio de acciones.**
16	Sea T_a el conjunto de acciones en T.
17	Usar GLA para discretizar el espacio de acciones, $D_a = \{a_1, a_2, ..., a_m\}$, $a_i \in A$, a partir de T_a.
18	Sea $VQ^A : A \rightarrow D_a$ la función, que dada cualquier acción en A, devuelve el vecino más
19	cercano en D_a.
20	**4. Devolver** T, Γ, VQ^A, D_a

Tabla 5.4: Recogida de experiencia para el algoritmo CMAC-VQQL y diseño del cuantificador vectorial.

En primer lugar se recogen las tuplas de experiencia de la forma $< s_h, a_h, s_{h+1}, r_{h+1} >$ donde s_h es el estado en el que se encuentra el agente, a_h es la acción que decide ejecutar en el estado s_h, s_{h+1} es el estado al que se transita tras ejecutar la acción a_h en el estado s_h, y r_{h+1} es el refuerzo que recibe el agente en esta transición (líneas 05 a 11 del algoritmo en la tabla 5.4). Esta recogida de experiencia se ejecuta durante K episodios compuestos por H pasos. La recogida de las tuplas se lleva a cabo, como en el caso del algoritmo G-VQQL, mediante la política de comportamiento π_T. Este comportamiento, π_T, puede ser aleatorio o puede ser un comportamiento "razonable" como se discutió en la sección 5.1.1. Generalmente en dominios de baja dimensionalidad y de poca complejidad, un comportamiento aleatorio

nos permitirá explorar la mayor parte del espacio de estados y de acciones. No obstante, en dominios más complejos y con una alta dimensionalidad tanto en el espacio de estados y de acciones, la exploración aleatoria es insuficiente para construir un conjunto de tuplas de experiencia representativo para, en este caso, discretizar el espacio de acciones y construir el cuantificador vectorial asociado. En el algoritmo CMAC-VQQL estamos interesados en un comportamiento π_T, que sea capaz de explorar eficientemente el espacio de acciones para posteriormente diseñar el cuantificador, en contraposición al modelo G-VQQL, en el cuál el comportamiento π_T tenía que ser capaz de explorar eficientemente tanto el espacio de estados como del espacio de acciones.

No obstante, si nos fijamos nuevamente en el dominio del *Octopus Arm* (sección 4.2.4) de la competición de aprendizaje por refuerzo, podemos ver como la exploración aleatoria, aún estando interesados en sólo explorar eficientemente el espacio de acciones, no es válida por dos motivos fundamentales. El espacio de acciones del dominio tiene 32 dimensiones, lo que implica que existan muchas posibles acciones que se pueden ejecutar en el entorno. En otras palabras, la exploración aleatoria tardaría un tiempo prohibitivo en recuperar del entorno un conjunto de tuplas lo suficiente representativo con la mayoría de las acciones que son ejecutables en el dominio. Esta exploración no sólo tendría un alto coste temporal en cuanto a tiempo de recuperación de tuplas, sino también un alto coste de almacenamiento (debido al gran número de tuplas de experiencia) y computacional (puesto que este conjunto de tuplas serán la entrada al algoritmo GLA). Por otro lado, muchas de las acciones seleccionadas por el comportamiento aleatorio, aún siendo validas, serán acciones sin coherencia dentro del dominio. Por ejemplo, una posible acción en el dominio del *Octopus Arm* puede consistir en contraer al mismo tiempo todos los músculos dorsales y ventrales, lo que no origina ningún tipo de movimiento en el brazo. Por lo tanto, la utilización de un comportamiento base permite explorar en el espacio de acciones válidas y coherentes, lo que permite incrementar la velocidad de convergencia en el segundo paso del algoritmo.

En el paso 2 del algoritmo se reduce el espacio de estados mediante selección de características (líneas 12 a 14 del algoritmo en la tabla 5.4). Como en el caso del algoritmo G-VQQL descrito en la sección anterior, se aplica un algoritmo de aprendizaje automático supervisado al conjunto de tuplas de experiencias T para establecer la dependencia existente entre las características de los estados y los refuerzos. Mediante esta selección, se construye la función $\Gamma : S \rightarrow S'$ que permite seleccionar para un estado $s \in S$ con $S \subset \Re^n$, las características cuyos valores más influyen sobre la función de refuerzo. La aplicación de la función Γ sobre un estado s da lugar a un estado s' de dimensión menor que n. En cualquier caso, la ejecución de este paso es opcional, y si no se ejecuta se puede considerar la función Γ como la función identidad, $\Gamma : S \rightarrow S$. En el paso 3 del algoritmo (líneas 15 a 19 del algoritmo en la tabla 5.4) se discretiza el espacio de acciones mediante el algoritmo GLA que utiliza como entrada el conjunto de acciones recuperado en el paso 1, T_a.

5.2.2. Segundo Paso: Aprendizaje de la Política de Comportamiento.

La descripción del segundo paso del algoritmo se muestra en la tabla 5.5.Una vez que se han recuperado del entorno un conjunto T de tuplas de experiencia que se han utilizado para diseñar el cuantificador vectorial para el espacio de acciones, se procede a aprender la política de comportamiento. Como en el caso del algoritmo G-VQQL presentado en la tabla 5.2, en el algoritmo CMAC-VQQL en línea que se muestra en la tabla 5.5, se lleva a cabo un nuevo proceso de exploración para aprender la política de comportamiento óptima.

El algoritmo recibe la función Γ que permite seleccionar las características más relevantes

para el espacio de estados, junto con el conjunto de acciones discreto $D_a = \{a_j, j \in 1, \ldots, m\}$ construidos en el paso anterior. También recibe como parámetro el tamaño del vector de parámetros y del vector de características, v. Cuanto más elevado es este parámetro, mayor es la resolución de CMAC, aunque se requerirá un mayor coste computacional. Por último el algoritmo recibe el número de episodios, K, y el número de pasos por episodio, H, que se llevarán a cabo durante el nuevo proceso de exploración con el entorno.

CMAC-VQQL en línea $(\Gamma, D_a, v, K, H, \alpha, \gamma) \rightarrow Q_a(s)$

Dados:
  La función de selección de características, $\Gamma : S \rightarrow S'$.
  El conjunto de prototipos para las acciones, $D_a = \{a_j, j \in 1, \ldots, m\}$.
  Tamaño del vector de características y del vector de parámetros, v.
  Un número máximo de episodios de aprendizaje a ejecutar, K.
  Un número máximo de pasos por episodio de aprendizaje, H.
  Los parámetros de aprendizaje, α y γ.
1. Diseño de CMAC e inicialización.
  Construcción de la función que dado un estado devuelve un vector de características para ese estado,
  $\Gamma(s) \rightarrow \phi_{\Gamma(s)}$, donde $\phi_{\Gamma(s)} \subset \Re^v$ y $\phi_{\Gamma(s)} = (\phi_{\Gamma(s)}(1), \phi_{\Gamma(s)}(2), \ldots, \phi_{\Gamma(s)}(v))$ con $\phi_{\Gamma(s)}(i) \in \{0, 1\}$.
  Construcción de los vectores de parámetros $\{\vec{\theta}_{a_j}, j \in 1, \ldots, m\}$, donde cada $\theta_{a_j} \subset \Re^v$
  y $\theta_{a_j} = (\theta_{a_j}(1), \theta_{a_j}(2), \ldots, \theta_{a_j}(v))$ con $\theta_{a_j}(i) \in \Re$.
  Inicializar cada $\vec{\theta}_{a_j}$ arbitrariamente.
2. Aprender la función Q.
  Para $k = 1$ a K **hacer** ;; Para cada episodio
    Seleccionar el estado inicial, s_1
    Para $h = 1$ a H **hacer** ;; Para cada paso del episodio
      Seleccionar la acción $a_h \in D_a$ siguiendo alguna estrategia de exploración/explotación (e.g. $\epsilon - greedy$)
      Ejecutar la acción a_h, observar el refuerzo r_{h+1} y el nuevo estado s_{h+1}
      Calcular $Q_{a_h}(\Gamma(s_h)) = \sum_{i=1}^{v} \theta_{a_h}(i)\phi_{\Gamma(s_h)}(i)$
      Para cada $a_j \in D_a$ **hacer**
        $Q_{a_j}(\Gamma(s_{h+1})) = \sum_{i=1}^{v} \theta_{a_j}(i)\phi_{\Gamma(s_{h+1})}(i)$
      Calcular el error de diferencia temporal: $\delta \leftarrow r + \gamma \max_a Q_a(\Gamma(s_{h+1})) - Q_{a_h}(\Gamma(s_h))$
      $\theta_{a_h} \leftarrow \theta_{a_h} + \alpha\delta$
3. Devolver $Q_a(s)$.

Tabla 5.5: Algoritmo CMAC-VQQL en su versión en línea.

En primer lugar, en el paso 1 del algoritmo (líneas 07 a 12 del algoritmo en la tabla 5.5) se diseña la función que permite mapear de un estado al que se le ha aplicado selección de características, $\Gamma(s)$, a su vector de características correspondiente, $\phi_{\Gamma(s)}$. Hasta ahora se ha explicado CMAC para el caso de *rejillas* n-dimensionales, es decir, cada *rejilla* tenía tantas dimensiones como número de dimensiones tenía un estado (ver por ejemplo la figura 2.4). No obstante, esto es inviable computacionalmente cuando el dominio tiene un espacio de estados con un número elevado de dimensiones. En el caso del dominio del simulador empresarial *SIMBA*, con 12 dimensiones para los estados, si consideramos que cada dimensión la dividimos en 20 intervalos regulares, tendríamos 12^{20} *tejas* por cada *rejilla*, o lo que es lo mismo, el vector de parámetros por cada acción tendría 12^{20} elementos. Si consideramos un nivel de granuralidad fina, con 32 *rejillas* superpuestas, cada vector de parámetros estaría formado por $12^{20} \times 32$ elementos. Por lo tanto, en lugar de considerar *rejillas* n-dimensionales, se consideran *rejillas* uni-dimensionales, es decir, se divide el rango de cada dimensión de estado en intervalos regulares tal como se muestra en la figura 5.4. Realizando esta simplificación, el vector de parámetros por cada acción estará compuesto

por $12 \times 20 \times 32$ elementos.

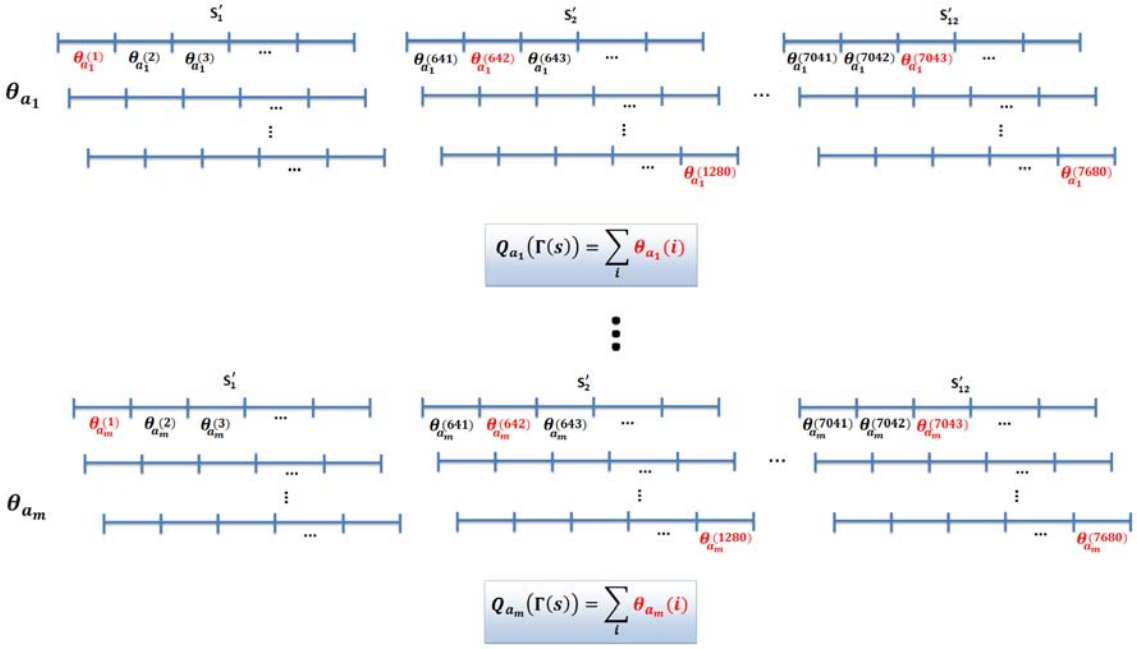


Figura 5.4: Representación de uni-dimensional de las rejillas en CMAC-VQQL.

En el ejemplo de la figura 5.4, el estado resultante de aplicar la selección de características tiene 12 dimensiones, $\Gamma(s) = (s'_1, s'_2, ..., s'_{12})$. En la figura 5.4 se muestran además, por cada dimensión de estado, varias de estas *rejillas* uni-dimensionales superpuestas y desplazadas unas de otras. Cada una de estas *rejillas* está dividida en intervalos regulares, y cada intervalo tiene asociado un elemento del vector de parámetros, $\theta_{a_j}(i)$. En el ejemplo de la figura 5.4 se han considerado 32 rejillas superpuestas por cada dimensión del estado $\Gamma(s)$, donde cada *rejilla* está dividida en 20 intervalos regulares. Por lo tanto, cada vector de parámetros asociado a una acción tiene un tamaño de $v = 7680$ (12 *variables de estado* $\times$ 32 *rejillas* $\times 20$ *intervalos por rejilla* $= 7680$), i.e., $\theta_{a_j} = (\theta_{a_j}(1), \theta_{a_j}(2), \ldots, \theta_{a_j}(7680))$. Un intervalo se encuentra activo si el valor de la variable de estado cae dentro de ese intervalo. En este caso se dice que la característica asociada a ese intervalo toma el valor de 1 (en la figura 5.4 el parámetro correspondiente se encuentra en rojo) y tomará el valor de 0 en caso contrario. De esta forma, la función $Q_{a_j}(\Gamma(s))$ para la acción a_j y el estado $\Gamma(s)$, se calcula como la suma de todos los elementos del vector de parámetros θ_{a_j} asociados a intervalos activos (i.e., el valor de la característica asociada a ese intervalo es igual a 1).

En el paso 2 del algoritmo (líneas 13 a 23 del algoritmo en la tabla 5.5), se realiza un nuevo proceso de exploración para aprender la política de comportamiento óptimo. Este nuevo proceso de exploración puede ser llevado a cabo mediante alguna estrategia de exploración/explotación como $\epsilon-greedy$. A medida que avanza el proceso de aprendizaje, se van actualizando los valores de los pesos asociados a cada uno de los vectores de parámetros, θ_{a_j}, empleando para esta actualización el error de diferencia temporal (*TD-error*) obtenido en cada paso.

Como en el caso del algoritmo G-VQQL, el algoritmo CMAC-VQQL en línea mostrado en la tabla 5.5 tiene también su versión fuera de línea (Tabla 5.6). En el algoritmo CMAC-VQQL fuera de línea la exploración para aprender la política de comportamiento óptima

se lleva a cabo utilizando el conjunto de tuplas de experiencia, T, recuperado del entorno en el primer paso del algoritmo mostrado en la tabla 5.4 (línea 01). Además, a diferencia del algoritmo CMAC-VQQL en línea, CMAC-VQQL fuera de línea recibe como entrada el cuantificador vectorial para el espacio de acciones construido en el primer paso del algoritmo de la tabla 5.4 (línea 03). Como en el caso del algoritmo G-VQQL en línea, las acciones $a_h \in T_a$ se corresponden con acciones ejecutadas por el comportamiento π_T. Por lo tanto, estas acciones pueden tomar cualquier valor dentro del rango definido para una acción. De ahí que, previa actualización del vector de parámetros en el paso 2 del algoritmo en la tabla 5.6, sea necesario identificar mediante el cuantificador VQ^A a qué acción discreta dentro del conjunto $D_a = a_j, j \in 1, \ldots, m$, pertenece la acción a_h recuperada del conjunto de tuplas de experiencia (línea 15).

CMAC-VQQL Fuera de Línea $(T, \Gamma, VQ^A, D_a, v, \alpha, \gamma) \rightarrow Q_a(s)$

00 **Dados:**
01 El conjunto de tuplas de experiencia, T.
02 La función de selección de características, $\Gamma : S \rightarrow S'$.
03 El cuantificador vectorial para el espacio de acciones, VQ^A.
04 El conjunto de prototipos para las acciones, $D_a = \{a_j, j \in 1, \ldots, m\}$.
05 Tamaño del vector de características y del vector de parámetros, v.
06 Los parámetros de aprendizaje, α y γ.
07 **1. Diseño de CMAC e inicialización.**
08 Construcción de la función que dado un estado devuelve un vector de características para ese estado,
09 $\Gamma(s) \rightarrow \phi_{\Gamma(s)}$, donde $\phi_{\Gamma(s)} \subset \Re^v$ y $\phi_{\Gamma(s)} = (\phi_{\Gamma(s)}(1), \phi_{\Gamma(s)}(2), \ldots, \phi_{\Gamma(s)}(v))$ con $\phi_{\Gamma(s)}(i) \in \{0, 1\}$.
10 Construcción de los vectores de parámetros $\{\vec{\theta}_{a_j}, j \in 1, \ldots, m\}$, donde cada $\theta_{a_j} \subset \Re^v$
11 y $\theta_{a_j} = (\theta_{a_j}(1), \theta_{a_j}(2), \ldots, \theta_{a_j}(v))$ con $\theta_{a_j}(i) \in \Re$.
12 Inicializar cada $\vec{\theta}_{a_j}$ arbitrariamente.
13 **2. Aprender la función Q.**
14 **Para cada** $< s_h, a_h, s_{h+1}, r_{h+1} > \in T$ **hacer** ;; Para cada tupla de experiencia en T
15 Transformar: $a_h \leftarrow VQ^A(a_h)$
16 Calcular $Q_{a_h}(\Gamma(s_h)) = \sum_{i=1}^{v} \theta_{a_h}(i)\phi_{\Gamma(s_h)}(i)$
17 Para cada $a_j \in D_a$ **hacer**
18 $Q_{a_j}(\Gamma(s_{h+1})) = \sum_{i=1}^{v} \theta_{a_j}(i)\phi_{\Gamma(s_{h+1})}(i)$
19 Calcular el error de diferencia temporal: $\delta \leftarrow r + \gamma \max_a Q_a(\Gamma(s_{h+1})) - Q_{a_h}(\Gamma(s_h))$
20 $\theta_{a_h} \leftarrow \theta_{a_h} + \alpha\delta$
21 **3. Devolver** $Q_a(s)$.

Tabla 5.6: Algoritmo CMAC-VQQL en su versión fuera de línea.

Nuevamente, la calidad de la política de comportamiento resultante de la aplicación del algoritmo CMAC-VQQL estará segada por la calidad del conjunto de tuplas de experiencia, T, recuperado en el paso anterior del algoritmo. Si el comportamiento π_T utilizado en el primer paso es el comportamiento de un experto, la política resultante estará limitada por la calidad del conjunto de tuplas de experiencia que es capaz de generar este experto.

5.3. Experimentos y Resultados

En esta sección se presentan los experimentos que se han llevado a cabo para analizar el comportamiento de los algoritmos G-VQQL y CMAC-VQQL propuestos. Para este análisis se han empleado los dominios del *Cart-Pole*, el *Octopus Arm* y el dominio del simulador empresarial SIMBA descritos en la sección 4.2. No se aportan resultados en el dominio del

Helicopter puesto que no se ha conseguido la convergencia de los algoritmos G-VQQL y CMAC-VQQL, ni tampoco en el dominio *Automatic Car Parking Problem* al tratarse de un dominio de similar complejidad que el *Cart-Pole*. Para cada dominio se presenta en primer lugar la aplicación del algoritmo G-VQQL, posteriormente la aplicación del algoritmo CMAC-VQQL, y por último una comparación del rendimiento de los algoritmos propuestos con algunos algoritmos extraídos de la literatura (sección 4.1). El rendimiento de los algoritmos G-VQQL y CMAC-VQQL se comparará con los algoritmos CACLA (Tabla 2.7), una aproximación basada en aprendizaje por refuerzo evolutivo (Tabla 2.9), y el algoritmo CMAC (Tabla 2.4).

El algoritmo CACLA ha demostrado obtener mejores resultados en dominios como el *Cart-Pole* que otros algoritmos bien conocidos en aprendizaje por refuerzo para espacios de estados y acciones continuos [van Hasselt and Wiering, 2007]. La aproximación basada en aprendizaje por refuerzo evolutivo que se utiliza en la comparación fue la ganadora de la competición de aprendizaje por refuerzo en el dominio del *Helicopter* en la edición de 2009 [Martín and de Lope Asiaín, 2009]. Por último, el algoritmo CMAC empleado en la comparación ha demostrado comportarse mejor que aproximaciones basadas en aprendizaje basado en instancias [Santamaría *et al.*, 1998]. Además, estos algoritmos permiten la comparación de G-VQQL y CMAC-VQQL con un representante de una filosofía de aprendizaje diferente dentro del aprendizaje por refuerzo: un algoritmo basado en arquitecturas del tipo actor-crítico, un algoritmo evolutivo que realiza una búsqueda directa sobre el espacio de políticas, y un algoritmo de aproximación de funciones puro. Se ha llevado a cabo una experimentación exhaustiva con el fin de encontrar las mejores configuraciones de estos algoritmos extraídos de la literatura en cada uno de los dominios. No obstante, por claridad, en las comparativas sólo se muestran los mejores resultados obtenidos. A no ser que se indique lo contrario, las curvas de aprendizaje y las desviaciones que se presentan en cada una de las gráficas de resultados se han construido a partir de 10 procesos de aprendizaje independientes. De la misma forma, en cada gráfica de resultados se muestra en el eje x los episodios y en el eje y el refuerzo acumulado obtenido en cada episodio.

Por último, en estos experimentos se tratan de demostrar dos hechos. Por un lado, en el dominio del *Cart-Pole* donde un comportamiento aleatorio es suficiente para explorar adecuadamente el espacio, se demuestra la rápida convergencia de los algoritmos G-VQQL y CMAC-VQQL frente al resto de algoritmos con los que se comparan. Por otro lado, en los dominios del *Octopus Arm* y el simulador empresarial SIMBA se demuestra las ventajas de extraer conocimiento del dominio a partir de un comportamiento base que puede ser utilizado para mejorar el rendimiento de los algoritmos. En nuestro caso, este comportamiento permite una mejor exploración en los dominios donde un comportamiento aleatorio sólo produce una exploración parcial o inadecuada del espacio, de forma que se puedan obtener mejores discretizaciones que faciliten el posterior proceso de aprendizaje.

5.3.1. *Cart-Pole*

La descripción detallada de este dominio se encuentra en la sección 4.2.2. En esta sección se presenta en primer lugar la aplicación del algoritmo G-VQQL en el dominio *Cart-Pole* (sección 5.3.1.1). Posteriormente se describe la aplicación del algoritmo CMAC-VQQL (sección 5.3.1.2). Por último, se muestra la comparativa del rendimiento de los algoritmos G-VQQL y CMAC-VQQL con algunos algoritmos extraídos de la literatura (sección 5.3.1.3).

5.3.1.1. Ejecución del algoritmo G-VQQL

La ejecución del algoritmo G-VQQL requiere llevar a cabo dos pasos distintos: el aprendizaje de los cuantificadores vectoriales (Tabla 5.1) y el aprendizaje de la política de comportamiento que puede ser en línea realizando un nuevo proceso de exploración (Tabla 5.2) o reutilizando el conjunto de tuplas de experiencia recuperadas en el paso anterior (Tabla 5.3). A continuación se detallan cada uno de estos pasos.

Primer Paso: Aprendizaje de los cuantificadores vectoriales

Para el diseño del cuantificador vectorial asociado al espacio de estados y del cuantificador vectorial asociado al espacio de acciones, es necesario llevar a cabo los pasos descritos en el algoritmo presentado en la Tabla 5.1:

1. Recoger tuplas de experiencia: Previo a la aplicación del algoritmo G-VQQL, en primer lugar es necesario recoger del entorno un conjunto lo suficientemente grande de tuplas de experiencia, T, mediante una determinada política de comportamiento π_T. La figura 5.5 muestra una proyección de los estados resultantes en T, tras utilizar una política de comportamiento π_T aleatoria para realizar la exploración de los espacios de estados y acciones. El conjunto T resultante está compuesto por 50283 tuplas de experiencia.

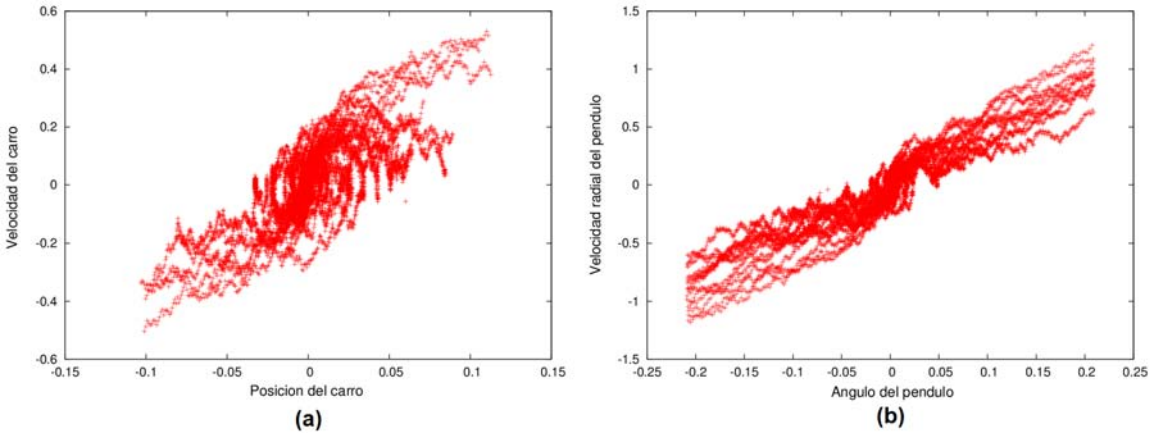


Figura 5.5: Exploración resultante del espacio de estados utilizando un comportamiento π_T aleatorio en el dominio del *Cart-Pole*.

La gráfica 5.5 (a) muestra la posición del carro, x, en el eje de abscisas, y la velocidad del carro, x', en el eje de ordenadas. La gráfica 5.5 (b) muestra el ángulo del péndulo, ϕ, y la velocidad radial del mismo, ϕ'. La figura 5.5 muestra que, aunque se está utilizando un comportamiento aleatorio π_T para realizar la exploración inicial del entorno, los estados obtenidos en T no están distribuidos uniformemente por el espacio de estados sino que siguen una distribución determinada, lo que indica la fuerte relación estadística que hay entre las características de los estados. En el caso de que se realizara una discretización uniforme tomando el rango de cada característica y dividiéndolo en 10 intervalos regulares, se obtendrían 10^4 prototipos que representarían la totalidad del espacio euclídeo representado, puesto que son 4 las características que describen a un estado en este dominio. No obstante, muchos de estos prototipos obtenidos mediante la discretización uniforme nunca serían utilizados, puesto que representan a regiones del espacio de estados sin datos de entrada. El algoritmo GLA utilizado para el diseño del cuantificador vectorial aprovecha las

relaciones estadísticas que existen entre las características de entrada para producir centroides o prototipos sólo en aquellas zonas donde realmente son necesarios, tal y como se comprobará más adelante.

La figura 5.6 muestra la distribución de las acciones contenidas en T y que han sido tomadas durante el proceso de exploración utilizando el comportamiento aleatorio π_T. En el caso del dominio del *Cart-Pole*, el espacio de acciones es uni-dimensional. El valor de la característica asociada a una acción se corresponde con la fuerza que se le imprime al carro en uno u otro sentido con el objetivo de que el palo permanezca en equilibrio encima del carro. Este valor se encuentra comprendido en el rango de valores $[-1, 1]$. En la figura 5.6 se muestra como la exploración aleatoria produce acciones en todo el rango posible de valores para una acción.

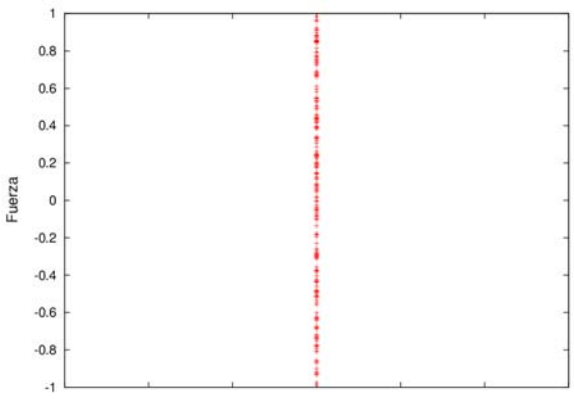


Figura 5.6: Exploración resultante del espacio de acciones utilizando un comportamiento π_T aleatorio en el dominio del *Cart-Pole*.

2. Reducir la dimensión del espacio de estados: En este paso, como se describió en la sección 5.1.1, se seleccionan las características de estado que más relación tienen con la función de refuerzo, y se desechan aquellas que no tienen relación. En otras palabras, se desechan aquellas características para las cuales, independientemente de sus cambios de valor, la función de refuerzo no se ve alterada. El dominio *Cart-Pole* cuenta tan sólo con cuatro características para los estados, la posición del carro, x, la velocidad del mismo, x', el ángulo del péndulo, ϕ, y la velocidad radial asociada al péndulo, ϕ'. Este número de características es fácilmente abordable directamente por el algoritmo GLA, que ha mostrado su eficiencia para discretizar espacios de estados de mayores dimensiones [Fernández and Borrajo, 1999; García *et al.*, 2007], sin necesidad de aplicar ningún filtrado de características. No obstante, en esta sección se analiza la relación existente entre cada característica y la función de refuerzo, para ver si alguna es susceptible de ser eliminada.

Para llevar a cabo este análisis se emplea el conjunto de tuplas de experiencia, T, obtenidas en el paso anterior. La tabla 5.7 muestra la selección de características de estado llevada a cabo por el evaluador ***CfsSubsetEval***, que evalúa la calidad de un subconjunto de características teniendo en cuenta la habilidad individual de predicción de cada una de las características, así como el grado de redundancia entre ellas. Se prefieren los subconjuntos de características que estén altamente correlacionados con la clase y tengan baja intercorrelación [Hall, 1999]. A su vez este evaluador emplea dos métodos de búsqueda diferentes:

1. ***BestFirst***, el cual busca en el espacio de los subconjuntos de atributos utilizando la estrategia *greedy hill-climbing* con *backtracking*.

2. ***GreedyStepWise***, el cual se basa en una búsqueda *hill-climbing* sin *backtracking* que va añadiendo nuevas características al mejor subconjunto de características encontrado, siempre y cuando al hacerlo se consiga una mejora.

Los experimentos se han realizado utilizando la implementación de estos algoritmos que hay en la herramienta de minería de datos WEKA. Para el método de test en la evaluación de los subconjuntos de atributos seleccionados se ha utilizado validación cruzada de 10 *folds*.

Evaluación de Atributos	Búsqueda	x	x'	ϕ	ϕ'
CfsSubsetEval	*BestFirst*	60 %	20 %	90 %	50 %
	GreedyStepWise	60 %	10 %	80 %	40 %

Tabla 5.7: Selección de atributos en el Cart-Pole utilizando *CfsSubsetEval* como evaluador de subconjuntos de características con búsqueda *BestFirst* y *GreedyStepWise*.

La tabla 5.7 muestra por cada característica de estado, las veces que dicha características es seleccionada en cada uno de los 10 *folds* construidos en la validación cruzada. Un valor alto implica que la característica ha sido seleccionada en muchos *folds* y por tanto se ha estimado que tiene una mayor relación con la función de refuerzo. La característica correspondiente a la posición del carro, x, fue seleccionada el 60 % de los *folds* con búsqueda *BestFirst* y cuando se empleó búsqueda *GreedyStepWise*. En cambio, la velocidad del carro, x', es seleccionada en el 20 % de los *folds* cuando se emplea búsqueda *BestFirst* y en el 10 % cuando se empleó la búsqueda *GreedyStepWise*. El ángulo del péndulo, ϕ, está fuertemente relacionado con la función de refuerzo puesto que la característica asociada es seleccionada en el 90 % de los *folds* cuando se emplea búsqueda *BestFirst* y en el 80 % cuando se emplea búsqueda *GreedyStepWise*. Por último, la característica asociada a la velocidad radial del péndulo, ϕ', es seleccionada en el 50 % de los *folds* con búsqueda *BestFirst* y en el 40 % con búsqueda *GreedyStepWise*.

La característica más relevante en este dominio es la característica asociada al ángulo del péndulo, ϕ. Esto tiene sentido puesto que uno de los objetivos en este dominio, tal como se definió en la sección 4.2.2, es mantener el péndulo lo más vertical posible sin que se desbalancé encima del carro. La segunda característica más relevante es la característica relacionada con la posición del carro en la pista, x, que tiene que ver con el segundo objetivo del dominio en el que se trataba de mantener al carro lo más centrado posible en la pista. Aunque la posición del carro, x, y el ángulo del péndulo, ϕ, contribuyen en la misma medida a la función de refuerzo presentada en la sección 4.2.2 en la ecuación 4.8, el algoritmo *CfsSubsetEval* establece que es el ángulo de péndulo, ϕ, la característica que más influye en la función de refuerzo, y que la posición del carro, x, tiene menor influencia. Esto indica que la mayor parte de los episodios generados mediante la política de exploración aleatoria, π_T, en el paso anterior finalizaron porque el ángulo del péndulo, ϕ, superó cierto ángulo (i.e., se desbalanceó), y no porque la posición del carro, x, excediera los límites de la pista. De esta forma, mientras la posición del carro, y el ángulo del péndulo contribuyen en la misma medida en la primera parte de la ecuación 4.8 cuando $-\theta < \phi < \theta$ y $-l/2 < x < l/2$, es el ángulo del péndulo, ϕ, el que contribuye en mayor medida a la segunda parte de la ecuación 4.8 cuando la función de refuerzo toma valor -1. La siguiente característica más relevante según *CfsSubsetEval* es la velocidad radial del péndulo, ϕ'. La última característica, la velocidad del carro, x', es la que menos relevancia tiene.

Por lo tanto, se ha estimado que todas las características de estado en el *Cart-Pole* presentan, en mayor o menor medida, algún tipo de relación con la función de refuerzo, y

se considera que todas son relevantes para llevar a cabo con éxito el posterior proceso de aprendizaje. En este caso se puede suponer, por coherencia con los algoritmos mostrados en las tablas 5.1, 5.2 y 5.3, que la función Γ aplicada es la función identidad : $\Gamma : \{x, x', \phi, \phi'\} \rightarrow \{x, x', \phi, \phi'\}$, y por tanto se tiene que el conjunto de estados T'_s resultante de aplicar la función Γ sobre T_s en el paso 2 de la tabla 5.1 es igual a T_s, $T'_s = \Gamma(T_s) = T_s$.

3. Discretizar el espacio de estados: En adelante, se va a considerar al conjunto de estados T_s el conjunto de estados original sin selección de características en T. La figura 5.7 muestra discretizaciones de diferentes tamaños obtenidos al aplicar el algoritmo GLA a T_s. Estas discretizaciones tienen como finalidad representar el espacio de estados continuo presente en el dominio del *Cart-Pole*. En esta figura se muestra cómo se distribuyen los prototipos o centroides para alfabetos de 8 (Figura 5.7 (a)), 16 (Figura 5.7 (b)), 32 (Figura 5.7 (c)), 64 (Figura 5.7 (d)), 128 (Figura 5.7 (e)), y 256 (Figura 5.7 (f)) centroides. En cada gráfica se muestra en el eje de abscisas el valor correspondiente a la posición del carro, x, y en el eje de ordenadas el valor correspondiente a la velocidad del mismo, x'. En la secuencia de gráficas que se muestran en la figura 5.7 el número de prototipos o centroides calculados se van desdoblando hasta los 256 centroides (Figura 5.7 (f)) donde la masa de puntos resultante ha adquirido una forma similar a la original (Figura 5.5 (a)). Esto es así en cuanto a forma, pero no en cuanto a densidad de puntos, puesto que en la figura 5.7 (f) hay representados 256 puntos mientras que en la figura 5.5 (a) hay representados 50283 puntos correspondientes a los estados en T_s. Por lo tanto, GLA obtiene centroides o prototipos representativos para aquellas zonas donde realmente son útiles, es decir, los prototipos o centroides están más adaptados al conjunto de datos inicial a diferencia de los prototipos obtenidos en otro tipo de discretizaciones como la discretización uniforme.

Una vez obtenida la discretización del espacio de estados, se diseña el cuantificador vectorial correspondiente. Para el caso de la figura 5.5 (a) se obtiene un conjunto discreto de estados representativos o centroides de tamaño 8, $D_s = \{s_1, s_2, \ldots, s_8\}$. El cuantificador vectorial en este caso, $VQ^S : S \rightarrow D_s$, es el encargado de asociar un estado de entrada $s \in S$ al estado más cercano en D_s. Se procede de igual forma para diseñar cuantificadores vectoriales de diferentes tamaños. A medida que se aumentan el tamaño de D_s disminuye la medida de distorsión establecida por la ecuación 2.56. Esto significa que se reduce la distancia existente entre un estado de entrada $s \in S$ y su correspondiente estado representativo o centroide dentro del conjunto de estados D_s. Es decir, el estado $s \in S$ se generaliza cada vez a estados en D_s más similares cuanto mayor es el tamaño de la discretización empleada o mayor es el número de estados en D_s. Aunque un mayor tamaño de la discretización implica un menor error de generalización, también implica un mayor tamaño de la tabla Q lo que puede dificultar la convergencia del algoritmo de aprendizaje. Por lo tanto, hay que buscar para cada problema el tamaño adecuado de la discretización a emplear.

4. Discretizar el espacio de acciones: La aplicación del algoritmo GLA sobre las acciones T_a recogidas anteriormente durante la exploración aleatoria del entorno mediante el comportamiento π_T (Figura 5.6), produce los alfabetos de diferentes tamaños que se muestran en la secuencia de gráficas de la figura 5.8.

La figura 5.8 muestra la distribución de los centroides o prototipos para discretizaciones de tamaño 8 (Figura 5.8 (a)), 16 (Figura 5.8 (b)), 32 (Figura 5.8 (c)), y 64 (Figura 5.8 (f)). En cada gráfica se muestra la fuerza que es aplicada al carro para mantener al péndulo en equilibrio. Como se muestra en la figura 5.6 el comportamiento aleatorio, π_T, utilizado para explorar el entorno generó acciones distribuidas uniformemente a lo largo del rango permitido para las acciones [-1,1]. Esto último unido al hecho de que este dominio está compuesto por sólo una dimensión, y que por tanto el algoritmo GLA no puede beneficiarse de

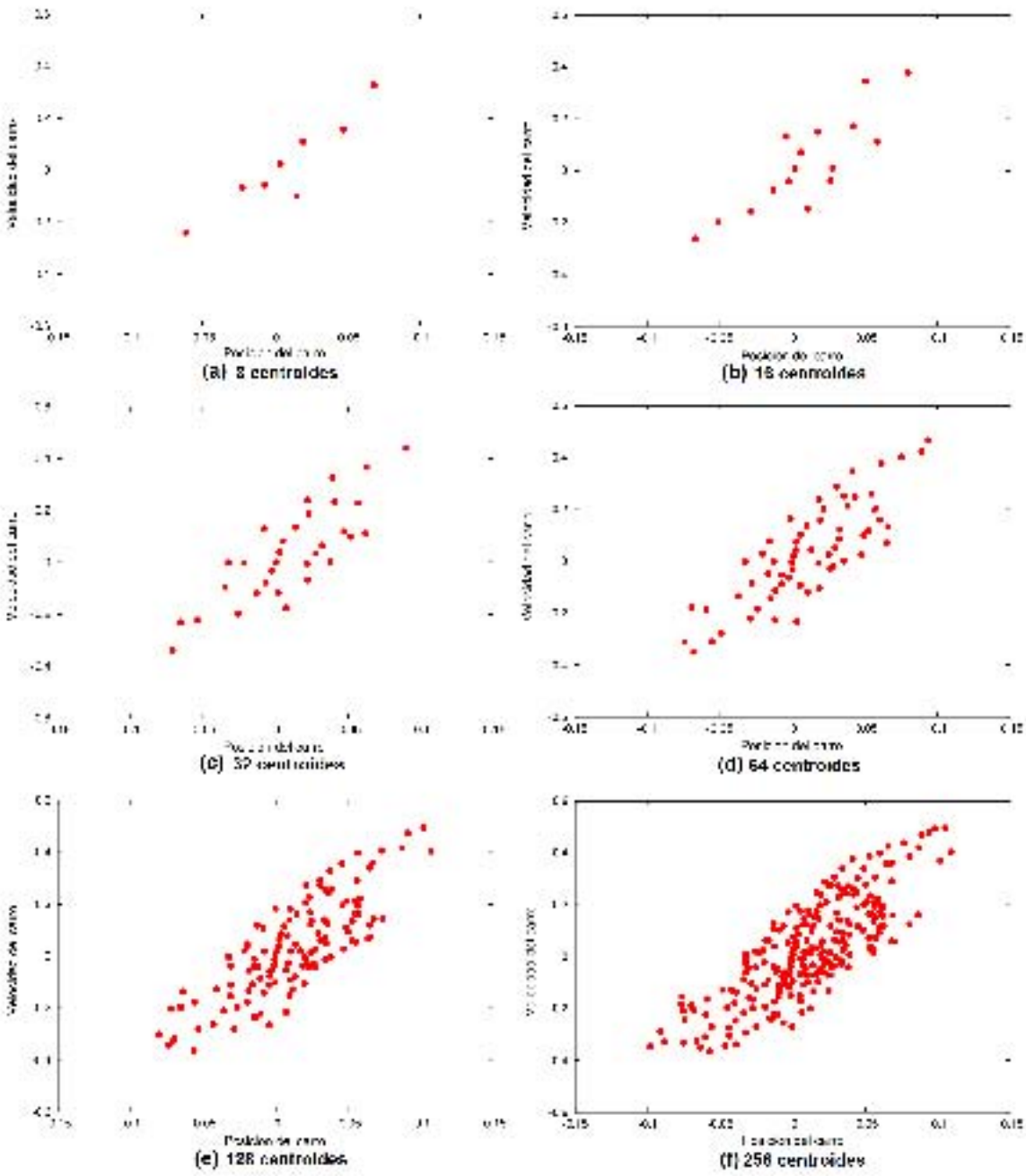


Figura 5.7: Alfabetos de diferentes tamaños obtenidos mediante GLA en el dominio del *Cart-Pole* para el conjunto de estados T_s.

las relaciones estadísticas que aparecen entre variables, hace que la discretización resultante del espacio de acciones sea equivalente a una discretización uniforme.

El proceso de construcción del cuantificador vectorial asociado al espacio de acciones, VQ^A, es análogo al proceso de construcción del cuantificador vectorial asociado al espacio de estados descrito en el paso anterior y que es utilizado en la versión fuera de línea del algoritmo (Tabla 5.3). A partir de las discretizaciones realizadas del espacio de acciones se construye el cuantificador vectorial. Si se selecciona la discretización de tamaño 8 mostrada

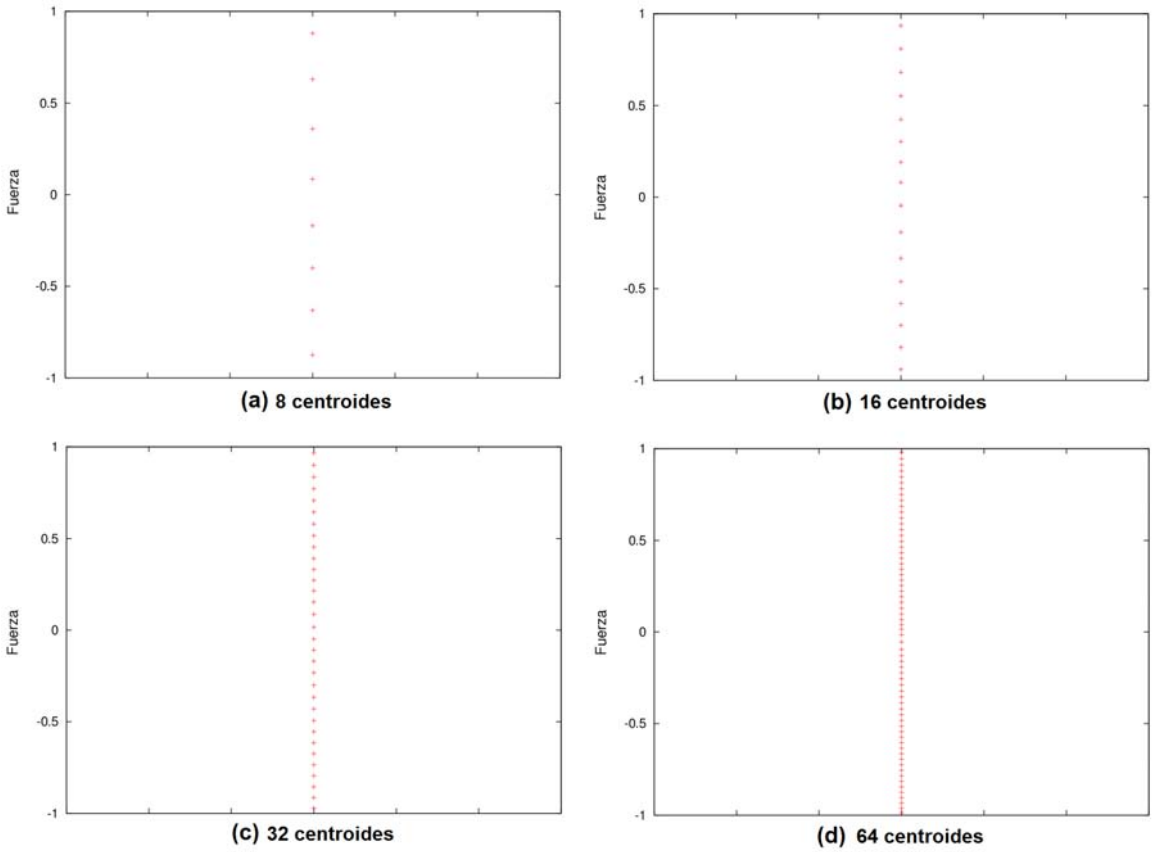


Figura 5.8: Alfabetos de diferentes tamaños obtenidos mediante GLA en el dominio del *Cart-Pole* para el conjunto de acciones T_a.

en la figura 5.8, se obtiene el conjunto de centroides o acciones representativas, $D_a = \{a_1, a_2, \ldots, a_8\}$. El cuantificador vectorial VQ^A, es el encargado de asociar una acción $a \in A$ a la acción más similar en $D_a = \{a_1, a_2, \ldots, a_8\}$. En el caso de la versión del algoritmo en línea (Tabla 5.2), el conjunto de acciones $D_a = \{a_1, a_2, \ldots, a_8\}$ contiene la colección finita de acciones que el agente empleará para aprender la tarea, es decir, el agente está limitado a ejecutar 8 acciones distintas. Si se utiliza una discretización de mayor tamaño, se aumenta el repertorio de acciones que el agente es capaz de ejecutar. No obstante, como para el caso del espacio de estados, discretizaciones de mayor tamaño, implican también un mayor tamaño de la tabla Q, lo que puede dificultar la convergencia de los algoritmos de aprendizaje a la hora de aprender la política de comportamiento óptima.

Segundo Paso: Aprendizaje de la Política de Comportamiento

Una vez que se han diseñado los cuantificadores vectoriales, se ejecuta el paso de aprendizaje en línea de la política de comportamiento descrito en la tabla 5.2, en el cual hay que seleccionar el tamaño de las discretizaciones que se utilizarán para el espacio de estados y el espacio de acciones. En este dominio, el aprendizaje fuera de línea mostrado en la tabla 5.3 no produce ningún tipo de convergencia, por lo que sólo se muestran los resultados del algoritmo en línea (tabla 5.2). La figura 5.9 muestra el proceso de aprendizaje del algoritmo

cuando se utilizan alfabetos de diferentes tamaños para discretizar el espacio de estados y el espacio de acciones. Concretamente se muestra el rendimiento del algoritmo cuando se utiliza una discretización de 32 centroides para el espacio de estados y 16 centroides para el espacio de acciones (32×16), 64 centroides para el espacio de estados y 32 centroides para el espacio de acciones (64×32), 128 centroides para el espacio de estados y 64 centroides para el espacio de acciones (128×64). Los valores de los parámetros para estos procesos de aprendizaje son: $\alpha = 0{,}3$, $\gamma = 1{,}0$. La política de exploración empleada es $\epsilon - greedy$ con $\epsilon = 0$ (completamente avariciosa) durante todo el proceso de aprendizaje.

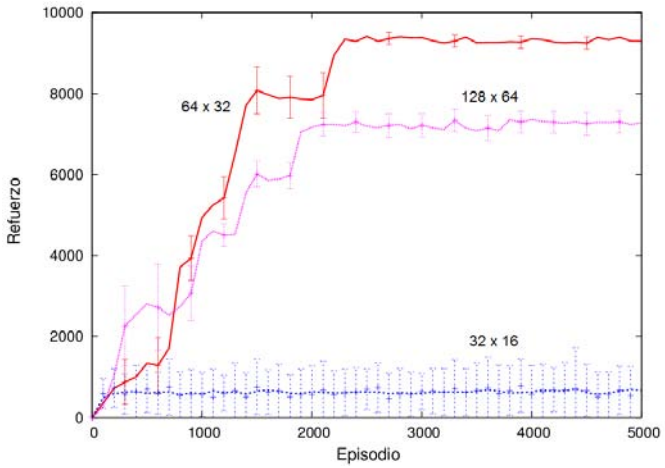


Figura 5.9: Resultados de diferentes configuraciones de G-VQQL en el *Cart-Pole*.

Con la configuración 32×16 no se logra aprender una política de comportamiento para mantener al péndulo balanceado encima del carro. En cambio, las configuraciones 64×32 y 128×64 sí consigue aprender una política de comportamiento cercana a la óptima, aunque es la configuración 64×32 la que finalmente obtiene un mayor rendimiento.

5.3.1.2. Ejecución del algoritmo CMAC-VQQL

La aplicación del primer paso del algoritmo para obtener el cuantificador vectorial asociado a las acciones descrito en la tabla 5.4, es análogo a la aplicación del primer paso del algoritmo G-VQQL descrito en la sección 5.3.1.1, salvo que para el algoritmo CMAC-VQQL sólo se construye el cuantificador vectorial asociado a las acciones. La figura 5.10 muestra el resultado de la aplicación del segundo paso del algoritmo en su versión en línea, descrito en la tabla 5.5, utilizando diferentes configuraciones de CMAC y diferentes tamaños de cuantificadores vectoriales asociados a las acciones. Igual que en el caso anterior, la versión fuera de línea del algoritmo presentada en la tabla 5.6 no produce ningún tipo de convergencia. La curva de aprendizaje etiquetada como "$5 \times 5 \times 8 \times 8$ 32 *acciones*", establece que en la configuración de CMAC implementada, el rango de la variable correspondiente a la posición del carro, x, se ha dividido en 5 intervalos regulares, al igual que el rango de la velocidad del mismo, x', mientras que los rangos correspondientes al ángulo del péndulo,ϕ, y la velocidad angular del mismo, ϕ', se han dividido en 8 intervalos regulares. Además, el cuantificador vectorial asociado a las acciones utilizado tiene un tamaño de 32. En todos los casos, se han solapado y desplazado 32 rejillas para conseguir una mayor precisión. Con lo cual el vector de pesos, θ, asociado a cada acción estará compuesto por $5 \times 5 \times 8 \times 8 \times 32\ rejillas\ = 51200$ elementos. El resto de configuraciones mostradas en la

figura 5.10 se leen de la misma manera. El resto de parámetros se establecen a: $\alpha = 0{,}125$, $\gamma = 1{,}0$. Como en el caso del algoritmo G-VQQL, la política de exploración empleada es $\epsilon - greedy$ con $\epsilon = 0$ (completamente avariciosa) durante todo el proceso de aprendizaje.

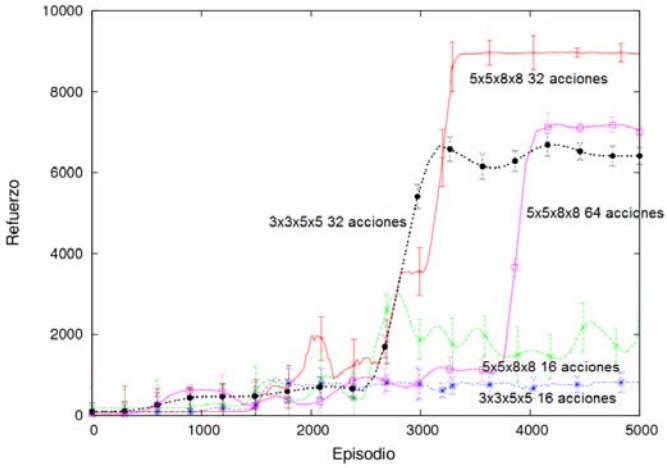


Figura 5.10: Resultados de diferentes configuraciones de CMAC-VQQL en el *Cart-Pole*.

Las configuraciones "$3 \times 3 \times 5 \times 5$ 16 *acciones*" y "$5 \times 5 \times 8 \times 8$ 16 *acciones*" incrementan ligeramente su rendimiento aunque no consiguen en ningún caso aprender una política de comportamiento que permita mantener al péndulo balanceado encima del carro. En cambio, la configuración que mejor rendimiento obtiene es la "$5 \times 5 \times 8 \times 8$ 32 *acciones*", que aprende una política de comportamiento cercana a la óptima alrededor del episodio 3200. No obstante, utilizando la misma configuración para CMAC y un conjunto discretizado de acciones, D_a, de tamaño 64, reduce la velocidad de convergencia y no obtiene una política de comportamiento como la anterior.

5.3.1.3. Comparativa del rendimiento G-VQQL y CMAC-VQQL

La figura 5.11 muestra la comparación del rendimiento del algoritmo G-VQQL y CMAC-VQQL con los algoritmos CACLA (Tabla 2.7), la aproximación basada en aprendizaje por refuerzo evolutivo (Tabla 2.9), y el algoritmo CMAC (Tabla 2.4). En la figura 5.11 (a) se muestra el rendimiento de la mejor configuración del algoritmo G-VQQL encontrada anteriormente que emplea 64 centroides para discretizar el espacio de estados y 32 centroides para el espacio de acciones, la mejor configuración del algoritmo CMAC-VQQL encontrada "$5 \times 5 \times 8 \times 8$ 32 *acciones*".

En relación al algoritmo G-VQQL, de la comparativa que se muestra en la figura 5.11 (a) se pueden extraer dos conclusiones importantes. En primer lugar, el algoritmo G-VQQL presenta una mayor velocidad de convergencia a una política cercana a la óptima, es decir, es el que consigue aprender más rápidamente a mantener el péndulo en equilibrio encima del carro. Esto es debido principalmente a que no tiene que realizar una búsqueda en la totalidad de los espacios de estados y acciones como el resto de algoritmos, sino que aborda una versión discreta del problema con tan sólo 64 estados y 32 acciones. Esta mayor velocidad en el aprendizaje de la política de comportamiento, se aprecia en mayor detalle en la figura 5.11 (b), en la que sólo se muestran los 5000 primeros episodios de la figura 5.11 (a). El siguiente algoritmo en converger es CACLA, seguido del algoritmo evolutivo. En el algoritmo CACLA, la red de neuronas empleada para aproximar la política de comportamiento

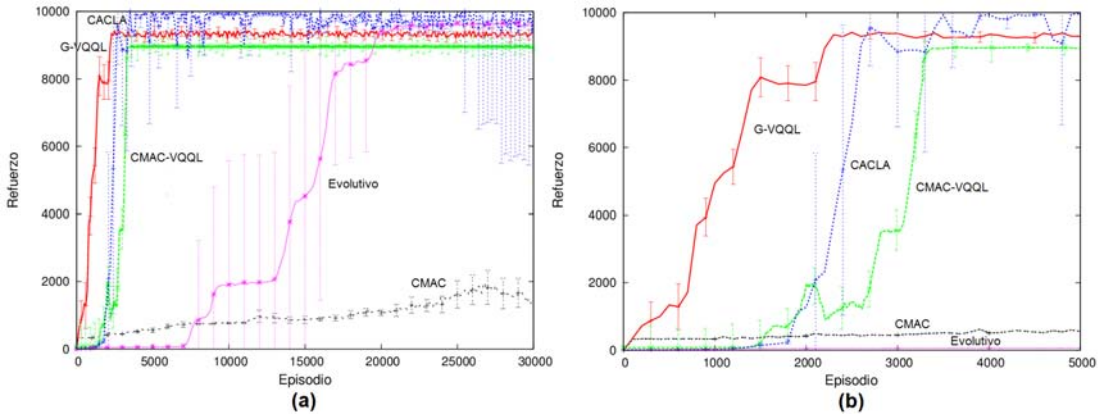


Figura 5.11: Comparación de los resultados de G-VQQL y CMAC-VQQL con diferentes algoritmos en el *Cart-Pole*.

tiene una topología $4 \times 20 \times 1$, es decir, 4 neuronas en la capa de entrada correspondientes a cada una de las características de los estados, 20 neuronas en la capa oculta, y 1 neurona en la capa de salida correspondiente a la característica de la acción, mientras que la red de neuronas empleada para aproximar la función de valor utilizada tiene la misma topología que la anterior. En el caso del algoritmo por refuerzo evolutivo, también se tratan de evolucionar redes de neuronas con topología $4 \times 20 \times 1$. Estas configuraciones se han establecido *ad-hoc*, tras probar con multitud de variaciones. El algoritmo evolutivo es el que presenta una mayor desviación, ya que presenta una mayor dependencia de la modificación aleatoria que se hacen de los pesos de las redes de neuronas que pueden propiciar la convergencia del algoritmo en episodios iniciales o en episodios más avanzados del proceso de aprendizaje. Por último, el algoritmo CMAC no logra encontrar ninguna política para mantener el péndulo en equilibrio.

En segundo lugar, el algoritmo CACLA y el algoritmo evolutivo son capaces de encontrar mejores políticas de comportamiento que el algoritmo G-VQQL. Esto es debido a que la ventaja de trabajar con un conjunto reducido y discreto de acciones, aunque se favorece la convergencia del algoritmo, es un inconveniente a la hora de encontrar mejores políticas de comportamiento. G-VQQL se encuentra limitado por el conjunto de acciones estático que utiliza para aprender, mientras que el algoritmo CACLA y la aproximación basada en aprendizaje por refuerzo evolutivo consideran un espacio de acciones continuo de forma que pueden ir "refinando" cada vez más las acciones que ejecutan.

En cuanto al algoritmo CMAC-VQQL, también tiene una rápida velocidad de convergencia, aunque en menor medida que el algoritmo G-VQQL y CACLA. El algoritmo CMAC-VQQL cuenta con un vector de pesos θ por cada acción de un gran tamaño, 51200 elementos, cuyos valores hay que ajustar adecuadamente, lo que ralentiza la velocidad de convergencia del algoritmo. Por otro lado, en la figura 5.11 (b) se puede observar más claramente como el refuerzo final que obtiene es menor que los conseguidos por G-VQQL y CACLA. Por lo tanto, CMAC-VQQL también se ve penalizado por manejar un conjunto estático de acciones, aunque en mayor medida que el algoritmo G-VQQL. Cabe destacar que CMAC-VQQL consigue un mejor resultado que la versión totalmente continua del algoritmo etiquetado como CMAC en la figura 5.11. La discretización adecuada del espacio de acciones reduce considerablemente la complejidad del problema lo que ha propiciado un mejor rendimiento con respecto a la versión totalmente continua del algoritmo.

5.3.2. *Octopus Arm*

La descripción de este dominio se encuentra detallada en la sección 4.2.4. En esta sección se presenta la aplicación del algoritmo G-VQQL (sección 5.3.2.1), la aplicación del algoritmo CMAC-VQQL (sección 5.3.2.2) y por último la comparativa de rendimiento de estos algoritmos con algunos extraídos de la literatura (sección 5.3.2.3).

5.3.2.1. Ejecución del algoritmo G-VQQL

La ejecución del algoritmo G-VQQL en el dominio del *Octopus Arm* requiere, como en el caso anterior, la ejecución de los pasos de aprendizaje de los cuantificadores vectoriales, y de aprendizaje de la política de comportamiento descritos en las secciones 5.1.1 y 5.1.2 respectivamente. En las siguientes secciones se describe cómo se aplican estos pasos en el dominio.

Primer Paso: Aprendizaje de los cuantificadores vectoriales

La ejecución del primer paso del algoritmo G-VQQL requiere a su vez de la ejecución de una serie de pasos que le permitan recoger del entorno las tuplas de experiencia con las cuales diseñar el cuantificador vectorial asociado al espacio de estados y el cuantificador vectorial asociado al espacio de acciones (Tabla 5.1). A continuación se describe la aplicación de todos estos pasos:

1. Recoger tuplas de experiencia: En primer lugar, se ha de recoger del entorno un número suficiente de tuplas de experiencia, T, que permitan construir de forma adecuada los cuantificadores vectoriales. En este dominio, debido a la alta dimensionalidad tanto del espacio de estados como del espacio de acciones, la ejecución aleatoria de acciones no permite realizar una exploración del espacio. La ejecución de acciones aleatoria produce movimientos incoherentes en el brazo, que prácticamente es incapaz de moverse del sitio de partida, y menos aún alcanzar el punto meta. La exploración aleatoria en dominios de alta dimensionalidad presentan dos problemas que se ya se describían en la sección 5.1.1:

1. El alto número de dimensiones del espacio de acciones hacen prácticamente imposible que mediante una exploración aleatoria se puedan explorar todas las posibles acciones que son válidas en el dominio.

2. La exploración aleatoria produce acciones incoherentes, que aún siendo válidas, no conducen al agente a explorar las regiones del espacio de estados más "interesantes".

Estos dos problemas causan además que sea imposible explorar de forma adecuada el espacio de estados. Por este motivo, en este caso la exploración se lleva a cabo mediante un comportamiento experto π_T definido en el apéndice B que permite explorar de forma adecuada los espacios de estados y acciones. Este comportamiento experto es capaz de completar con éxito la tarea aunque está lejos de comportarse de manera óptima. El comportamiento experto, π_T, permite solucionar los problemas que plantea la exploración aleatoria en estos dominios. Más específicamente, el comportamiento experto, π_T, permite explorar el espacio de acciones válidas y que además son coherentes, permitiendo a su vez visitar las regiones del espacio de estados más "interesantes", es decir, aquellas zonas del espacio de estados que resultan útiles para posteriormente aprender la función de valor óptima o cercana a la

óptima. La figura 5.12 muestra las regiones del espacio de estados visitadas por el comportamiento experto utilizado en el proceso de recogida de tuplas de experiencias, T. En este proceso se han recogido 42883 tuplas.

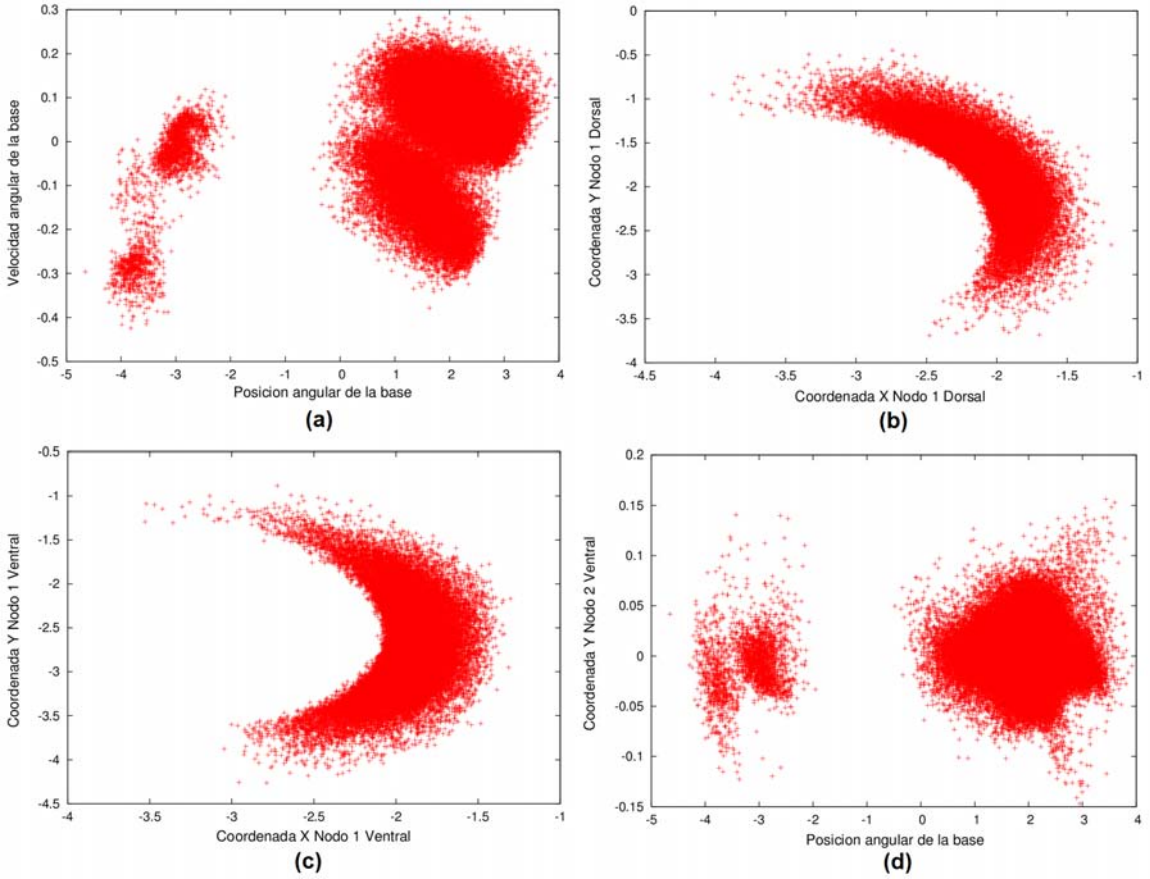


Figura 5.12: Exploración resultante del espacio de estados utilizando un comportamiento π_T experto en el dominio del *Octopus Arm*.

En la secuencia de gráficas que se muestran en la figura 5.12 se presenta en la gráfica **(a)** la posición angular de la base de brazo en el eje de abscisas, y la velocidad angular de la base en el eje de ordenadas; en la gráfica **(b)** se muestra la coordenada x del nodo 1 del brazo en la zona dorsal en el eje de abscisas, y la coordenada y del mismo nodo en el eje de ordenadas; en la gráfica **(c)** se muestra en el eje de abscisas la coordenada x del primer nodo del brazo en la zona ventral, y en el eje de ordenadas se muestra la coordenada y del mismo nodo; por último, en la gráfica **(d)** se muestra la posición angular de la base en el eje de abscisas y la coordenada y del segundo nodo en la zona ventral en el eje de ordenadas. Las gráficas se han construido a partir de los estados T_s pertenecientes a las tuplas de experiencia, T. En todas las gráficas se puede apreciar la fuerte relación estadística que existe entre las diferentes variables del espacio de estados generadas por el comportamiento experto, π_T. Estas relaciones son las que explotará el algoritmo GLA en el paso de discretización del espacio de estados.

La figura 5.13 muestra las regiones del espacio de acciones visitadas por el comportamiento experto, π_T. En la figura 5.13 (a) se muestra en el eje de abscisas la contracción que se aplica al compartimento 1 de la zona dorsal, y en el eje de ordenadas la contracción

que se aplica al compartimento 10 de la zona ventral; la figura (b) presenta en el eje de abscisas el nivel de contracción que se aplica al primer compartimento de la zona ventral y en el eje de ordenadas la contracción del décimo compartimento de la zona dorsal; en la gráfica (c) en el eje de abscisas se muestra la contracción del primer compartimento en la sección transversal, y en el eje de ordenadas la contracción del primer compartimento en la zona ventral; por último, en la gráfica (d) se muestra en el eje de abscisas la rotación en el sentido del reloj de la base del brazo y el en eje de ordenadas se muestra el nivel de contracción que se aplica al noveno compartimento de la zona ventral.

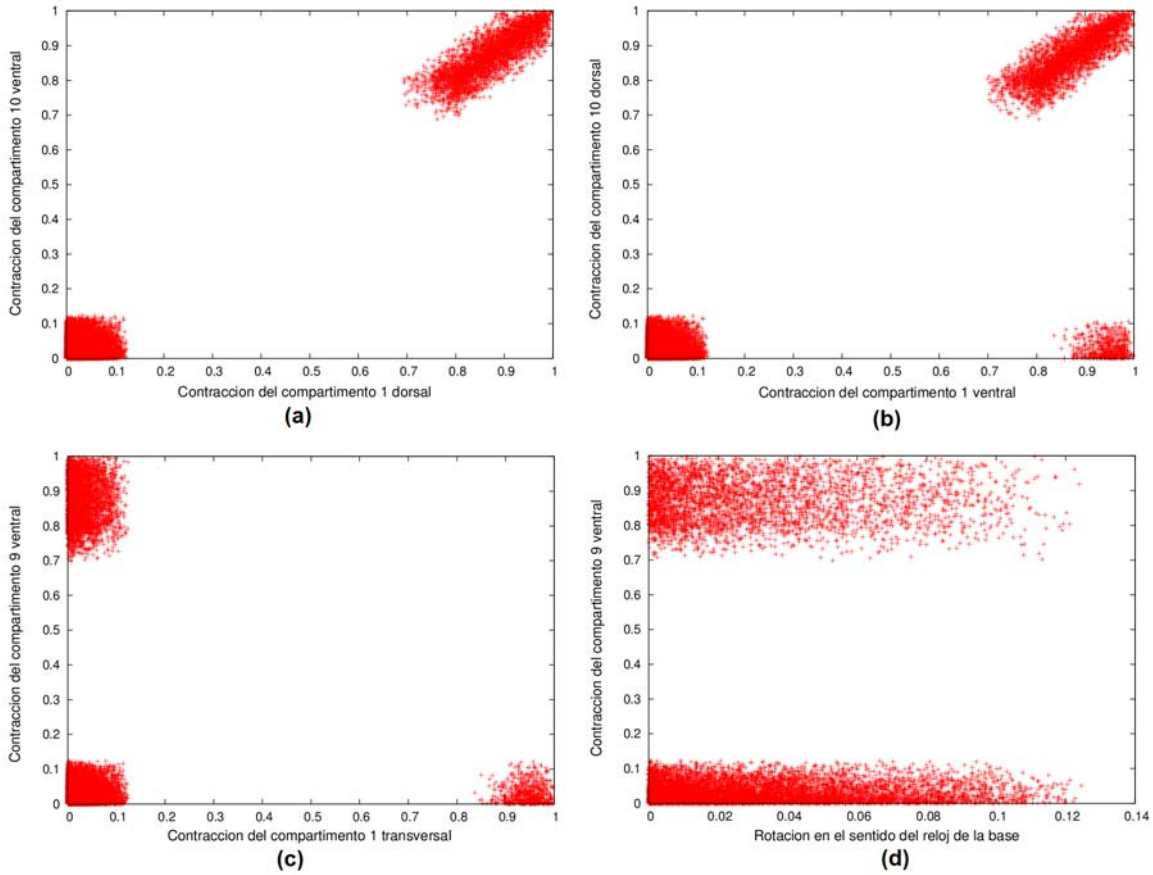


Figura 5.13: Exploración resultante del espacio de acciones utilizando un comportamiento π_T experto en el dominio del *Octopus Arm*.

A diferencia del dominio del *Cart-Pole*, las acciones no se distribuyen de forma aleatoria por el espacio de acciones, sino que se distribuyen sobre regiones bien definidas, aquellas regiones visitadas por el comportamiento π_T. Las gráficas en la figura 5.6 se han construido a partir de la acciones, T_a, recogidas en el conjunto de tuplas de experiencia T. En esta ocasión, el algoritmo GLA sí que es capaz de aprovechar las relaciones estadísticas que existen entre las características correspondientes a las acciones, para generar centroides o prototipos en aquellas regiones del espacio donde realmente son necesarios.

2. Reducir la dimensión del espacio de estados: Con el fin de reducir la dimensionalidad del espacio de estados, se aplica el evaluador *CfsSubsetEval* para establecer qué características tienen una mayor relación con la función de refuerzo del dominio. La aplicación de este evaluador se ha llevado a cabo empleando dos algoritmos de búsqueda di-

ferentes: *Best First*, y *GreedyStepWise*. La figura 5.14 muestra el número de características seleccionadas por cada nodo del brazo robótico. Como se explicó en la sección 4.2.4, cada nodo está compuesto por cuatro características diferentes relativas a la posición del nodo y a su velocidad.

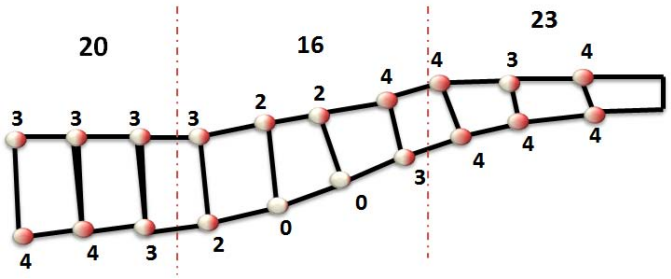


Figura 5.14: Selección de atributos en el *Octopus Arm* utilizando *CfsSubsetEval* como evaluador de subconjuntos de características con búsqueda *BestFirst* y *GreedyStepWise*.

La figura 5.14 muestra como el algoritmo selector de características empleado ha desechado por cada nodo las características menos relevantes. En la figura 5.14 también se puede apreciar como la mayoría de características seleccionadas se encuentran en el primer tercio del brazo (20 características seleccionadas), y en el último tercio (23 características), siendo este último tercio el que mayor número de características tiene. Esto es indicativo de que para dirigir adecuadamente el brazo es suficiente con controlar su parte inicial y su parte final, dejando "relajada" la parte central cuyos movimientos vendrán determinados por los movimientos de la parte inicial y final del brazo. El número de características seleccionadas es de 60, correspondientes a las 59 características seleccionadas para los nodos del brazo, más 1 característica asociada a la velocidad de la base. Por lo tanto, con la selección de características llevada a cabo, se ha reducido considerablemente la complejidad del problema, pasando de 82 características para el espacio de estados a sólo 60.

En adelante, se va a considerar al conjunto de estados T_s el conjunto de estados original sin selección de características en T. En este caso se puede suponer, por coherencia con los algoritmos mostrados en las tablas 5.1, 5.2 y 5.3, que la función Γ aplicada es la función identidad : $\Gamma(s) \rightarrow s$, y por tanto se tiene que el conjunto de estados T'_s resultante de aplicar la función Γ sobre T_s en el paso 2 de la tabla 5.1 es igual a T_s, $T'_s = \Gamma(T_s) = T_s$. En cambio, se va a considerar al conjunto de estados T'_s el conjunto de estados resultante de aplicar la función de selección de características Γ descrita en esta sección al conjunto de estados original, $T'_s = \Gamma(T_s)$. De esta forma se analizará el rendimiento del algoritmo cuando se utiliza el conjunto de estados original sin selección de características, T_s, y cuando se utiliza un conjunto al que se le ha aplicado selección de características para eliminar las características consideradas irrelevantes, T'_s.

3. Discretizar el espacio de estados: En esta sección sólo se muestra el proceso de discretización aplicado al conjunto de estados original sin selección de características, T_s, puesto que la discretización del conjunto de estados con selección de características, T'_s, se realiza de la misma manera.

El conjunto de estados original, T_s, se utiliza como entrada al algoritmo GLA. La figura 5.15 muestra cómo se distribuyen por el espacio de estados de alfabetos de tamaño 8 (Figura 5.15 (a)), 16 (Figura 5.15 (b)), 32 (Figura 5.15 (c)), 64 (Figura 5.15 (d)), 128 (Figura 5.15 (e)), y 256 centroides (Figura 5.15 (f)). En cada gráfica que se presenta en la

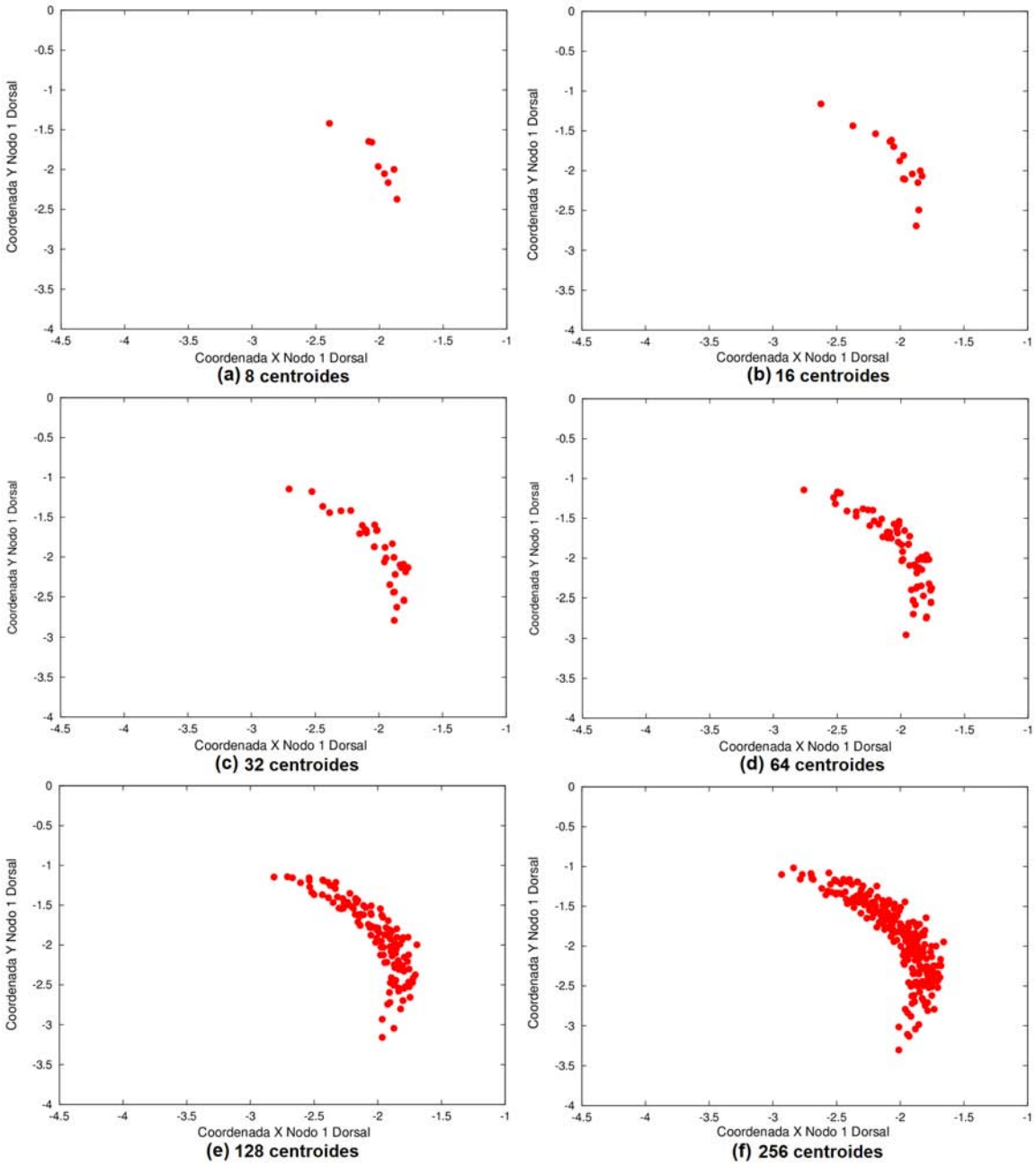


Figura 5.15: Alfabetos de diferentes tamaños obtenidos mediante GLA en el dominio del *Octopus Arm* para el conjunto de estados T_s.

figura, en el eje x se muestra la coordenada x del primer nodo del brazo en la parte dorsal, mientras que en el eje y se muestra la coordenada y de este mismo nodo. A medida que se incrementa el tamaño del alfabeto, la forma de la figura se va pareciendo cada vez más a la masa de puntos original para estas dos dimensiones representadas en la figura 5.12 (b). No obstante, en la figura 5.12 (b) hay representados 42883 puntos mientras que en la figura 5.15 (f) tan sólo hay 256 puntos. Por tanto, el algoritmo GLA sólo introduce centroides o prototipos en aquellas zonas del espacio de estados donde realmente son necesarios.

Una vez que se obtiene la discretización del espacio de estados, se diseña el cuantificador vectorial $VQ^S : S \rightarrow D_s$, encargado de asociar un estado $s \in S$ al estado más cercano dentro del conjunto de estados discretizado, D_s. El tamaño de D_s dependerá del tamaño de la discretización seleccionada.

4. Discretizar el espacio de acciones: En esta sección se aplica el algoritmo GLA sobre el conjunto de acciones, T_a, para discretizar también el espacio de acciones.

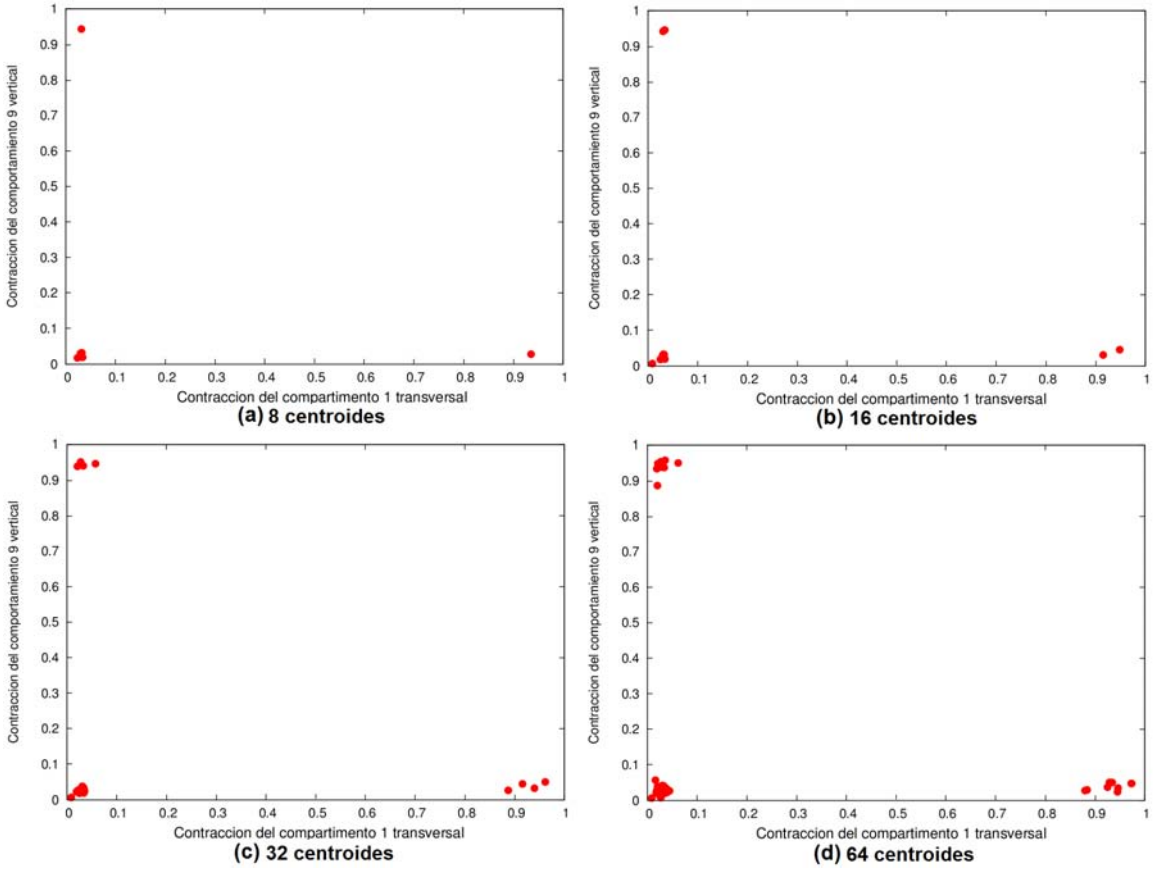


Figura 5.16: Alfabetos de diferentes tamaños obtenidos mediante GLA en el dominio del *Octopus Arm* para el conjunto de acciones T_a.

La figura 5.25 muestra alfabetos de diferentes tamaños obtenidos mediante el algoritmo GLA. Los tamaños de los alfabetos se incrementan desdoblándose desde 8 centroides (Figura 5.25 (a)) hasta los 64 centroides (Figura 5.25 (d)). En cada gráfica se muestra en el eje x la contracción transversal del primer del compartimento, mientras que en el eje y se muestra la contracción vertical del noveno compartimento . En la secuencia de gráficas se puede apreciar cómo los centroides conservan la relación estadística que existe entre ambas variables, y que también era apreciable en el conjunto de acciones original recogidas mediante el comportamiento base (figura 5.13 (c)). Por lo tanto, esta relación estadística se mantendrá para estas variables durante todo el proceso de aprendizaje evitando explorar zonas del espacio donde no exista. De esta forma, se mantiene la coherencia entre los valores de las diferentes variables de una acción dada por el comportamiento base π_T. Los alfabetos obtenidos se utilizan para construir el conjunto discreto de acciones, D_a, que es empleado para aprender la política de comportamiento óptima en la versión en línea del algoritmo

(Tabla 5.2). En la versión fuera de línea del algoritmo (Tabla 5.3), a partir de estos alfabetos se diseña el cuantificador vectorial $VQ^A : A \rightarrow D_a$, que asocia una acción $a \in A$ a una de las acciones del conjunto discreto de acciones D_a.

Segundo Paso: Aprendizaje de la Política de Comportamiento

Una vez que se diseñan los cuantificadores vectoriales tanto para el espacio de estados como para el espacio de acciones se aprende la política de comportamiento. La figura 5.17 (a) muestra la ejecución de la versión en línea del algoritmo mostrada en la tabla 5.2 para diferentes tamaños de discretizaciones para los estados y acciones. Los valores de los parámetros de aprendizaje son: $\alpha = 0{,}2$, $\gamma = 1{,}0$ y $\epsilon = 0$ (estrategia de exploración completamente avariciosa) durante todo el proceso de aprendizaje. La curva de aprendizaje etiquetada como 64×16 indica que se ha empleado 64 centroides para representar el espacio de estados y 16 centroides para representar el espacio de acciones. La figura 5.17 (a) muestra los resultados cuando se utiliza la función Γ identidad $\Gamma(s) \rightarrow s$, es decir, sin selección de características. En la gráfica se observa que con 16 centroides para las acciones la velocidad de convergencia del algoritmo es muy alta. Con un número mayor de centroides para las acciones la velocidad de convergencia es menor aunque se mejora la calidad de la política aprendida. Esto es un comportamiento esperable puesto que al aumentar la resolución, se aumenta el tamaño de las tablas por lo que hay que estimar más parámetros y se tarda más. Por otro lado, al haber más resolución el grano es más fino y se ajustan más las políticas de comportamiento. Además, en todos los casos se consigue mejorar el rendimiento de la política de comportamiento π_T empleada para explorar los espacios de estados y acciones. La figura 5.17 (b) muestra la ejecución fuera de línea del algoritmo G-VQQL mostrado en la tabla 5.3 para la configuración 128×16. El aprendizaje en la versión fuera de línea del algoritmo se realiza empleando el mismo conjunto de estados T_s recogido en la fase de exploración anterior, y que también ha sido utilizado para construir los cuantificadores vectoriales. En este caso, se consigue mejorar el rendimiento del comportamiento base π_T, pero no se consigue mejorar las políticas obtenidas con la version en línea del algoritmo.

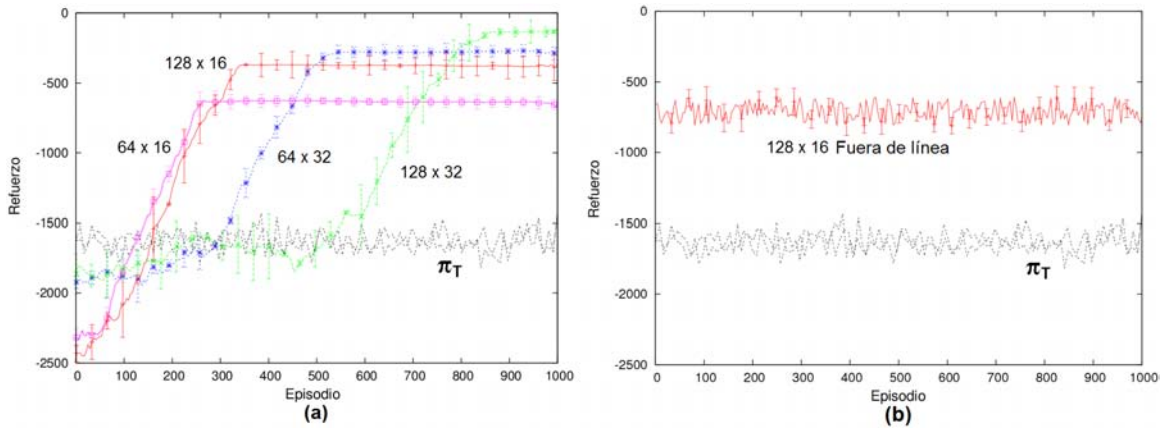


Figura 5.17: (a) Resultados de diferentes configuraciones de G-VQQL versión en línea en el *Octopus Arm* (b) Resultado de G-VQQL versión fuera de línea en el *Octopus Arm*

La figura 5.18 muestra los resultados de las configuraciones 64×16 (Figura 5.18 (a)) y 128×16 (Figura 5.18 (b)) cuando se emplea la función Γ de selección de características

descrita en la figura 5.14. Las curvas de aprendizaje en las que se ha aplicado esta selección de características se encuentran etiquetas como 64×16 *con selección* y 128×16 *con selección*, mientras que las curvas etiquetadas como 64×16 y 128×16 se corresponden con las curvas mostradas en la figura 5.17 (a) sin selección de características. En ambos casos se aprecia cómo la aplicación de la selección de características propuesta mejora la calidad de las políticas obtenidas, siendo la mejora más significativa en el caso de la configuración 64×16 *con selección*.

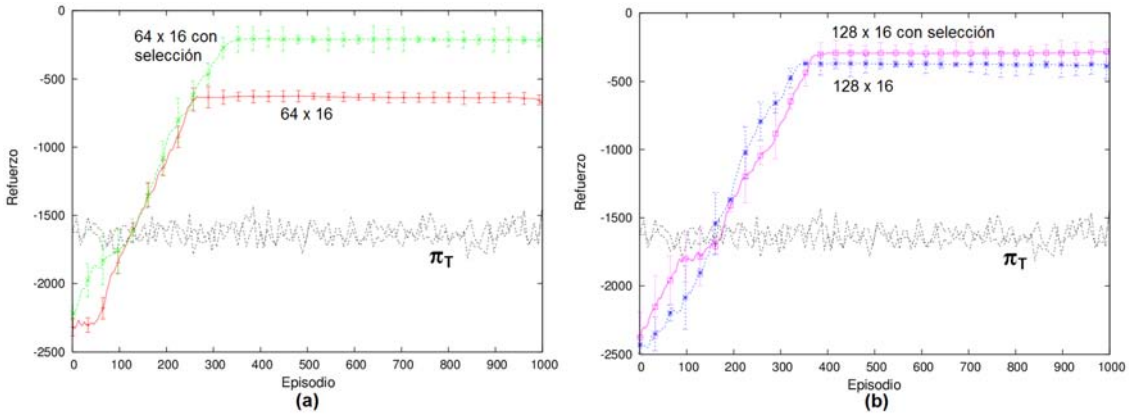


Figura 5.18: Resultados de diferentes configuraciones de G-VQQL versión en línea en el *Octopus Arm* con selección de características.

5.3.2.2. Ejecución del algoritmo CMAC-VQQL

La obtención del cuantificador vectorial asociado a las acciones se construye con la aplicación del primer paso del algoritmo CMAC-VQQL mostrado en la tabla 5.4. Este proceso es el mismo que el llevado a cabo en el primer paso del algoritmo G-VQQL descrito en la sección anterior. La figura 5.19 (a) muestra los resultados del algoritmo CMAC-VQQL en su versión en línea para diferentes tamaños del vector de pesos θ y discretizaciones del espacio de acciones. Los valores de los parámetros de aprendizaje en este caso son: $\alpha = 0{,}2$, $\gamma = 1{,}0$ y $\epsilon = 0$ (estrategia de exploración completamente avariciosa) durante todo el proceso de aprendizaje. La curva de aprendizaje etiquetada como 5610×32 *acciones* indica en primer lugar la suma de los intervalos regulares en los que se ha dividido cada una de las 82 características que componen los estados en el dominio (i.e., 5610 intervalos), y por último el tamaño de la discretización del espacio de acciones empleada (i.e., 32 acciones). Además, en todas las configuraciones CMAC empleadas en los experimentos se han solapado y desplazado 32 rejillas para obtener una mayor precisión en el proceso de generalización de los estados. Por lo tanto el vector de pesos θ contiene por cada acción 5610×32 $rejillas$ $=$ 179520 elementos. En el caso de las curvas de aprendizaje en las que la suma de los intervalos regulares se incrementa hasta 28050, el vector de pesos θ contiene por cada acción 28050×32 $rejillas$ $=$ 897600 elementos.

La figura 5.19 (a) muestra como las configuraciones 5610×16 *acciones* y 5610×32 *acciones* tienen una mayor velocidad de convergencia y obtienen mejores políticas de comportamiento que las configuraciones 28050×16 *acciones* y 28050×32 *acciones*. En este último caso además, apenas se consigue igualar el rendimiento del comportamiento base π_T. Por lo tanto, aunque un mayor número de elementos en el vector de pesos θ implica

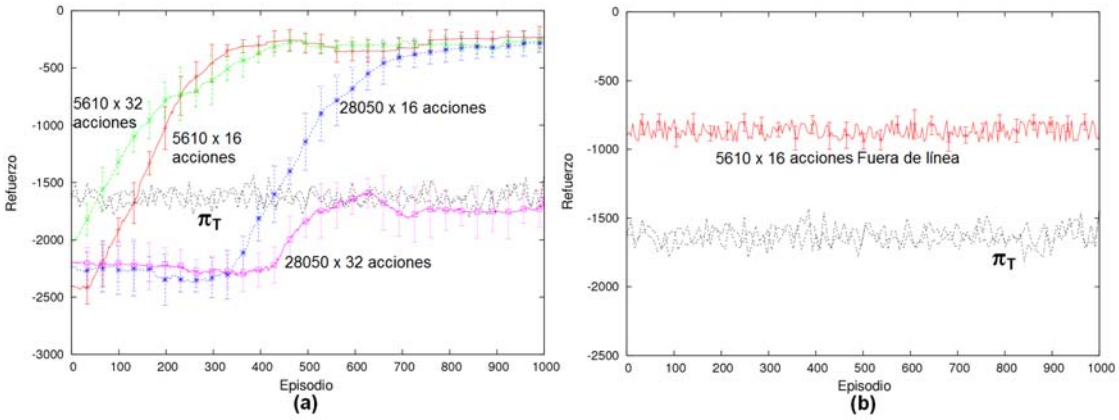


Figura 5.19: (a) Resultados de diferentes configuraciones de CMAC-VQQL versión en línea en el *Octopus Arm* (b) Resultado de CMAC-VQQL versión fuera de línea en el *Octopus Arm*.

una mayor precisión en la generalización de los estados, también implica una mayor cantidad de elementos que deben ser ajustados adecuadamente, lo que ralentiza la velocidad de convergencia e impide, incluso, el aprendizaje de mejores políticas de comportamiento. La figura 5.19 (b) muestra el rendimiento del algoritmo CMAC-VQQL en su versión fuera de línea (tabla 5.6) para una de las mejores configuraciones obtenidas con la versión en línea, la configuración 5610 $\times$ 16 *acciones*. En este caso, con la versión fuera de línea también se consigue mejorar el rendimiento del comportamiento base, aunque como en el caso del algoritmo G-VQQL, no se consigue incrementar el rendimiento de las políticas obtenidas con la versión en línea del algoritmo.

La figura 5.20 muestra la mejora que se produce cuando se emplea la función Γ de selección de características descrita en la figura 5.14.

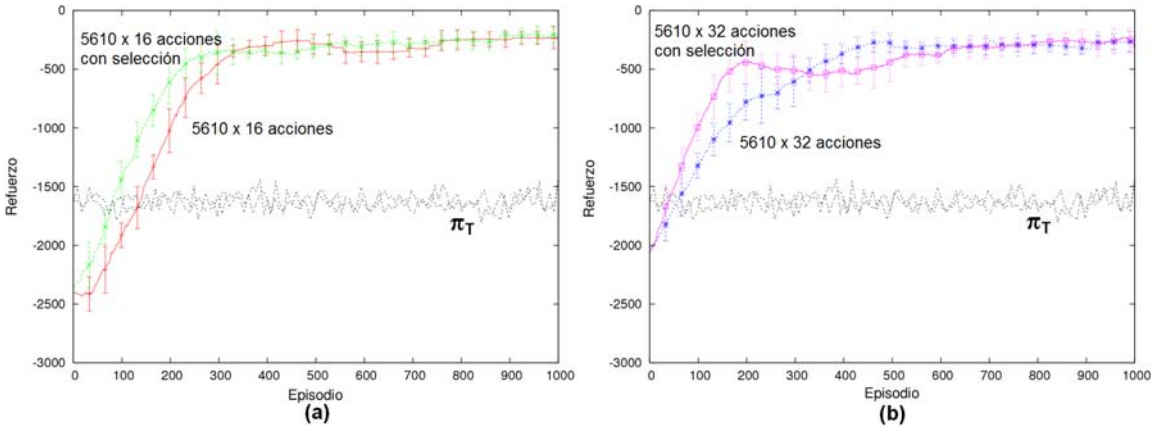


Figura 5.20: Resultados de diferentes configuraciones de CMAC-VQQL versión en línea en el *Octopus Arm* con selección de características.

La figura 5.20 (a) muestra la mejora que se produce cuando se emplea la selección de características propuesta con la configuración 5610$\times$16 *acciones*, mientras que la figura 5.20 (b) muestra la mejora que se produce con la configuración 5610 $\times$ 32 *acciones*. En ambos casos se aprecia como se incrementa la velocidad de convergencia del algoritmo, puesto que

con la selección de características se reduce la dimensionalidad del problema, reduciendo el tamaño del vector de pesos θ, lo que implica, a su vez, un menor número de parámetros que deben ser ajustados adecuadamente.

5.3.2.3. Comparativa del rendimiento G-VQQL y CMAC-VQQL

La figura 5.21 muestra la comparación del rendimiento de las mejores configuraciones obtenidas para los algoritmos G-VQQL (64×16 *acciones* con selección) y CMAC-VQQL (5610×16 *acciones* con selección) con un algoritmo que emplea aprendizaje por refuerzo evolutivo (Tabla 2.9), con el algoritmo CACLA basado en arquitecturas de tipo actor-crítico (Tabla 2.7), y con el algoritmo CMAC que trata de aproximar la función de valor acción mediante CMAC abordando directamente el problema con unos espacios de estados y acciones continuos (Tabla 2.4).

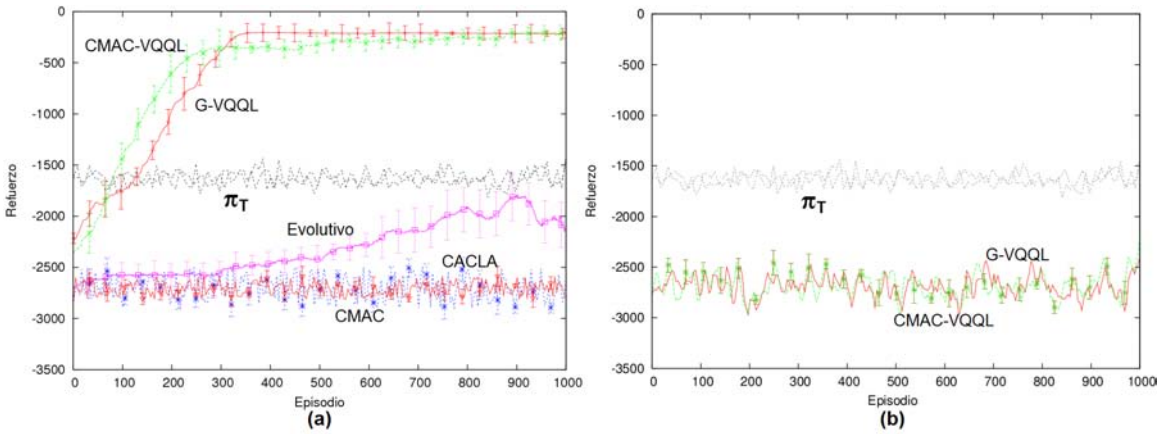


Figura 5.21: (a) Comparación de los resultados de G-VQQL y CMAC-VQQL con diferentes algoritmos en el *Octopus Arm* cuando se utiliza el comportamiento base π_T en el paso de recogida de experiencia (Tablas 5.1 y 5.4) (b) Comparación de los resultados de G-VQQL y CMAC-VQQL con diferentes algoritmos en el *Octopus Arm* cuando se utiliza un comportamiento aleatorio en el paso de recogida de experiencia (Tablas 5.1 y 5.4).

En la figura 5.21 (a), se muestra el rendimiento de los algoritmos G-VQQL y CMAC-VQQL cuando se emplea el comportamiento base π_T en el paso de recogida de experiencia de los algoritmos (Tablas 5.1 y 5.4). En cambio, la figura 5.21 (b) muestra el rendimiento de los algoritmos G-VQQL y CMAC-VQQL cuando en el paso de recogida de experiencia no se ha empleado el comportamiento base π_T, sino un comportamiento aleatorio. En la comparación que se muestra en la figura 5.21 (a) destacan dos hechos fundamentales. Por un lado, tanto el algoritmo G-VQQL como el algoritmo CMAC-VQQL comienzan sus procesos de aprendizaje obteniendo un refuerzo de alrededor de -2200, mientras que el resto de algoritmos comienzan con un refuerzo menor de alrededor de -2700. Esto se debe a que a los algoritmos G-VQQL y CMAC-VQQL han recibido mediante el comportamiento π_T un conjunto de acciones válidas y coherentes, por lo que no tienen que gastar el tiempo explorando los inmensos espacios de estados y acciones continuos de este dominio como hacen el resto de técnicas. En segundo lugar, debido al primero, los algoritmos G-VQQL y CMAC-VQQL consiguen una rápida convergencia y una política de comportamiento cercana a la óptima con un refuerzo de aproximadamente -200. En la figura 5.21 (b), donde se ha empleado un comportamiento aleatorio en el paso de recogida de experiencia, los algoritmos G-VQQL y CMAC-VQQL

no logran alcanzar ningún tipo de convergencia. Este hecho demuestra que la exploración aleatoria en este dominio produce una exploración inadecuada del espacio que repercute negativamente en la calidad de las discretizaciones que posteriormente se construyen para aprender una política de comportamiento correcta.

Mientras, el resto de algoritmos no son capaces de obtener una política de comportamiento que supere al comportamiento base π_T. En el caso del algoritmo Evolutivo, se están tratando de evolucionar redes de neuronas de gran complejidad con una topología $82 \times 165 \times 32$ (i.e., con 82 neuronas en la capa de entrada, 165 neuronas en la capa oculta y 32 neuronas en la capa de salida) lo que dificulta encontrar de forma aleatoria en todo el espacio de pesos una red de neuronas con un comportamiento adecuado. Aunque se ha experimentado con un número muy elevado de topologías *ad-hoc*, en la figura 5.21 sólo se muestran los resultados de la topología con 165 neuronas en la capa oculta, número establecido siguiendo el Teorema de Kolmogorov donde se enuncia que una red de neuronas con N neuronas en la capa de entrada, y con $2N + 1$ neuronas en la capa oculta, resulta ser un aproximador universal de funciones continuas bajo las condiciones de Kolmogorov [Hecht-Nielsen, 1987]. No obstante, en la figura 5.21 se muestra como el algoritmo Evolutivo logra aprender un comportamiento ligeramente inferior al comportamiento base π_T, aunque la velocidad de convergencia es muy lenta. En cuanto al algoritmo CACLA, la dificultad se incrementa puesto que se trata de aproximar la función de valor mediante una red de neuronas con una topología $82 \times 165 \times 1$ a la vez que se aproxima una red de neuronas con la política de comportamiento con una topología $82 \times 165 \times 32$, lo que hace que no se alcance ningún tipo de convergencia. Por último, el algoritmo CMAC también divide en intervalos regulares las 32 características asociadas a las acciones al mismo tiempo que se solapan y se desplazan 32 rejillas para alcanzar una mayor precisión. Esto hace que el vector de pesos θ incremente enormemente el número de elementos con respecto a las configuraciones utilizadas por el algoritmo CMAC-VQQL, lo que impide la convergencia del algoritmo.

Además, se han analizado los tiempos de entrenamiento empleados en las configuraciones de los algoritmos que se muestran en la figura 5.21 (a), donde tanto el algoritmo G-VQQL como CMAC-VQQL emplean el comportamiento base, π_T, en el paso de recogida de experiencia. La tabla 5.8 muestra, por cada algoritmo, la duración media en minutos de la ejecución de 1000 episodios de entrenamiento, el minuto en torno al cual el algoritmo converge y el tiempo medio en milisegundos que utiliza en cada paso de un episodio. Así, por ejemplo, el algoritmo G-VQQL emplea una media de 57.1 minutos en la ejecución de los 1000 episodios, produciéndose la convergencia en torno al minuto 39.7, con una duración media en cada paso de un episodio de 6.83 milisegundos.

Por lo tanto, el algoritmo G-VQQL es el más rápido en aprender una política de comportamiento adecuada y es también el que menos tiempo consume en ejecutar cada paso de aprendizaje. Le sigue el algoritmo CMAC-VQQL puesto que logra la convergencia en el minuto 73.35 con una duración total en todo el proceso de entrenamiento de 109.21 minutos. No obstante, la gran cantidad de parámetros empleada en el algoritmo CMAC-VQQL hace que la duración media de un paso se incremente hasta los 12.11 milisegundos. Este hecho también se hace evidente en el algoritmo CMAC, donde no sólo se utilizan rejillas divididas en intervalos y solapadas para las características de los estados, sino también para las características de las acciones, lo que incrementa aún más el número de parámetros, necesitando 17.83 milisegundos por paso. Como el algoritmo CMAC, el algoritmo evolutivo y el algoritmo CACLA no consiguen la convergencia a políticas de comportamiento adecuadas, aunque en este caso la duración media de cada paso de aprendizaje es menor (7.1 milisegundos y 9.41 milisegundos respectivamente). En el caso del algoritmo evolutivo, este

	Duración (min.)	Convergencia (min.)	Paso (miliseg.)
G-VQQL	57.1	39.7	6.83
CMAC-VQQL	109.21	73.35	12.11
CMAC	694.18	-	17.83
Evolutivo	229.93	-	7.1
CACLA	348.13	-	9.41

Tabla 5.8: Duración media en minutos que emplea cada algoritmo en la ejecución de 1000 episodios de entrenamiento, el minuto en torno al cual el algoritmo converge y la duración media de cada paso de un episodio en el dominio del *Octopus Arm*. El símbolo "-" indica que el algoritmo no ha conseguido la convergencia a una política de comportamiento adecuada. Los experimentos se han llevado a cabo en un ordenador con un procesador *quadcore* de 2.4 GHz con 4GB de RAM sobre Linux.

hecho se debe a que los pasos que más duran son aquellos en los cuales se evoluciona la población, circunstancia que ocurre una vez que se han evaluado todos los individuos de la misma. En el resto únicamente se presenta al individuo correspondiente el estado recibido del entorno y se calcula la salida. En el caso del algoritmo CACLA, cada paso conlleva la actualización de los pesos de dos redes de neuronas de gran complejidad, lo que hace que se incremente el tiempo de ejecución de cada paso. Aún así, el número de parámetros es más reducido que los requeridos por los algoritmos CMAC-VQQL y CMAC.

5.3.3. SIMBA

La descripción del simulador empresarial SIMBA se encuentra detallada en la sección 4.2.5. En esta sección se describe en primer lugar la aplicación del algoritmo G-VQQL en el dominio del simulador empresarial SIMBA (sección 5.3.3.1). Posteriormente se describe la aplicación del algoritmo CMAC-VQQL (sección 5.3.3.2). Por último, se muestra la comparativa de rendimiento de los algoritmos propuestos con diferentes algoritmos extraídos de la literatura (sección 5.3.3.3).

5.3.3.1. Ejecución del algoritmo G-VQQL

La ejecución del algoritmo G-VQQL en el simulador empresarial SIMBA requiere llevar a cabo los pasos descritos en las secciones 5.1.1 y 5.1.2 cuya aplicación, se describe detalladamente a continuación.

Primer Paso: Aprendizaje de los cuantificadores vectoriales

La construcción de los cuantificadores vectoriales se lleva a cabo realizando los pasos descritos en la tabla 5.1. Estos pasos aplicados al dominio del simulador empresarial SIMBA son:

1. Recoger tuplas de experiencia: La recogida de tuplas de experiencia en este dominio se realiza mediante la utilización de un comportamiento experto, π_T, como ya se hizo en el dominio del *Octupus* descrito en la sección anterior. La toma de acciones aleatoria en este dominio provoca que el agente caiga continuamente en bancarrotas lo que le impide explorar de forma adecuada el espacio de estados. Es decir, la toma de acciones aleatoria permite explorar sólo una región del espacio de estados relacionada con

bancarrotas, y que no es interesante para el posterior proceso de aprendizaje, puesto que en esta región del espacio no se encuentra la política óptima o políticas cercanas a la óptima. El comportamiento experto π_T toma acciones válidas y coherentes lo que permite explorar el espacio de estados de forma más eficiente que un comportamiento aleatorio sin caer continuamente en bancarrotas.

La figura 5.22 muestra cómo se distribuyen los estados visitados por el comportamiento experto, π_T, en el dominio del simulador empresarial SIMBA. En la gráfica 5.22 (a) se muestra en el eje x el margen operativo y en el eje y el esfuerzo publicitario en ventas. En cambio, en la gráfica 5.22 (b) se muestra en el eje x el valor contable y en el eje y la estimación de la publicidad. En ambas gráficas se puede apreciar la fuerte relación estadísticas que existe entre las diferentes variables de estado. Estas relaciones serán aprovechadas por el algoritmo GLA para incluir centroides o prototipos sólo en aquellas zonas del espacio de estados donde son necesarios.

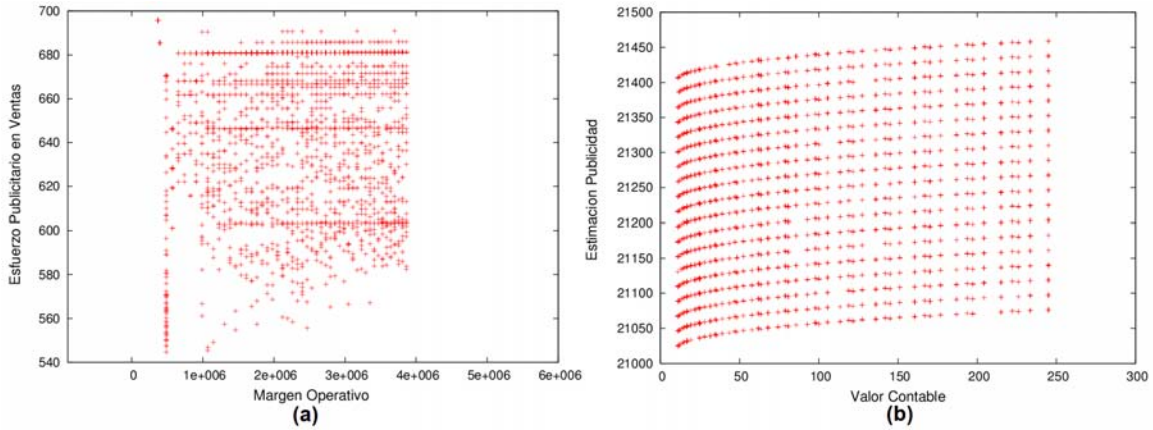


Figura 5.22: Exploración resultante del espacio de estados utilizando un comportamiento π_T experto en SIMBA.

La figura 5.23 muestra las regiones del espacio de acciones que se han explorado mediante el comportamiento experto, π_T. La gráfica 5.23 (a) muestra la relación que existe entre el precio del producto que se establece, y el presupuesto que se dedica a la publicidad, mientras que la gráfica 5.23 (b) muestra la relación que existe entre la compra de materias primas y el presupuesto dedicado a formación. De nuevo, las gráficas indican que también existe una fuerte relación estadística entre las variables de las acciones. Además, la gráfica 5.23 (a) indica que cuando se incrementa el precio del producto se dedica más presupuesto en la publicidad la empresa. Este tipo de relaciones que existen entre éstas y otras variables no se pierden durante el posterior proceso de discretización del espacio de estados.

De esta forma, en la discretización resultante se mantienen las relaciones que existen entre las variables de acción ejecutadas mediante el comportamiento experto, π_T, lo que asegura que se estarán tomando decisiones válidas y coherentes.

2. Reducir la dimensión del espacio de estados: En este paso se ejecuta el evaluador de atributos *CfsSubsetEval* implementado en la herramienta WEKA, utilizando los métodos de búsqueda *BestFirst* y *GreedyStepWise* [Witten and Frank, 2005]. Ambos métodos de búsqueda seleccionan las 12 características para los estados que se muestran en la tabla 5.9.

De esta forma, la función $\Gamma(s) \rightarrow s'$ proyecta el estado s de 170 características a un estado s' de tan sólo 12, reduciendo considerablemente la complejidad del problema. Las

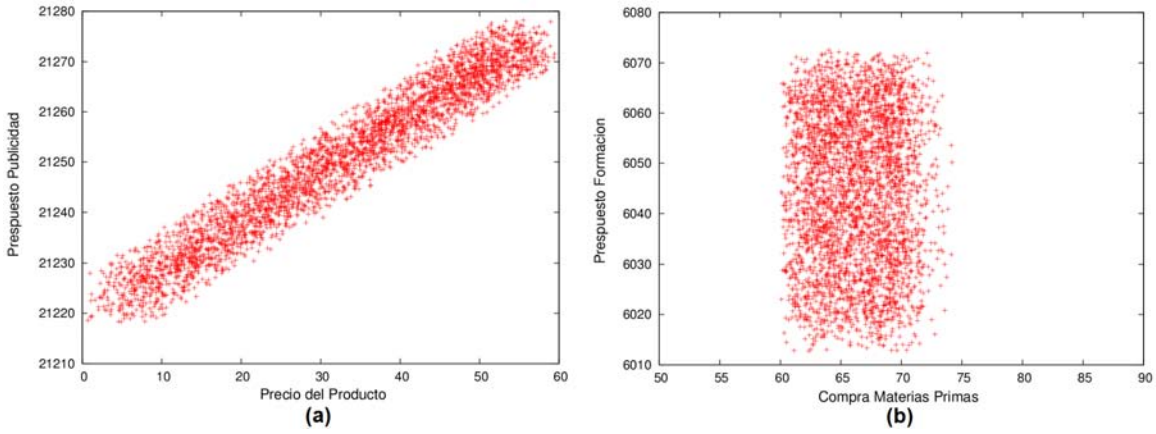


Figura 5.23: Exploración resultante del espacio de acciones utilizando un comportamiento π_T experto en SIMBA.

Características de los estados	**Características de las acciones**
Valor Contable	Precio de venta
Recursos humanos	Presupuesto en publicidad
Coste materias primas	Presupuesto en la red de ventas
Margen Operativo	Información comercial
Gastos financieros	Presupuesto en formación
Resultado antes de impuestos	Producción programada
Impuestos	Compra de materias primas
Gastos en formación	Presupuesto en I+D
Descubierto en bancos	Préstamo
Producción económica	Periodo de préstamo
Estimación publicidad	
Esfuerzo publicitario de la red de ventas	

Tabla 5.9: Subconjunto de características del espacio de estados y de acciones consideradas en SIMBA.

características seleccionadas son aquellas que mayor relación tienen con la función de refuerzo, es decir, para este dominio, son aquellas que están directamente relacionadas con los beneficios que obtiene la compañía. Esta selección tiene sentido puesto que las 170 características de los estados ofrecen información económica muy variada referente a las distintas áreas funcionales de la compañía (recursos humanos, operaciones, finanzas, marketing, ...), con sólo unas pocas características relacionadas directamente con los beneficios que se obtienen. Cabe destacar el hecho de que la mayor parte de las características seleccionadas por los algoritmos se enmarcan dentro del área financiera de la compañía.

Además, dada la complejidad del dominio, también se ha reducido el número de características del espacio de acciones, pasando de 25 características a las 10 características mostradas en la tabla 5.9. Esta reducción del espacio de acciones ha sido aconsejada por expertos en economía que consideran que las características descartadas tienen una repercusión menor en los beneficios finales que obtiene la compañía.

3. Discretizar el espacio de estados: En este paso, el algoritmo GLA recibe co-

mo entrada el conjunto de estados al que previamente se le ha aplicado la selección de características descrita en el paso anterior, T'_s. La figura 5.24 presenta la distribución de los centroides o prototipos para alfabetos de 32 (figura 5.24 (a)), 64 (figura 5.24 (b)), 128 (figura 5.24 (c)) y 256 centroides (figura 5.24 (d)). En el eje x de cada gráfica se muestra el margen operativo, mientras que en el eje y se muestra el esfuerzo publicitario en ventas. Como en los anteriores dominios, los centroides sólo se distribuyen por aquellas zonas del espacio de estados donde realmente son necesarios, es decir, sobre aquellas zonas del espacio de estados que han sido visitadas con el comportamiento base π_T en la fase de exploración. De hecho, se puede apreciar cómo a medida que se incrementan el número de centroides empleados, la masa de puntos adquiere cada vez en mayor medida la forma de la masa de puntos original presentada en la figura 5.22 (a).

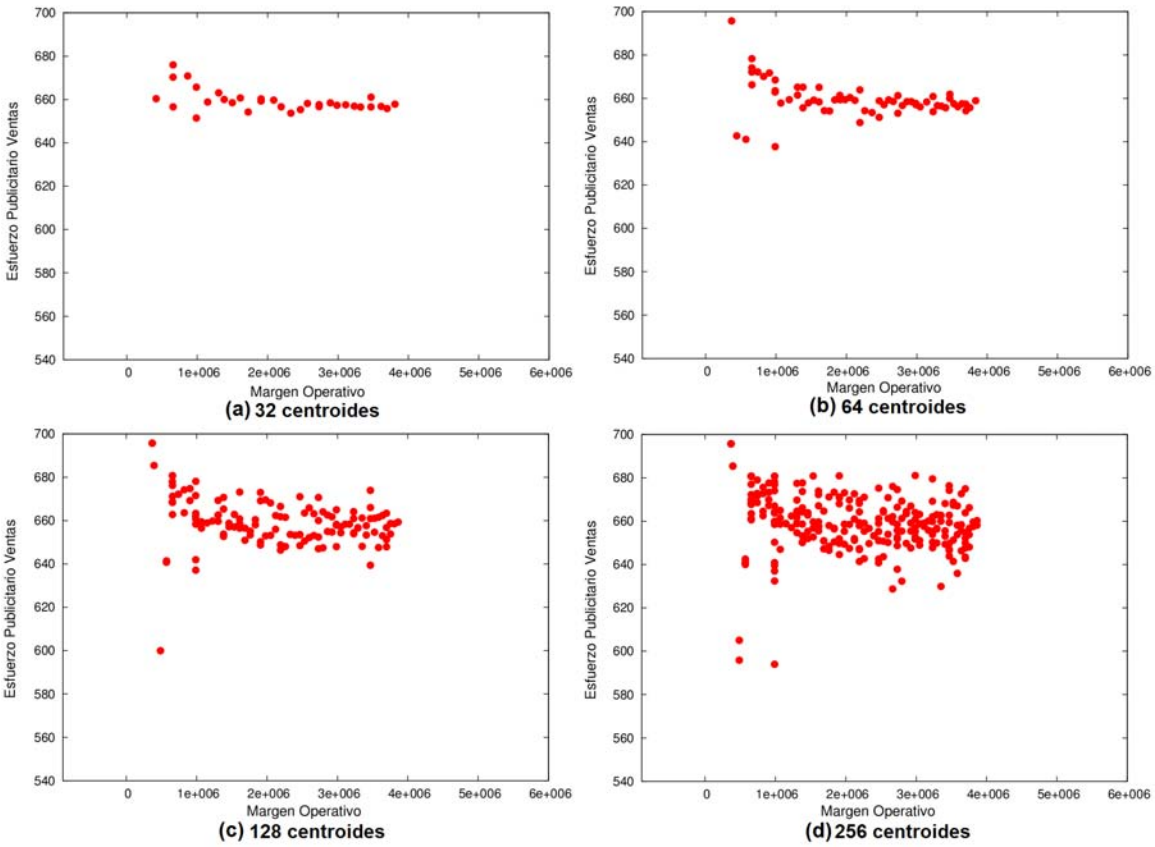


Figura 5.24: Alfabetos de diferentes tamaños obtenidos mediante GLA en el dominio del simulador empresarial SIMBA para el conjunto de estados T'_s.

El cuantificador vectorial correspondiente, $VQ^S : S \rightarrow D_s$, se diseña a partir de una de las discretizaciones obtenidas, y se encargará de asociar mediante distancia Euclídea un nuevo estado $s \in S$ al estado más cercano en el conjunto de estados discretizados D_s.

4. Discretizar el espacio de acciones: En este caso, se proporciona al algoritmo GLA el conjunto de acciones T_a recogidas del entorno mediante la ejecución del comportamiento base, π_T, con el fin de obtener discretizaciones de diferentes tamaños para el espacio de acciones. La figura 5.25 muestra diferentes tamaños de discretizaciones que van desde los 8 centroides (figura 5.25 (a)) hasta los 64 (figura 5.25 (d)). En cada gráfica de la figura, se presenta en el eje x el precio del producto, y en el eje y el presupuesto destinado a

publicidad. En la secuencia de gráficas se aprecia como a medida que se incrementa el número de centroides la masa de puntos adquiere la forma de la masa de puntos de la figura 5.23 (a). Además, se puede apreciar cómo se conserva la relación estadística que existe entre el precio del producto y el presupuesto en publicidad, de forma que también en el espacio de acciones discretizado, cuanto mayor es el precio del producto mayor es el presupuesto destinado a publicidad. De esta forma, durante el proceso de exploración se evitan explorar zonas del espacio donde no exista esta relación, de la que sabemos que es coherente puesto que viene determinada por el comportamiento base utilizado, y que podría ser incluso obligatoria para alcanzar una política de comportamiento óptima.

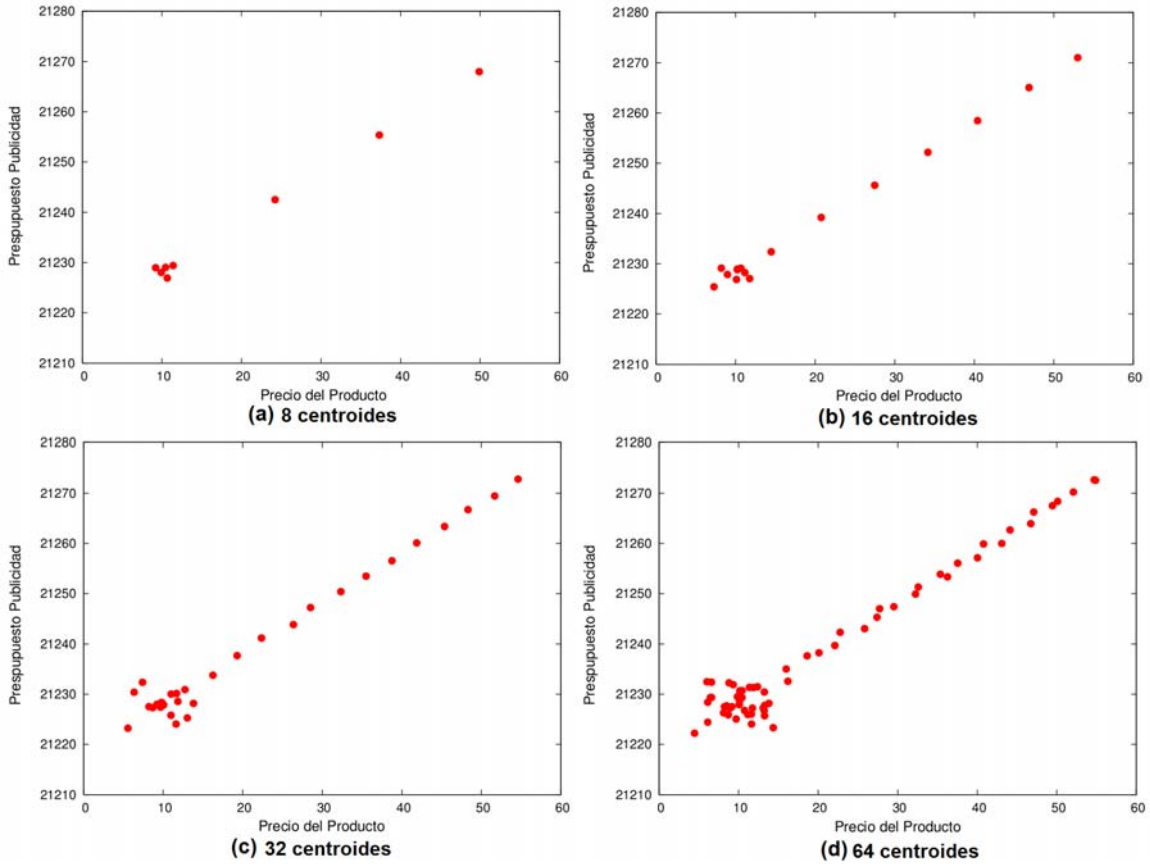


Figura 5.25: Alfabetos de diferentes tamaños obtenidos mediante GLA en el dominio en el dominio del simulador empresarial SIMBA para el conjunto de acciones T_a.

En la versión en línea del algoritmo (Tabla 5.2), las discretizaciones obtenidas se utilizan para construir el conjunto discreto de acciones D_a empleado para aprender la política de comportamiento óptima. En la versión fuera de línea del algoritmo (Tabla 5.3), a partir de estos alfabetos se diseña el cuantificador vectorial $VQ^A : A \rightarrow D_a$, que asocia una acción $a \in A$ a la acción más cercana del conjunto discreto de acciones D_a.

Segundo Paso: Aprendizaje de la Política de Comportamiento

Una vez que se han construido los cuantificadores vectoriales para el espacio de estados y de acciones a partir del comportamiento base, π_T, se aprende la política de comportamiento

mediante Q-Learning. La figura 5.26 muestra los resultados del algoritmo G-VQQL en su versión en línea y fuera de línea para diferentes tamaños de discretizaciones del espacio de estados y de acciones. En los experimentos llevados a cabo, en el algoritmo Q-Learning la tasa de aprendizaje, α se ha establecido a 0.125 y el factor de descuento, γ, a 0.9. Se ha utilizado una estrategia de exploración $\epsilon - greedy$ reduciendo el valor de ϵ de 1 (comportamiento aleatorio) hasta 0 (comportamiento completamente avaricioso) en 0.015 por cada episodio o simulación. La curva etiquetada como 64×32 indica que el aprendizaje se lleva a cabo con una discretización de tamaño 64 para el espacio de estados y una discretización de tamaño 32 para el espacio de acciones. En ambas gráficas, además, se muestra cuál es el rendimiento medio del comportamiento base π_T utilizado para explorar el espacio de estados y de acciones que se mantiene en torno a los 8.4 millones de euros de beneficios. En cuanto a G-VQQL en su versión en línea (figura 5.26 (a)), con la configuración 32×16 no se logra aprender ningún tipo de comportamiento debido a que el tamaño de las discretizaciones empleadas son demasiado pequeñas. Las configuraciones 64×32 y 128×64 con un mayor número de centroides representativos tanto para los estados como para las acciones, obtienen un rendimiento claramente superior al rendimiento del comportamiento base, π_T. De entre estas dos configuraciones, la configuración 64×32 es la que obtiene un mayor rendimiento llegando hasta los 13 millones de euros de beneficios aproximadamente.

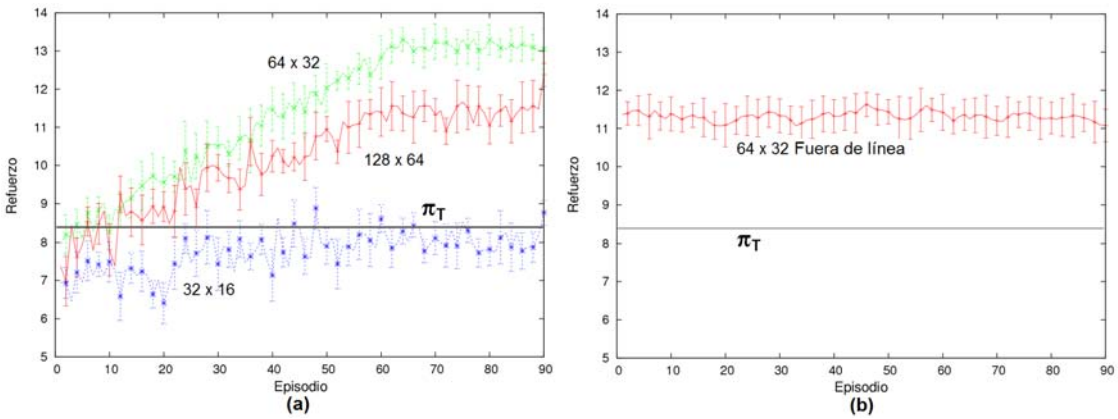


Figura 5.26: (a) Resultados de diferentes configuraciones de G-VQQL versión en línea en *SIMBA* (b) Resultado de G-VQQL versión fuera de línea en *SIMBA*

En cuanto a G-VQQL en su versión fuera de línea (figura 5.26 (b)), se muestra el rendimiento de la configuración 64×32 que mejores resultados ofrece en la versión en línea del algoritmo. En este caso, el aprendizaje se realiza con el conjunto de tuplas de experiencia T recogido del entorno mediante el comportamiento base π_T. Aunque la versión fuera de línea mejora el rendimiento del comportamiento base hasta los 11.5 millones de euros de beneficios aproximadamente, no se consiguen mejorar los resultados obtenidos por la versión en línea del algoritmo.

5.3.3.2. Ejecución del algoritmo CMAC-VQQL

La ejecución del algoritmo CMAC-VQQL también se compone de los pasos de recogida de tuplas de experiencia para construir el cuantificador vectorial asociado a las acciones descrito en la tabla 5.4, y de aprendizaje de la política de comportamiento descrito en la tabla 5.5 para la versión en línea y en la tabla 5.6 para la versión fuera de línea. La ejecu-

ción del primer paso del algoritmo es análogo a la ejecución del primer paso del algoritmo G-VQQL descrito en la sección anterior, pero en el caso del algoritmo CMAC-VQQL se construye un único cuantificador a partir del conjunto de acciones T_a recogido del entorno mediante el comportamiento π_T. La figura 5.27 muestra los resultados para diferentes configuraciones del algoritmo CMAC-VQQL en su versión en línea (figura 5.27 (a)) y en su versión fuera de línea (figura 5.27 (b)).

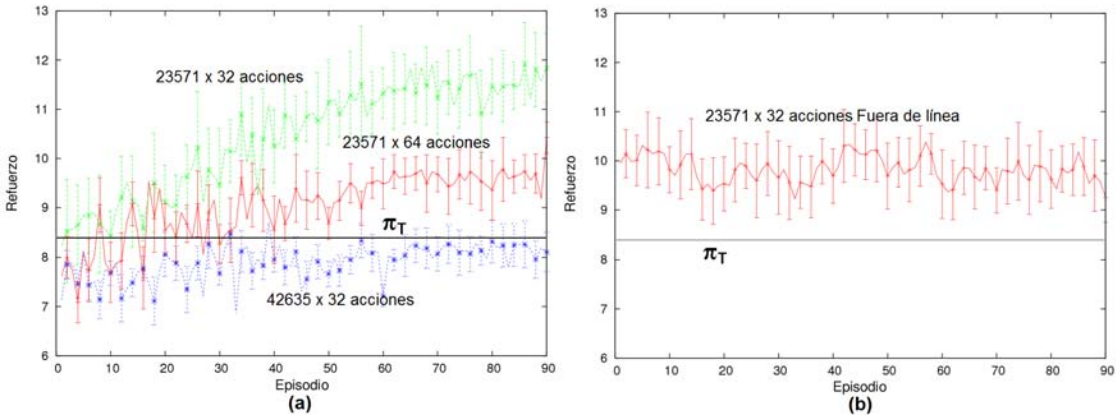


Figura 5.27: (a) Resultados de diferentes configuraciones de CMAC-VQQL versión en línea en *SIMBA* (b) Resultado de CMAC-VQQL versión fuera de línea en *SIMBA*

Los valores de los parámetros empleados para los procesos de aprendizaje son los mismos que en el caso del algoritmo G-VQQL descrito anteriormente. Además, como en el caso del algoritmo G-VQQL, se muestra el rendimiento del comportamiento base que alcanza los 8.4 millones de euros de beneficios. La etiquetas de las curvas de aprendizaje que se muestran en la figura, muestran en primer lugar la suma de todos los intervalos regulares en los cuales se divide cada una de las características asociadas a los estados, y en segundo lugar el tamaño de la discretización empleado para el espacio de acciones. Así, la etiqueta 23571×32 *acciones* indica que, en el proceso de aprendizaje correspondiente, la suma de intervalos para las características de estados asciende a 23571 y el tamaño asociado a la discretización del espacio de acciones es de 32 centroides. Además, en cada configuración, como en los dominios anteriores, se han solapado y desplazado 32 rejillas para alcanzar una mayor precisión en la generalización del espacio de estados. De esta forma, en la configuración con 23571 intervalos, el vector de pesos θ por cada acción está compuesto por $23571 \times 32 = 754272$ elementos, mientras que en la configuración con 42635 el vector de pesos θ está compuesto por $42635 \times 32 = 1364320$ elementos. En la versión en línea del algoritmo, la configuración 23571×32 *acciones* es la que mejor rendimiento obtiene llegando alrededor de los 12 millones de euros de beneficios. El incremento en el número de acciones de esta configuración hasta las 64 acciones produce políticas de comportamiento de menor calidad en torno a los 9.5 millones de euros de beneficios, y con el incremento en el número de intervalos en los que se dividen las características de los estados hasta los 42635 no se consiguen aprender comportamientos que mejoren el rendimiento del comportamiento base π_T. Por lo tanto, el incremento en el número de acciones o de elementos del vector de pesos ralentiza el aprendizaje puesto que es mucho mayor el número de parámetros que hay que ajustar adecuadamente obteniendo políticas de menor calidad, o incluso impidiendo algún tipo de convergencia del algoritmo.

En la versión fuera de línea del algoritmo presentada en la figura 5.27 (b), se muestran los resultados de la configuración 23571×32 *acciones* que mejor resultados ha ofrecido en

la versión en línea de la figura 5.27 (a). En este caso, se presentan las tuplas de experiencia recogidas del entorno mediante el comportamiento base π_T al algoritmo de la tabla 5.6. En este modo de funcionamiento, el algoritmo aprende una política de comportamiento que se sitúa entorno a los 10 millones de euros de beneficios mejorando ampliamente los resultados del comportamiento base π_T, aunque de menor rendimiento que la obtenida con la versión en línea del algoritmo.

5.3.3.3. Comparativa del rendimiento G-VQQL y CMAC-VQQL

Por último, la figura 5.28 muestra la comparativa del rendimiento de las mejores configuraciones obtenidas para los algoritmos G-VQQL (64×32 *acciones*) y CMAC-VQQL (23571×32 *acciones*) con el algoritmo evolutivo (Tabla 2.9), con el algoritmo CACLA (Tabla 2.7), y con el algoritmo CMAC para espacios de estados y acciones continuos (Tabla 2.4).

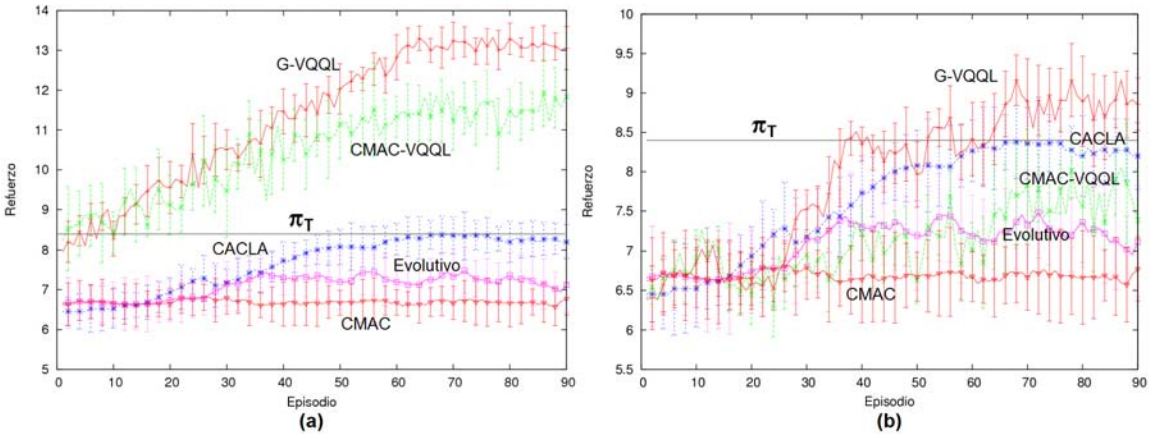


Figura 5.28: (a) Comparación de los resultados de G-VQQL y CMAC-VQQL con diferentes algoritmos en *SIMBA* cuando se utiliza el comportamiento base π_T en el paso de recogida de experiencia (Tablas 5.1 y 5.4) (b) Comparación de los resultados de G-VQQL y CMAC-VQQL con diferentes algoritmos en *SIMBA* cuando se utiliza un comportamiento aleatorio en el paso de recogida de experiencia (Tablas 5.1 y 5.4).

En la figura 5.28 (a), se muestra el rendimiento de los algoritmos G-VQQL y CMAC-VQQL cuando se emplea el comportamiento base π_T en el paso de recogida de experiencia de los algoritmos (Tablas 5.1 y 5.4), mientras que en la figura 5.28 (b) se muestra el rendimiento de los algoritmos G-VQQL y CMAC-VQQL cuando se emplea un comportamiento aleatorio en esta recogida de experiencia. En este dominio, todos los algoritmos abordan el problema reducido obtenido a partir de la selección de características llevada a cabo, con 12 características para los estados y 10 características para las acciones.

En cuanto a la comparación con el rendimiento del resto de algoritmos, como en el caso del dominio del *Octupus Arm*, en la figura 5.28 (a) destacan dos hechos fundamentales. Por un lado, se puede apreciar como los algoritmos G-VQQL y CMAC-VQQL comienzan sus procesos de aprendizaje en torno a los 8 millones de euros de beneficios, un rendimiento ligeramente inferior al comportamiento base π_T, mientras que el resto de algoritmos comienzan sus procesos de aprendizaje en torno a los 6.5 millones de euros de beneficios. Esta diferencia se debe a que la discretización de la región del espacio de acciones explorada

mediante el comportamiento base, π_T, proporciona a los algoritmos G-VQQL y CMAC-VQQL un conjunto de acciones válidas y coherentes desde el inicio, mientras que el resto de algoritmos tienen que explorar sobre todo el espacio de acciones lo que dificulta e incluso puede impedir la convergencia a políticas de comportamiento razonables. De esta forma, los algoritmos G-VQQL y CMAC-VQQL buscan la política de comportamiento óptima en regiones definidas del espacio que se consideran "interesantes" recorridas por el comportamiento base π_T y manteniendo las mismas relaciones estadísticas entre las características de las acciones definidas por éste, por lo que no tienen que explorar toda la inmensidad de los espacios de estados y acciones de este dominio como hacen el resto de aproximaciones. Por otro lado, se aprecia como ambos algoritmos son capaces de mejorar considerablemente el rendimiento del comportamiento base, π_T, empleado para realizar la exploración de los espacios de estados y acciones. En la figura 5.28 (b) en cambio, se puede apreciar cómo el comportamiento aleatorio permite explorar el espacio lo suficiente como para después construir cuantificadores que permiten al algoritmo G-VQQL superar ligeramente el rendimiento del comportamiento base, π_T, y del resto de algoritmos, y al algoritmo CMAC-VQQL superar el rendimiento del algoritmo basado en aprendizaje por refuerzo evolutivo y el del algoritmo CMAC. No obstante, estos resultados son muy inferiores a los obtenidos por los algoritmos G-VQQL y CMAC-VQQL cuando se emplea el comportamiento base, π_T, en el paso de recogida de experiencia. De nuevo, la exploración aleatoria en este dominio produce una exploración inadecuada del espacio que influye negativamente tanto en la calidad de las discretizaciones que se construyen, como en el posterior proceso de aprendizaje. Para ambos casos, el algoritmo G-VQQL obtiene mejores resultados que el algoritmo CMAC-VQQL debido al gran número de parámetros que hay que emplear en la configuración CMAC para obtener una buena política de comportamiento, lo que dificulta la convergencia del algoritmo.

En el caso del algoritmo evolutivo, se tratan de evolucionar redes de neuronas de arquitectura $12 \times 25 \times 10$, es decir, de 12 neuronas en la capa de entrada correspondientes a cada una de las características de los estados, una única capa oculta con 25 neuronas y 10 neuronas en la capa de salida correspondientes a las características de las acciones. Aunque se han probado multitud de topologías *ad-hoc*, finalmente se muestran los resultados para una red de 25 neuronas ocultas siguiendo el Teorema de Kolmogorov [Hecht-Nielsen, 1987]. En este dominio, la población empleada en el algoritmo evolutivo es de 10 individuos por lo que se producen evoluciones cada 10 episodios o simulaciones, es decir, una vez que se ha probado el rendimiento de todos los individuos de la población. En el caso del algoritmo CACLA, se trata de aproximar dos redes de neuronas en lugar de sólo una, la correspondiente a la política de comportamiento con una arquitectura $12 \times 25 \times 10$ y la correspondiente a la función de valor con una arquitectura $12 \times 25 \times 1$ que aproxima el valor de la función de valor para un estado en concreto. En ambos casos, se produce una mejora en el rendimiento, siendo más evidente en el algoritmo CACLA que llega a obtener un rendimiento similar al comportamiento base, aunque, de nuevo, la complejidad de las redes de neuronas que se tratan de aproximar impide la obtención de mejores políticas de comportamiento. En el caso del algoritmo CMAC, las características de las acciones también se dividen en intervalos regulares a la vez que se solapan y desplazan 32 rejillas para incrementar la precisión de la generalización. Estos intervalos hay que sumarlos a los empleados para generalizar los estados, lo que resulta en un vector de pesos θ con una enorme cantidad de parámetros que hay que ajustar adecuadamente, lo que impide el aprendizaje de mejores políticas de comportamiento.

Además de la complejidad de las redes de neuronas en el algoritmo evolutivo y en CA-

CLA, y el enorme número de elementos del vector de pesos θ necesario en el algoritmo CMAC, existen otros dos factores fundamentales que impiden el aprendizaje de mejores políticas de comportamiento por parte de estas técnicas. Por un lado, en la mayoría de las ocasiones estas técnicas no conservan las relaciones estadísticas existentes entre las características de las acciones que son necesarias para explorar adecuadamente el espacio. Por otro lado, incumplen continuamente las restricciones semánticas existentes en el modelo económico (e.g., una compañía no puede vender más producto que el stock que tiene disponible), lo que también dificulta la exploración adecuada del espacio. Modelar este tipo de conocimiento requeriría una gran interacción con expertos y puede no ser trivial.

5.4. Discusión

En este capítulo se han presentado los algoritmos G-VQQL y CMAC-VQQL para el aprendizaje en dominios con espacios de estados y de acciones de naturaleza continua y de grandes dimensiones. Además, se han puesto de relieve las dos grandes dificultades existentes en este tipo de dominios, y que hacen que fracasen las técnicas que tratan de resolver directamente el problema completamente continuo sin ningún tipo de reducción o discretización. Por un lado, el alto número de dimensiones del espacio de acciones dificulta cualquier tipo de exploración en busca de las acciones óptimas o cercanas a las óptimas. Una estrategia de exploración típica como $\epsilon - greedy$ podría ralentizar excesivamente el proceso de aprendizaje mediante la búsqueda aleatoria en el inmenso espacio de acciones. Además, una exploración aleatoria o inadecuada produce en la mayor parte de las ocasiones acciones incoherentes, que aún siendo válidas, no conducen al agente a explorar las regiones del espacio de estados más "interesantes", es decir, aquellas zonas del espacio de estados que resultan útiles para aprender una política de comportamiento óptima o cercana a la óptima. Por otro lado, abordar directamente un problema continuo y con un gran número de dimensiones sin ningún tipo de simplificación requiere la utilización de complejas estructuras con un alto número de parámetros que requieren ser ajustados y que dificultan la convergencia de los algoritmos (e.g., redes de neuronas de gran complejidad). En estos casos, puede ser necesario incorporar cierto conocimiento del dominio para que las técnicas de aprendizaje no fracasen.

En nuestro caso, los algoritmos G-VQQL y CMAC-VQQL atajan estas dos grandes dificultades mediante la exploración previa del entorno con un comportamiento base subóptimo que recoge experiencias de zonas relevantes del espacio de estados y de acciones. A partir de estas experiencias se construyen las discretizaciones y los cuantificadores asociados. La discretización del espacio de acciones mantiene las mismas relaciones estadísticas entre las características de las acciones que las exhibidas por el comportamiento base durante la exploración, lo que acelera considerablemente el proceso de exploración puesto que no se exploran aquellas acciones que incumplen estas relaciones, es decir, en todo momento se ejecutan acciones consideradas válidas y coherentes. La utilización del comportamiento base por parte de los algoritmos G-VQQL y CMAC-VQQL es más generalizable a otros dominios que la realizada por el algoritmo por refuerzo evolutivo empleado por Koppejan et al. [Koppejan and Whiteson, 2009] and Martín et al. [Martín and de Lope Asiaín, 2009] en el dominio del *Helicopter* donde se introducen varios comportamientos base en la población inicial con el fin de facilitar y acelerar la convergencia del algoritmo. Estos comportamientos base no son siempre modelables mediante redes de neuronas lo que impide la introducción de este conocimiento en la población inicial, problema que se ve agravado en dominios com-

plejos de grandes dimensiones. Esta utilización tampoco es comparable a los procedimientos de inicialización propuestos en algunos casos [Driessens and Džeroski, 2004], donde antes de comenzar el proceso de entrenamiento en línea se realiza un *pre-entrenamiento* fuera de línea en el cual se presenta al algoritmo un conjunto de tuplas de experiencia generadas mediante un comportamiento base o experto. Este procedimiento también constituye un método de inicialización, que podría ser utilizado por los algoritmos propuestos tan sólo encadenando el aprendizaje fuera de línea (Tablas 5.3 y 5.6) con el aprendizaje en línea (Tablas 5.2 y 5.6), con la ventaja añadida que aportan nuestros algoritmos que sólo se centran en regiones relevantes del espacio. Además, una selección de características adecuada junto con la discretización de los espacios de estados y acciones, o simplemente del espacio de acciones, reduce considerablemente el número de parámetros que hay que ajustar adecuadamente para aprender una política de comportamiento óptima o cercana a la óptima, en comparación con las técnicas que tratan de resolver el problema completamente continuo. No obstante, el principal inconveniente de utilizar un comportamiento base subóptimo, es también el inconveniente que surge en las técnicas basadas en aprendizaje por demostración [Argall *et al.*, 2009], y es que el proceso de aprendizaje se ve fuertemente limitado por las capacidades del comportamiento que se utilice.

En dominios más pequeños como el *Cart-Pole* donde el comportamiento base utilizado para explorar el entorno es un comportamiento aleatorio, los algoritmos G-VQQL y CMAC-VQQL también han demostrado tener un alto rendimiento y una rápida velocidad de convergencia a políticas cercanas a las óptimas. No obstante, en dominios pequeños aparecen las desventajas asociadas a abordar problemas continuos previamente discretizados en lugar de resolver directamente el problema continuo como hacen las técnicas con las que se comparan los algoritmos G-VQQL y CMAC-VQQL. Ambos algoritmos emplean un conjunto estático y discreto de acciones durante todo el proceso de aprendizaje y no es posible encontrar para un estado dado una acción mejor y diferente a alguna disponible en el conjunto discreto de acciones. Por lo tanto, el rendimiento de la política de comportamiento que se aprende estará limitada por que el conjunto discreto de acciones que se disponga. En dominios con un número más elevado de características en los estados y las acciones como el simulador empresarial SIMBA, la exploración aleatoria no permite obtener unas buenas discretizaciones del espacio de estados y del espacio de acciones lo que limita posteriormente los procesos de aprendizaje. Aún así, el algoritmo G-VQQL consigue obtener mejores resultados que el resto del algoritmos con los que se ha comparado. En el dominio del *Octopus Arm* en cambio, la exploración aleatoria no permite realizar ningún tipo de exploración del espacio y los algoritmos no logran aprender una política de comportamiento válida. El éxito de los algoritmos G-VQQL y CMAC-VQQL queda supeditado a una exploración adecuada de los espacios de estados y acciones que permita construir unas discretizaciones que posibiliten la convergencia de los algoritmos.

En cuanto a la seguridad que otorga al agente el aprendizaje con los algoritmos propuestos, la utilización de comportamientos base expertos para recoger experiencia en aquellas zonas del espacio más relevantes evita posteriormente, durante el proceso de aprendizaje, visitar zonas inútiles del espacio, lo que acelera la velocidad de convergencia del algoritmo y reduce, en cierta medida, el número de daños que puede sufrir el agente (e.g., el número de bancarrotas que sufre una compañía en SIMBA). Por otro lado, aunque el aprendizaje fuera de línea en ambos algoritmos ha demostrado tener un rendimiento inferior a las versiones en línea, no se puede afirmar que el aprendizaje en línea sea mejor al aprendizaje fuera del línea. No es posible establecer una comparación directa puesto que el número de tuplas de experiencia utilizado por el aprendizaje en línea es mucho mayor que el utilizado

en el aprendizaje fuera de línea. Además, el aprendizaje fuera de línea ayuda a mantener la seguridad del agente en todo momento. Con este aprendizaje, en el caso del dominio del *Octopus* y el simulador empresarial SIMBA, en primer lugar las experiencias se recogen del entorno mediante un comportamiento base que, aunque subóptimo, es capaz de completar la tarea con éxito. Posteriormente, el aprendizaje del agente se realiza sin interaccionar con el entorno, sólo con las tuplas de experiencia recogidas por el comportamiento base en un paso anterior de forma segura. Una vez entrenado con estas tuplas, el agente, tanto en el dominio del *Octopus* como en SIMBA (no así en el *Cart-Pole*), es capaz de aprender una política de comportamiento superior en rendimiento al comportamiento base utilizado en la exploración del entorno. El problema surge cuando el agente utiliza esta política aprendida fuera de línea para interaccionar con el entorno por primera vez, puesto que se desconoce a priori la calidad de la política aprendida o, en el peor de los casos, si se ha producido algún tipo de aprendizaje (e.g., podría tratarse de una política que produce bancarrotas de la compañía todo el tiempo).

Por último, en los dominios de experimentación probados, el algoritmo G-VQQL ha demostrado tener un rendimiento ligeramente superior al algoritmo CMAC-VQQL. La combinación de técnicas de aproximación de funciones y técnicas de discretización necesitan un mayor número de parámetros que requieren ser ajustados frente a aproximaciones que emplean sólo técnicas de discretización, lo que puede ralentizar y dificultar el proceso de aprendizaje. Si la discretización de los espacios de estados y acciones es adecuada, es posible hacer frente a problemas de gran complejidad obteniendo una rápida convergencia a políticas de comportamiento con un alto rendimiento.

Capítulo 6

Exploración Segura del Espacio de Estados y Acciones

En este capítulo se aborda el problema de la exploración segura en el contexto de Aprendizaje por Refuerzo. El aprendizaje por refuerzo se adapta bien a problemas con dinámicas complejas y donde los espacios de estados y acciones son de grandes dimensiones. No obstante, el problema de la exploración segura en este tipo de problemas supone un desafío añadido. Las técnicas tradicionales de exploración no son útiles para resolver tareas que entrañan algún tipo de riesgo, y donde procesos como el de prueba y error podrían seleccionar acciones cuya ejecución desde algunos estados producen daños en el agente, en el sistema de aprendizaje, o en entidades externas. Por eso, cuando el agente interacciona con un entorno peligroso de una alta dimensionalidad tanto para el espacio de estados como para el espacio de acciones, surge una importante pregunta: ¿Cómo se pueden evitar (o al menos minimizar) que la exploración de los espacios de estados y acciones cause daños? En este capítulo se introduce el algoritmo ***P****olicy* ***I****mprovement through* ***S****afe* ***R****einforcement* ***L****earning* (PI-SRL) que trata de dar respuesta a esta pregunta mejorando de una forma segura comportamientos robustos aunque subóptimos, denominados comportamientos base, para tareas con unos espacios de estados y acciones continuos, mediante el aprendizaje eficiente de la experiencia recuperada del entorno. En el capítulo anterior, los comportamientos base se utilizaban para exponer al agente a las zonas más interesantes o prometedoras del espacio para posteriormente realizar la discretización sobre estas regiones concretas de los espacios de estados y acciones. Procediendo de esta forma, se aceleraba considerablemente la velocidad de convergencia de los algoritmos, evitando consumir tiempo en explorar zonas inútiles del espacio de estados y, de forma colateral, reduciendo el número de visitas a situaciones con riesgo. El algoritmo PI-SRL también hace uso de estos comportamientos base o expertos para incorporar conocimiento sobre la tarea que se trata de aprender, aunque lo hace de dos formas diferentes:

1. En primer lugar, se utiliza para incorporar conocimiento inicial sobre la tarea, puesto que si se desconoce por completo es complicado realizar ningún tipo de aprendizaje. Además, al inicio del proceso de aprendizaje, donde no se tiene ningún tipo de conocimiento sobre el entorno, es imposible mantener al sistema en un estado *seguro* durante la exploración del espacio desconocido.

2. Por otro lado, se emplea para guiar el posterior proceso de exploración de los espacios de estados y acciones en busca de mejores políticas de comportamiento.

En este capítulo se argumenta por qué el uso de estos comportamientos base es indispensable en el aprendizaje de tareas complejas de grandes dimensiones y donde, además, el sistema de aprendizaje o entidades externas pueden sufrir daños cuando se seleccionan las acciones equivocadas. El método propuesto se evalúa en cuatro tareas de dinámica compleja, estocásticas y con unos espacios de estados y acciones continuos: el *Automatic Car Parking problem* (Sección 4.2.1), el *Cart-Pole* (Sección 4.2.2), el *Helicopter* (Sección 4.2.3), y el simulador empresarial SIMBA (Sección 4.2.5).

6.1. Introducción

Los investigados en aprendizaje por refuerzo están prestando cada vez más atención no sólo al aprendizaje de una política de comportamiento que maximice el refuerzo adquirido a largo plazo, sino también en la aproximación segura a soluciones en Problemas de Decisión Secuenciales (SDPs) [Schneider, 1996; Mihatsch and Neuneier, 2002; Perkins and Barto, 2002; Bagnell and Schneider, 2004; Hans, Alexander *et al.*, 2008; Martín and de Lope Asiaín, 2009; Koppejan and Whiteson, 2011]. Algunos estados del arte sobre la cuestión pueden encontrarse en [Geibel and Wysotzki, 2005; Defourny *et al.*, 2008]. No obstante, la aplicación de aprendizaje por refuerzo a tareas con riesgo debe asegurar el aprendizaje de una política de comportamiento razonable y, al mismo tiempo, respetar la seguridad del agente (i.e., evitando colisiones, daños). Muchas de las políticas de exploración utilizadas en aprendizaje por refuerzo no garantizan estas dos características. Por eso, cuando se aplican técnicas de aprendizaje por refuerzo a tareas con riesgo, surge una importante cuestión:

¿Cómo se puede llevar a cabo la exploración de los espacios de estados y acciones de forma que no cause daños y, al mismo tiempo, permita aprender la política de comportamiento óptima o una política cercana a la óptima?

En otras palabras, el agente debe explorar el entorno de forma segura y eficiente de modo que se logre minimizar el número de daños o colisiones que se produzcan. Existen muchos dominios en los cuales el proceso de exploración/explotación puede conducir a estados o acciones catastróficos para el agente [Geibel and Wysotzki, 2005]. El problema del helicóptero, introducido en la sección 4.2.3, es un ejemplo de este tipo de dominios. Este dominio involucra un gran riesgo puesto que algunas políticas pueden hacer que el helicóptero se estrelle, incurriendo en refuerzos negativos catastróficos. Algunas estrategias de exploración/explotación como $\epsilon - greedy$ pueden producir que el helicóptero se estrelle continuamente (especialmente si existe una alta probabilidad de seleccionar acciones aleatorias). Otro ejemplo de dominio peligroso se puede encontrar en la teoría de portfolio en economía, donde los analistas esperan encontrar un portfolio que maximice el beneficio que se obtiene a la vez que suprime el riesgo a sufrir grandes pérdidas [Luenberger, 1998]. El proceso de exploración en este tipo de dominios, mediante el cuál se encuentran y evalúan nuevas políticas, debe de llevarse a cabo con extremo cuidado. Para estos entornos se requiere un método que no sólo realice una exploración de los espacios de estados y acciones, sino que además se explore de forma segura.

En esta Tesis se propone un método para explorar de forma segura en tareas peligrosas y de naturaleza continua. El método requiere para su aplicación un comportamiento base (*baseline policy*) que es seguro, aunque subóptimo (de otra forma el aprendizaje no tendría sentido). Otras aproximaciones también utilizan un comportamiento base prede-

finido de una forma diferente a la nuestra. Koppejan and Whiteson [Koppejan and Whiteson, 2011] evolucionan redes de neuronas para aprender la tarea del helicóptero, pero empezando desde una red *prototipo* cuyos pesos se corresponden a un comportamiento base extraído del software de la competición de aprendizaje por refuerzo [Abbeel *et al.*, 2008]. Esta aproximación se puede considerar como una forma de inicialización que ha demostrado ser ventajosa en numerosos método evolutivos [Hernández-Díaz *et al.*, 2008; Poli and Cagnoni, 1997]. Martín y Lope [Martín and de Lope Asiaín, 2009] también evolucionan los pesos de redes de neuronas insertando en la población inicial varios comportamientos base (incluyendo el comportamiento base provisto en el software de la competición de aprendizaje por refuerzo para la tarea del helicóptero). En esta aproximación, para minimizar las probabilidades de evaluar políticas que pueden resultar catastróficas, se restringen los cruzamientos y mutaciones de forma que solo se permiten pequeñas modificaciones de los comportamientos base introducidos al inicio.

En esta Tesis, se introduce el algoritmo PI-SRL que permite el perfeccionamiento de comportamientos base utilizando aprendizaje por refuerzo en tareas con riesgo. El algoritmo PI-SRL está compuesto de dos pasos. En el primero, se aproxima con una base de casos y mediante técnicas de imitación del comportamiento (*Behavioral Cloning*) [Bain and Sammut, 1995; Anderson *et al.*, 2000; Abbott, 2008] un comportamiento base seguro aunque subóptimo. Para alcanzar este objetivo, se utilizan técnicas de Razonamiento Basado en Casos (*Case-Based Reasoning*) (CBR) [Aamodt and Plaza, 1994; Bartsch-Spörl *et al.*, 1999], que han sido aplicadas con éxito en tareas de imitación de comportamientos en el pasado [Floyd *et al.*, 2008]. En el segundo paso, el algoritmo PI-SRL trata de mejorar el rendimiento del comportamiento base aprendido anteriormente mediante la exploración segura de los espacios de estados y acciones. De esta forma, la base de casos construida en el primer paso, compuesta por casos de la forma *estado-acción*, es mejorada mediante el reemplazo de casos viejos e inútiles por nuevos con mejores acciones, y mediante la adición de nuevos casos que representan regiones más relevantes de los espacios de estados y acciones. Para llevar a cabo esta exploración, se añaden pequeñas cantidades de ruido aleatorio siguiendo una distribución Gaussiana a las acciones de la base de casos aprendida en el paso anterior. Esta estrategia de exploración Gaussiana ha demostrado obtener resultados muy positivos en trabajos previos [van Hasselt and Wiering, 2007].

Las principales aportaciones del método propuesto son: la utilización de una *función de riesgo* que permite determinar si un estado es seguro o no, y un comportamiento base que aconseja acciones seguras en estados considerados de riesgo (i.e., estados donde la toma de una acción equivocada puede conducir a situaciones catastróficas). Además, se presenta una nueva definición de riesgo basada en el espacio conocido y desconocido por el agente. Esta nueva definición de riesgo es completamente diferente de las definiciones tradicionales que se encuentran en la literatura [Geibel, 2001; Mihatsch and Neuneier, 2002; Geibel and Wysotzki, 2005]. Además, se reportan resultados del algoritmo PI-SRL en cuatro dominios de experimentación diferentes: el *Automatic Parking Car Problem* (sección 4.2.1), el *Cart-Pole* (sección 4.2.2), el *Helicopter Control Task* (sección 4.2.3), y el simulador empresarial SIMBA (sección 4.2.5). En la fase de experimentación, se propone aprender la política óptima o una política cercana a la óptima en cada uno de estos dominios, a la vez que se trata de minimizar el número de veces que el coche se estrella contra la pared, el número de desbalanceos del péndulo, las colisiones del helicóptero, y el número de veces que una compañía sufre una bancarrota durante el proceso de aprendizaje, respectivamente.

La aproximación propuesta puede entenderse como una forma de aprendizaje cooperativo que puede ser ilustrada mediante la analogía de aprendizaje padre-hijo/a, donde el

padre toma el rol del comportamiento base y el hijo/a toma el rol del agente que lleva a cabo el proceso de aprendizaje. Al principio, el hijo no sabe mucho acerca del mundo que le rodea. Para conocer más acerca de éste, necesita explorar. Durante esta exploración, la mayor parte de las situaciones con las que se encuentra son nuevas para él/ella. En estas situaciones, en las cuales no sabe qué hacer, prefiere llevar a cabo las acciones aconsejadas por su padre. De esta forma, el hijo/a incorpora el conocimiento del padre a su propio conocimiento sobre el mundo que le rodea. Después de un tiempo, el hijo/a se siente cómodo tomando las acciones aprendidas de su padre. Las situaciones desconocidas se han convertido en situaciones conocidas en las cuales sabe exactamente qué acción tomar. No obstante, la curiosidad del niño/a le lleva a explorar nuevas acciones más allá de las aconsejadas por su padre en estas situaciones conocidas. De esta forma, el niño/a investiga nuevas acciones que son similares a las aprendidas del padre. Durante este proceso, él/ella puede descubrir nuevas y mejores acciones que las aconsejadas por el padre en una situación determinada. No obstante, este proceso de investigación puede conducir al niño/a a situaciones nuevas y desconocidas para las cuales no tiene información del padre sobre cómo actuar. Si esto ocurre, el niño/a solicita inmediatamente el consejo de su padre sobre qué acción tomar en esta nueva situación descubierta. De esta forma, el niño/a explora de forma segura el mundo que le rodea.

El resto del capítulo se organiza de la siguiente manera. La sección 6.2 introduce algunas definiciones que serán necesarias para comprender posteriormente la descripción del algoritmo PI-SRL llevada a cabo de forma pormenorizada en la sección 6.3. La sección 6.5 muestra la evaluación realizada utilizando los diferentes dominios de experimentación con riesgo descritos en la sección 4.2. Para finalizar el capítulo, en la sección 6.6 se presentan las principales conclusiones y la conveniencia del método propuesto.

6.2. Definiciones

Para ilustrar el concepto de exploración segura que hay detrás de la aproximación propuesta, se considera el problema de navegación mostrado en la figura 6.1.

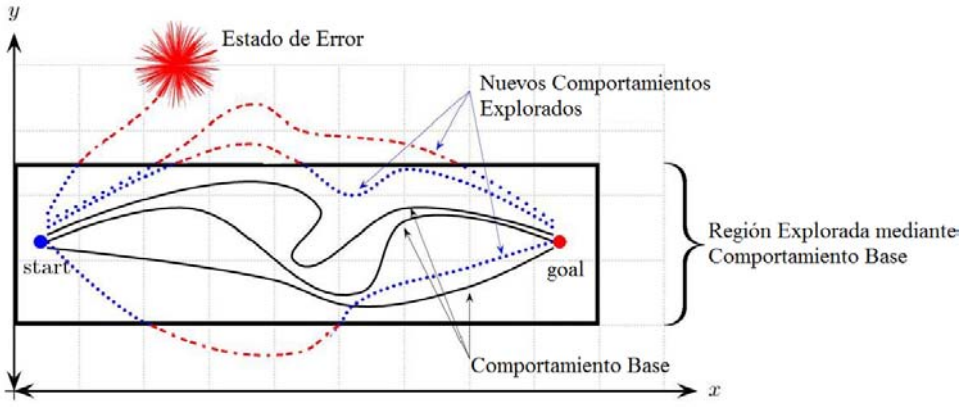


Figura 6.1: Estrategia de exploración basada en la adición de ruido aleatorio a las acciones.

En la figura, la tarea consiste en aprender una política de control que permita alcanzar el estado meta desde el estado de partida, a partir de un conjunto de trayectorias de demostración llevadas a cabo por un comportamiento base. En este entorno, se asume que la tarea es complicada debido a la estocasticidad y la compleja dinámica del entorno (e.g., una

superficie extremadamente irregular en el caso del dominio de navegación con robots o los efectos del viento en el caso de controlar un helicóptero). Esta estocasticidad hace imposible completar la tarea usando exactamente la misma trayectoria cada vez. Para llevar a cabo el aprendizaje de la tarea, se considera que se dispone de un conjunto de demostraciones sobre cómo se ejecuta correctamente (mostradas mediante las líneas negras continuas de la figura 6.1). Este conjunto de demostraciones está compuesto por diferentes trayectorias que cubren una región bien definida del espacio de estados (la región dentro del rectángulo en la figura 6.1). Nuestra aproximación se basa en añadir pequeñas cantidades de ruido aleatorio siguiendo una distribución Gaussiana a las trayectorias del comportamiento base que permitan encontrar nuevas y mejores trayectorias para llevar a cabo la tarea. Este ruido afecta a las trayectorias del comportamiento base de diferentes maneras, dependiendo de la cantidad de ruido aleatorio que se añada, y que determinará la cantidad de riesgo que se quiera asumir. Si no se desea asumir riesgos, la cantidad de ruido aleatorio que se añada a las trayectorias del comportamiento base debería ser 0, pero en este caso no se descubrirían nuevas y mejores formas de llevar a cabo la tarea (aunque el robot nunca se caería por un precipicio, o el helicóptero nunca se estrellaría). Si por el contrario se desea asumir un nivel intermedio de riesgo, se añadirían pequeñas cantidades de ruido aleatorio a las trayectorias del comportamiento base, de forma que sería posible encontrar nuevas trayectorias que permiten completar la tarea (las líneas punteadas azules en la figura 6.1). En algunos casos, la exploración de nuevas trayectorias, conduce al robot a regiones del espacio de estados desconocidas (líneas discontinuas rojas en la figura 6.1). El robot debe ser capaz de detectar esta situación (mediante una función de riesgo), y utilizar el comportamiento base para retornar a una situación segura y conocida del espacio de estados. Por último, si se desea asumir un nivel muy alto de riesgo, se añadirían grandes cantidades de ruido aleatorio a las trayectorias proporcionas por el comportamiento base, lo que de nuevo permitiría descubrir nuevas trayectorias. No obstante, un nivel muy alto de riesgo puede conducir al robot a estados del espacio de estados desconocidos muy alejados del espacio conocido, y dónde la probabilidad de sufrir o causar daños se incrementan. Este proceso de exploración permite al robot explorar progresivamente de una forma segura los espacios de estados y acciones permitiendo encontrar nuevas y mejores formas de completar la tarea que las llevadas a cabo por el comportamiento base, y donde la seguridad depende de la cantidad de riesgo que se asuma.

6.2.1. Estados de Error y de No-Error

En este capítulo se seguirá, en la medida de lo posible, la notación introducida por Geibel et al. [Geibel and Wysotzki, 2005] para posteriormente introducir nuestra propia definición del concepto de riesgo. Geibel et al. asocian el concepto de riesgo a los conceptos de estados de error y de no-error. Básicamente, los estados de error son aquellos que no se desean visitar o que pueden producir daños en el agente.

Definición 6.1 ***Estados de Error y de No-Error***. *Sea S el conjunto de estados. Sea $\Phi \subset S$ el conjunto de estados de error. Un estado $s \in \Phi$ es un estado no deseado y terminal, lo que significa que el control del agente finaliza cuando alcanza s con daños en el agente, en el sistema de aprendizaje o en entidades externas. Además, el conjunto $\Gamma \subset S$ se considera un conjunto de estados de no error y terminales con $\Gamma \cap \Phi = \emptyset$, y donde el control del agente finaliza normalmente sin daños.*

En términos de aprendizaje por refuerzo, si el agente entra en un estado de error, el episodio actual finaliza con daños en el agente, en el sistema de aprendizaje o en otra entidad. Por el contrario, si el agente entra en un estado de no-error, el episodio finaliza normalmente sin daños. De esta forma, Geibel et al. definen el riesgo asociado a un estado s con respecto a la política π, $\rho^{\pi}(s)$, como la probabilidad de que un estado en la secuencia $(s_i)_{i\geq 0}$, con $s_0 = s$, la cual es generada mediante la política de comportamiento π, termine en un estado de error $s' \in \Phi$. Por definición, $\rho^{\pi}(s) = 1$ si $s \in \Phi$. Si $s \in \Gamma$, entonces $\rho^{\pi}(s) = 0$ porque $\Phi \cap \Gamma = \emptyset$. Para estados $s \notin \Phi \cap \Gamma$, el riesgo depende de las acciones seleccionadas mediante la política de comportamiento π. Estas definiciones nos proporcionan el marco teórico para introducir nuestra propia definición de riesgo asociada a estados conocidos y desconocidos.

6.2.2. Estados Conocidos y Desconocidos en Espacios de Estados y Acciones Continuos

En esta Tesis se considera un espacio de estados continuo n-dimensional, $S \subset \Re^n$, donde cada estado $s = (s_1, s_2, \ldots, s_n) \in S$ es un vector de números reales y cada dimensión posee su propio dominio individual $D_i^s \subset \Re$. De forma similar, se considera un espacio de acciones continuo y m-dimensional $A \subset \Re^m$ donde cada acción $a = (a_1, a_2, \ldots, a_m) \in A$ es un vector de números reales y cada dimensión tiene su propio dominio individual $D_i^a \subset \Re$. Además, el agente que se considera aquí está dotado de una base de casos B de tamaño máximo η. Cada elemento de la memoria representa un par *estado-acción* (i.e., un caso) que el agente ha experimentado anteriormente.

DEFINICIÓN 6.2 ***(Base de Casos)***. *Una base de casos es un conjunto de casos, $B = \{c_1 \ldots, c_\eta\}$. Cada caso, c_i, consiste de un par estado-acción (s_i, a_i) que el agente ha experimentado en el pasado, y su correspondiente valor asociado $V(s_i)$. Es decir $c_i =< s_i, a_i, V(s_i) >$, donde el primer elemento representa la parte del problema del caso y se corresponde con el estado s_i, el siguiente elemento es la parte solución del caso, i.e., la acción esperada cuando el agente se encuentra en el estado s_i. El último elemento, $V(s_i)$, es el valor de la función de valor asociada con el estado. Cada estado s_i está compuesto de n variables de estado continuas, y cada acción a_i está compuesta de m variables de acción también de naturaleza continua.*

Cuando un agente recibe un nuevo estado, s_q, primero recupera de la base de casos B el vecino más cercano de acuerdo a alguna métrica de similitud, y entonces ejecuta la acción asociada al caso recuperado. En esta Tesis se considera la distancia Euclídea como métrica de similitud (Ecuación 6.1).

$$d(s_q, s_i) = \sqrt{\sum_{j=0}^{n} (s_{q,j} - s_{i,j})^2} \tag{6.1}$$

La distancia Euclídea es útil cuando se espera que la función de valor sea continua y sin cambios bruscos a lo largo de todo el espacio de estados [Santamaría *et al.*, 1998]. No obstante, puesto que la función de valor se desconoce *a priori* y la distancia Euclídea no se adapta muy bien en muchos problemas, algunos investigadores comienzan a preguntarse cómo aprender o adaptar la métrica de distancia en sí misma de forma que se consigan mejores resultados. En mucho casos, sería deseable la utilización de técnicas de aprendizaje

para aprender una métrica de similitud para inducir medidas de distancia más adecuadas para cada dominio específico [Taylor *et al.*, 2011]. No obstante, la discusión y la aplicación de este tipo de técnicas quedan fuera de los objetivos planteados en esta Tesis. En esta Tesis, sólo nos centramos en dominios en los cuales se ha comprobado que la distancia Euclídea funciona adecuadamente (la distancia Euclídea ha sido aplicada anteriormente con éxito al problema del *Automatic Car Parking Problem* [Cichosz, 1995], el *Cart-Pole* [Martín and de Lope Asiaín, 2009], el *Helicopter Control Task* [Martín and de Lope Asiaín, 2009], y el simulador empresarial SIMBA [Borrajo *et al.*, 2009]).

Tradicionalmente, los aproximadores basados en bases de casos utilizan un umbral de densidad, θ, para determinar cuando un nuevo caso debería ser añadido a la base de casos. Cuando la distancia del vecino más cercano en B al nuevo estado recibido, s_q, es mayor que θ, un nuevo caso se añade a la base de casos. De esta forma, el parámetro θ define el tamaño de la región de clasificación para cada caso en la base de casos B (Figura 6.2). Si un nuevo estado s_q se encuentra dentro de la región de clasificación del caso c_i, se considera que el estado s_q es un estado **conocido**. Por lo tanto, los casos en B describen la **Política Basada en Casos**, π_B^θ, y su función de valor asociada, $V^{\pi_B^\theta}$.

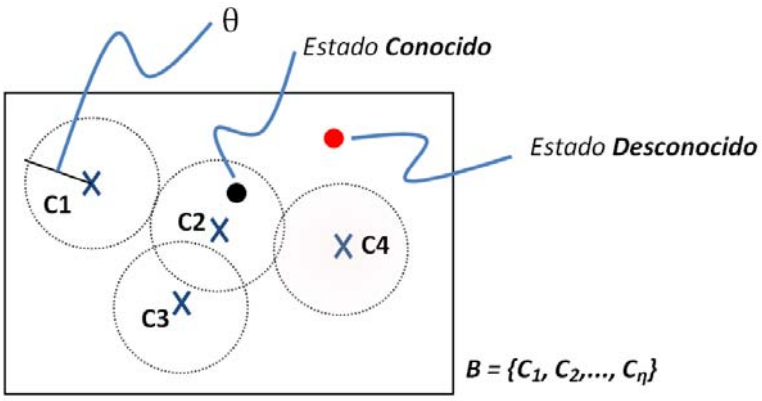


Figura 6.2: Estados conocidos y desconocidos.

DEFINICIÓN 6.3 ***(Estados Conocidos/Desconocidos)***. *Dada una base de casos* $B = \{c_1 \ldots, c_\eta\}$ *compuesta de casos* $c_i = (s_i, a_i, V(s_i))$, *y un umbral de densidad* θ, *un estado* s_q *es considerado* ***conocido***, *cuando* $\min_{1 \leq i \leq \eta} d(s_q, s_i) \leq \theta$; *en caso contrario el estado* s_q *es considerado* ***desconocido***. *Formalmente, se considera* $\Omega \subseteq S$ *el conjunto de estados* ***conocidos***, *y un conjunto adicional de estados* ***desconocidos*** $\Upsilon \subseteq S$ *con* $\Omega \cap \Upsilon = \emptyset$, *y* $\Omega \cup \Upsilon = S$.

Mediante la definición anterior, los estados se pueden clasificar fácilmente como estados **conocidos** o **desconocidos**. Cuando un agente recibe del entorno un nuevo estado $s \in \Omega$, ejecuta la acción a_i del caso c_i para el cual $d(s, s_i) = \min_{1 \leq j \leq \eta} d(s, s_j)$. No obstante, si el agente recibe un estado $s \in \Upsilon$, significa que la distancia para este estado a cualquier otro estado en B es mayor que θ, y por eso no se recupera ningún caso de la base de casos. Por lo tanto, el agente no sabe qué acción ejecutar en este estado.

DEFINICIÓN 6.4 ***(Función de Riesgo Basada en Casos)***. *Dada una base de casos* $B = \{c_1 \ldots, c_\eta\}$ *compuesta de casos* $c_i = (s_i, a_i, V(s_i))$, *el riesgo para cada estado* s *se define*

mediante la ecuación 6.2.

$$\varrho^{\pi_B^\theta}(s) = \begin{cases} 0 & \textit{if } \min_{1\leq j\leq \eta} d(s, s_j) < \theta \\ 1 & \textit{en caso contrario} \end{cases} \tag{6.2}$$

De esta forma, $\varrho^{\pi_B^\theta}(s) = 1$ si $s \in \Upsilon$ (i.e., s es **desconocido**), en el sentido de que el estado s no está asociado con ningún caso en la base de caso, y por tanto se desconoce qué acción seleccionar en este estado. Si $s \in \Omega$, entonces $\varrho^{\pi_B^\theta}(s) = 0$.

DEFINICIÓN 6.5 ***(Política Segura Basada en Casos)***. *La política basada en casos, π_B^θ, derivaba de la base de casos, $B = \{c_1 \ldots, c_\eta\}$, es segura si partiendo desde cualquier estado conocido inicial, s_0, con respecto a B, la ejecución de la Política Basada en Casos, π_B^θ, siempre produce estados de no-error con respecto a B.*

$$\forall s_0 \mid \varrho^{\pi_B^\theta}(s_0) = 0, \textit{ then } \forall (s_i)_{i>0}^{\pi_B^\theta} \varrho^{\pi_B^\theta}(s_i) = 0 \tag{6.3}$$

Además, se está asumiendo que la probabilidad de que la secuencia de estados $(s_i)_{i\geq 0}$ desde cualquier estado conocido $s_0 \in \Omega$, generada ejecutando la política π_B^θ, termine en un estado de error $s \in \Phi$ es $\rho^{\pi_B^\theta}(s_0) = 0$ (i.e., $\Omega \cap \Phi = \emptyset$).

DEFINICIÓN 6.6 ***(Cobertura de la Base de Casos Segura)***. *La cobertura de un estado individual s con respecto a la base de casos segura $B = \{c_1 \ldots, c_\eta\}$ se define mediante el estado s_i para el cual $\min_{1\leq i\leq \eta} d(s, s_i) \leq \theta$. Por lo tanto, se asume que la Base de Casos Segura no proporciona una acción para todo el espacio de estados, sino sólo para estados* ***conocidos*** *$s \in \Omega$.*

La figura 6.3 clarifica la relación que existe entre los estados conocidos/desconocidos y los estados de error/no-error. La región verde en la figura 6.3 representa la Política Segura Basada en Casos, π_B^θ, que se ha aprendido. Esta región del espacio de estados se corresponde con el espacio inicial conocido. El agente, siguiendo la política π_B^θ, nunca abandona la región verde del espacio de estados, y todos los episodios finalizan sin daños en el agente o en el sistema de aprendizaje. Por este motivo, un sub-conjunto de los estados de no-error es parte del espacio de estados conocido. Formalmente, sea Γ_Ω y Γ_Υ, subconjuntos de estados de no-error que pertenecen al espacio de estados conocido y desconocido, respectivamente, con $\Gamma_\Omega \cup \Gamma_\Upsilon = \Gamma$. Entonces $\Gamma_\Omega \subset \Omega$. La región amarilla en la figura 6.3 representa el espacio desconocido Υ. Un subconjunto de estados de no-error está en el espacio desconocido, al igual que el conjunto de estados de error. Formalmente, $\Gamma_\Upsilon \subset \Upsilon$ y $\Phi \subset \Upsilon$. La idea detrás del algoritmo PI-SRL propuesto en esta Tesis es:

- En primer lugar, aprender el espacio conocido derivado de la Política Segura Basada en Casos, π_B^θ (la región verde),
- en segundo lugar, ajustar el espacio conocido (la región verde) al espacio desconocido (región amarilla), de forma que se exploren nuevos y mejores comportamientos, evitando visitar la región roja (estados de error). Durante este proceso de ajuste del espacio conocido al espacio utilizado por políticas seguras y mejores, el algoritmo puede "olvidar" estados conocidos inútiles, como se mostrará en la sección 6.5.

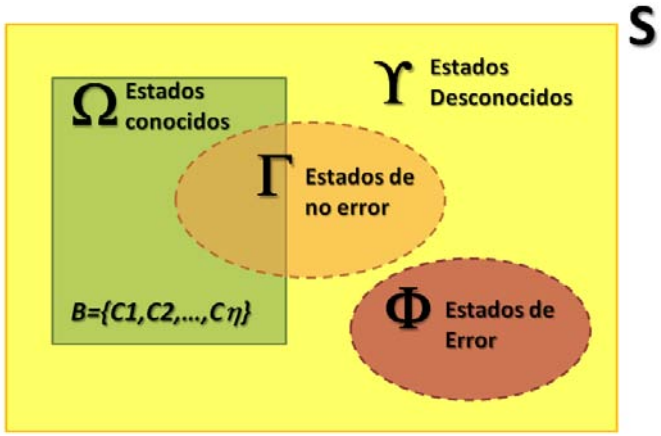


Figura 6.3: Estados Conocidos/Desconocidos y de Error/No-Error dada la base de casos B.

6.2.3. Las Ventajas de Utilizar Conocimiento Experto

En esta sección se ponen de relieve las ventajas de utilizar comportamientos expertos en aprendizaje por refuerzo. Estas ventajas son: (i) proveer conocimiento inicial sobre la tarea que se trata de aprender (cuya utilidad ha quedado demostrada en el capítulo 5), (ii) apoyar el proceso de exploración de los espacios de estados y acciones. Se argumenta por qué se cree que este conocimiento es indispensable para abordar mediante aprendizaje por refuerzo problemas realistas y muy complejos, con espacios de estados y acciones de grandes dimensiones, y donde la selección de una acción equivocada puede conducir a una situación indeseada.

Proporcionar conocimiento inicial sobre la tarea. La mayoría de los algoritmos de aprendizaje por refuerzo comienzan el aprendizaje sin un conocimiento previo sobre la tarea que se trata de abordar. En estos casos, se suelen utilizar estrategias de exploración como $\epsilon-greedy$. Esta estrategia produce una exploración aleatoria de los espacios de estados y acciones con el fin de recopilar conocimiento suficiente sobre la tarea que se trata de aprender. Solo cuando se ha recopilado suficiente información, el algoritmo comienza a aprender. No obstante, durante esta exploración aleatoria se malgasta mucho tiempo visitando regiones irrelevantes del espacio, donde no tiene por qué encontrarse la política óptima buscada. Este problema se agrava en dominios con espacios de estados y acciones de grandes dimensiones donde una exploración aleatoria podría no llegar nunca a alcanzar aquellas regiones relevantes del espacio. Además, en muchas tareas de aprendizaje por refuerzo donde se trabaja con robot reales, no es posible utilizar una exploración aleatoria para recopilar información del entorno. Cuando se trabaja con robots reales, suficiente información puede ser más información de la que un robot real puede ser capaz de recuperar el entorno. En un entorno de aprendizaje *real*, cada paso requiere una cierta cantidad de tiempo para ejecutarse, a diferencia de los entornos simulados donde el tiempo se puede *acelerar*. Por tanto, en un entorno *real*, el número de pasos de aprendizaje que se pueden ejecutar es considerablemente menor que en un entorno simulado, por eso estos pasos no pueden ser malgastados haciendo una exploración aleatoria del entorno. Por último, es completamente imposible evitar situaciones indeseables en tareas con riesgo sin cierta cantidad de conocimiento inicial sobre la tarea que se trata de aprender. La exploración aleatoria sin conocimiento inicial requiere visitar un estado indeseable antes de etiquetarlo como "indeseable". No obstante, las visi-

tas a estos estados puede resultar en daños en agente, en el sistema de aprendizaje, o en entidades externas, por lo que conviene evitarlos desde el comienzo del proceso.

Para paliar estos problemas se suelen emplear conjuntos finitos de demostraciones llevados a cabo por expertos, que son utilizados para incorporar conocimiento inicial sobre la tarea que se trata de resolver. Hay dos formas fundamentales de utilizar este comportamiento experto: (i) en primer lugar puede ser utilizado para inicializar el algoritmo de aprendizaje, y (ii) puede ser utilizado para derivar directamente una política de comportamiento a partir de estas demostraciones. En el primer caso, el algoritmo es provisto con ejemplos o demostraciones que utiliza para inicializar la función de valor, y para conducir al agente a través de las regiones más relevantes de los espacios de estados y acciones. Después de la ejecución de estos pasos iniciales de entrenamiento, el algoritmo de aprendizaje comienza a utilizar una estrategia de exploración Boltzmann o totalmente avariciosa basada en los valores a los que previamente se ha inicializado la función de valor [Driessens and Džeroski, 2004]. De esta forma, el algoritmo queda expuesto a las regiones más relevantes de los espacios de estados y acciones desde el principio del proceso de aprendizaje. El problema principal de esta aproximación es que el subsecuente proceso de exploración que se lleva a cabo una vez inicializada la función de valor, puede conducir a estados para los cuales no se ha suministrado demostración, y donde el agente no tiene información sobre cómo actuar y, por tanto, las probabilidades de daños se incrementan enormemente. El segundo caso se refiere a las aproximaciones basadas en Aprendizaje por Demostración [Argall *et al.*, 2009], donde una política de comportamiento se deriva de un conjunto de demostraciones suministradas por un experto. El problema de esta aproximación es que el rendimiento final de la política derivada está fuertemente limitada para las habilidades del experto. En este caso, una forma de mejorar el rendimiento del agente es explorar más allá de las demostraciones suministradas por el experto. No obstante, de nuevo, durante este proceso de exploración, se pueden visitar estados sin demostración (i.e., estados desconocidos).

Apoyo al proceso de exploración. Por lo tanto, proveer conocimiento inicial sobre la tarea que se trata de aprender ayuda a paliar los problemas que produce una exploración aleatoria, pero no es suficiente para prevenir situaciones indeseables en los subsiguientes procesos de exploración llevados a cabo para mejorar las habilidades del experto. Es necesario, además, un mecanismo que guíe este proceso de exploración manteniendo alejado al agente de los estados de error. En esta Tesis, se propone utilizar también al experto para proporcionar acciones en estados desconocidos en lugar de utilizar la política derivada de la aproximación actual de la función de valor. Una forma para prevenir que el agente se encuentre en situaciones sin demostración previa durante este proceso de exploración, sería pedir al experto que generase al principio una demostración por cada estado del espacio de estados. No obstante, esto no es posible porque (i) es computacionalmente inviable debido al gran número de estados que hay en el espacio de estados (que es de naturaleza continua), y (ii) el experto no puede ser forzado a dar una acción por cada estado puesto que muchos de esos estados son inútiles para aprender la política de comportamiento óptima. Por lo tanto, PI-SRL sólo solicita una acción del experto durante el proceso de exploración cuando realmente es requerida, es decir, cuando el agente se encuentra en un estado desconocido.

En este capítulo se considera que en todo momento está disponible un experto para la tarea que se trata de resolver, y a este experto se le considera el comportamiento base. Muchos trabajos hasta la fecha han hecho uso de expertos humanos, aunque en otros trabajos se han utilizado también expertos robóticos, políticas de comportamientos escritas a mano, o planificadores [Argall *et al.*, 2009]. En esta Tesis se utilizan controladores automáticos subóptimos como expertos en los diferentes dominios propuestos. Sea π_T la política de este

experto.

DEFINICIÓN 6.7 ***(Comportamiento Base)***. *La política π_T se considera el* ***comportamiento base***. *Se asume que este comportamiento base cuenta con tres características: (i) Es capaz de suministrar demostraciones seguras de la tarea que se trata de aprender a partir de las cuales se puede extraer conocimiento inicial, (ii) es capaz de apoyar el proceso de exploración llevado a cabo para superar las habilidades del experto suministrando acciones en estados desconocidos, reduciendo la probabilidad de visitar estados de error, y retornando el sistema a una situación conocida, y (iii) su rendimiento está lejos del óptimo, puesto que en otro caso el aprendizaje no tendría sentido.*

El algoritmo PI-SRL utiliza el comportamiento base π_T para dos propósitos diferentes. En primer lugar, utiliza las demostraciones del comportamiento base π_T para proporcionar conocimiento inicial sobre la tarea que se trata de aprender. Para ello, aprende la región inicial de estados conocidos que se deriva de la Política Segura Basada en Casos, π_B^θ. El propósito es imitar el comportamiento base π_T mediante la Política Segura Basada en Casos, π_B^θ. En segundo lugar, PI-SRL utiliza el comportamiento base π_T para apoyar el posterior proceso de exploración llevado a cabo para mejorar las habilidades de la previamente aprendida Política Segura Basada en Casos, π_B^θ. A medida que el proceso de exploración avanza, PI-SRL solicita acciones al comportamiento base π_T cuando realmente son requeridas, i.e., cuando el agente se encuentra en un estado desconocido sobre el que no ha recibido ninguna demostración previa sobre cómo actuar (figura 6.4). En este paso, el comportamiento base π_T, actúa como una política de respaldo ante estados desconocidos. Cuando el agente se encuentra en un estado desconocido, ejecuta las acciones provistas por el comportamiento base hasta que no se regrese a una situación de estados conocidos. Este proceso mantiene al agente alejado de los estados de error o, al menos, reduce el número de veces que se visitan. En este punto cabe destacar que el comportamiento base π_T no es capaz de demostrar la acción correcta para cada posible estado del espacio de estados pero, aunque no es capaz de proveer la mejor acción, al menos es capaz de seleccionar una acción más segura que la seleccionada por un proceso de exploración aleatoria.

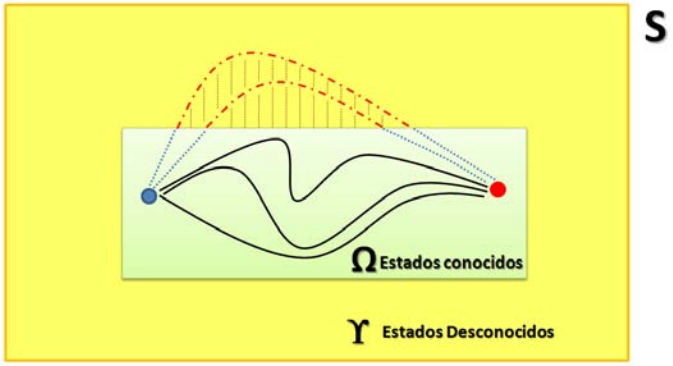


Figura 6.4: El proceso de exploración solicita una acción al comportamiento base cuando realmente se necesita.

6.2.4. El Parámetro de Riesgo

Para hacer la exploración tan segura como sea posible, parece recomendable no moverse arbitrariamente por el espacio de estados, sino expandir el espacio conocido gradualmen-

te comenzando por un estado conocido. Esta exploración es llevada a cabo perturbando o añadiendo ruido aleatorio siguiendo una distribución Gaussiana a las trayectorias estado-acción generadas mediante la política basada en casos π_B^θ. De esta forma, es posible encontrar nuevas y mejores formas de completar la tarea que se trata de aprender. La exploración Gaussiana se lleva a cabo alrededor de la aproximación actual de la acción a_i para el estado actual conocido $s_q \in \Omega$, con $c_i = (s_i, a_i, V(s_i))$ y $d(s_q, s_i) = \text{mín}_{1 \leq j \leq \eta}\, d(s_q, s_j)$. La acción que se selecciona finalmente se muestrea a partir de la distribución Gaussiana con media en la acción de salida dada por el caso seleccionado en B como se describe a continuación. Si a_i denota la acción que el algoritmo ofrece como salida, la probabilidad de seleccionar una acción a'_i, $\pi(s, a'_i)$, se calcula usando la Ecuación 6.4.

$$\pi(s, a'_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(a'_i - a_i)^2/2\sigma^2} \quad \text{if } \sigma^2 > 0. \tag{6.4}$$

La forma de la campana de la distribución Gaussiana depende del parámetro σ (la desviación estándar). En nuestro caso, σ se utiliza como un "parámetro de anchura". Valores elevados de σ implican una forma en la "campana Gaussiana" ancha, incrementando de esta forma la probabilidad de seleccionar acciones a'_i muy diferentes a la acción actual a_i. Por el contrario, un valor pequeño σ implica una forma en la "campana Gaussiana" estrecha, incrementando la probabilidad de visitar acciones a'_i muy similares a la acción actual a_i. Si $\sigma^2 = 0$, se asume que $\pi(s, a_i) = 1$. Por lo tanto, el valor de σ está directamente relacionado con la cantidad de ruido que se añade a las trayectorias estado-acción generadas mediante la política π_B^θ. Valores más elevados de σ implican más ruido, y más probabilidades de visitar estados desconocidos.

DEFINICIÓN 6.8 ***(Parámetro de Riesgo)***. *El parámetro σ se considera un parámetro de* ***riesgo***. *Valores elevados de σ incrementan las probabilidades de visitar estados distantes* ***desconocidos***, *y por lo tanto, se incrementa la probabilidad de visitar estados de error.*

La ejecución de estas acciones con ruido conduce al agente a los bordes del espacio conocido y lo fuerzan a ir ligeramente más allá, al espacio desconocido, en busca de mejores políticas seguras. Transcurrido un tiempo, la ejecución de este proceso de exploración incrementa el espacio de estados que es conocido por el agente. Este proceso de exploración se lleva a cabo para mejorar el rendimiento de la Política Segura Basada en Casos π_B^θ. El parámetro de riesgo, σ, y el parámetro θ, son parámetros de diseño que son elegidos por el usuario. En la sección 6.4 se dan las pautas sobre cómo se pueden seleccionar sus valores.

Por último, conviene destacar que nuestra aproximación está basada en dos asunciones lógicas en aprendizaje por refuerzo derivadas de los principios de la generalización [Kaelbling *et al.*, 1996; Sutton and Barto, 1998]:

i) **Estados cercanos tienen acciones óptimas similares**. En problemas con espacios de estados continuos, es imposible para el agente visitar cada estado del espacio de estados y almacenar la función de valor (o la acción óptima) para cada uno de estos estados. Este es el motivo por el cuál se necesitan las técnicas de generalización en aprendizaje por refuerzo. En problemas con espacios de estados continuos y de grandes dimensiones, se espera que estados similares tengan funciones de valor similares y acciones óptimas similares. Por lo tanto, es posible utilizar la experiencia recuperada del entorno

para aprender sobre un subconjunto limitado del espacio de estados y producir una buena aproximación sobre un subconjunto mucho mayor [Justin Boyan, 1995; Hu *et al.*, 2001; Fernández and Borrajo, 2008b]. Cabe destacar que en los dominios propuestos una acción óptima también es considerada una acción segura en el sentido de que nunca produce estados de error.

ii) **Acciones similares en estados similares tienden a producir efectos similares**. Si se considera un dominio determinista, la acción a_t ejecutada en el estado s_t siempre produce el mismo estado s_{t+1}. En un dominio estocástico, intuitivamente se espera que la ejecución de una acción a_t en el estado s_t produzca estados similares, i.e., produce estados $\{s^1_{t+1}, s^2_{t+1}, s^3_{t+1}, \ldots\}$ donde $\forall i, j \; i \neq j \; dist(s^i_{t+1}, s^j_{t+1}) \approx 0$. Además, la ejecución de la acción $a'_t \sim a_t$ en un estado $s'_t \sim s_t$ produce estados $\{s'^1_{t+1}, s'^2_{t+1}, s'^3_{t+1}, \ldots\}$ donde $\forall i, j$ $dist(s'^i_{t+1}, s^j_{t+1}) \approx 0$. En este capítulo, como se explicó anteriormente, se ha seleccionado la distancia Euclídea como medida de similitud puesto que ya ha sido utilizada con éxito en los dominios propuestos. Esta asunción también nos permite usar técnicas de generalización para el espacio de acciones puesto que las acciones que producen los mismos efectos sobre el entorno se pueden agrupar como una única acción [Jiang, 2004]. En problemas con espacio de acciones continuos la necesidad de técnicas de generalización es incluso más pronunciada [Kaelbling *et al.*, 1996]. En nuestro caso, este hecho también nos permite asumir que valores pequeños de σ incrementan la probabilidad de visitar estados conocidos, y por lo tanto, explorar menos y tomar menos riesgos, mientras que valores más grandes de σ incrementan las probabilidades de visitar estados de error.

6.3. El Algoritmo *Policy Improvement through Safe Reinforcement Learning*

El algoritmo está compuesto principalmente de dos pasos que se describen detalladamente a continuación.

6.3.1. Primer Paso: Modelado del Comportamiento Base mediante CBR

El primer paso del algoritmo PI-SRL (***P**olicy **I**mprovement through **S**afe **R**einforcement **L**earning*) se trata de una aproximación para la clonación de comportamientos (*behavioral cloning*) que utiliza CBR para hacer que un agente se comporte de forma similar al comportamiento de un experto, que en nuestro caso será la política de comportamiento base π_T [Floyd *et al.*, 2008]. Las aproximaciones en aprendizaje por demostración pueden recibir diferentes nombres dependiendo de lo que se está tratando de aprender [Argall *et al.*, 2009]. Para prevenir inconsistencias en la terminología, en esta Tesis, se considera la clonación de comportamientos (también conocido como aprendizaje por imitación) como un área dentro del campo más extenso de aprendizaje por demostración en la cual el objetivo es reproducir/imitar la política de comportamiento subyacente del experto (en nuestro caso, el comportamiento base) [Peters *et al.*, 2010; Abbott, 2008].

Cuando se utiliza CBR para la clonación de comportamientos, un caso se construye a partir del estado que el agente recibe del entorno, y de la correspondiente acción que el comportamiento base selecciona en ese estado. El objetivo en este primer paso del algoritmo PI-SRL es imitar de forma apropiada el rendimiento del comportamiento base, π_T, mediante

un conjunto finito de casos almacenados en una base de casos. En este punto, surge la pregunta:

¿Cómo se contruye una base de casos, B, mediante las demostraciones de trayectorias suministradas con un comportamiento base, π_T, de forma que, al final del proceso de aprendizaje, la política resultante derivada de la base de casos, π_B, imite el comportamiento de π_T?

El comportamiento base, π_T, es una función que mapea estados a acciones $\pi_T : S \rightarrow A$, i.e., una función que dado un estado, $s_i \in S$, devuelve la acción correspondiente $a_i \in A$. Nosotros estamos interesados en construir una política π_B derivada de una base de casos (s_j, a_j), de forma que para un nuevo estado s_q se recupera el caso más cercano mediante distancia Euclídea, $dist(s_q, s_j)$, y se devuelve la acción a_j correspondiente a este caso recuperado. Intuitivamente, uno puede considerar que la política π_B se puede construir simplemente almacenando todos los casos (s_i, a_i) recuperados de una interacción entre la política π_T y el entorno durante un número limitado de episodios, K. Al finalizar estos K episodios, uno espera que la política resultante en la base de casos, π_B, sea capaz de imitar apropiadamente el comportamiento π_T. No obstante, una experimentación informal en el dominio del *Helicopter* demuestra que esta forma de proceder no genera los resultados esperados. En el dominio del *Helicopter*, después de $K = 100$ episodios y un número prohibitivo de 600.000 casos almacenados, la política que se deriva de esta base de casos, π_B^θ, produce continuamente colisiones del helicóptero siendo, por tanto, incapaz de imitar correctamente el comportamiento base π_T. En dominios con espacios de estados y acciones de naturaleza continua y de grandes dimensiones, esta aproximación puede requerir un número enorme de episodios, y por tanto, un número prohibitivo de casos para que π_B^θ sea capaz de imitar el comportamiento de π_T. Cabe destacar que imitar perfectamente el comportamiento base, π_T, en este tipo de dominios puede requerir almacenar un número infinito de casos. La figura 6.5 intenta explicar por qué pensamos que este proceso de aprendizaje no funciona como se pretende. La figura 6.5 muestra la región del espacio representada simplemente almacenando todos los casos producidos por la política de comportamiento base π_T de la forma $c = (s, a)$. Cada caso almacenado cubre un área definida del espacio (i.e., cada caso almacenado, los círculos rojos en la figura, representa el centroide correspondiente de la región de Voronoi).

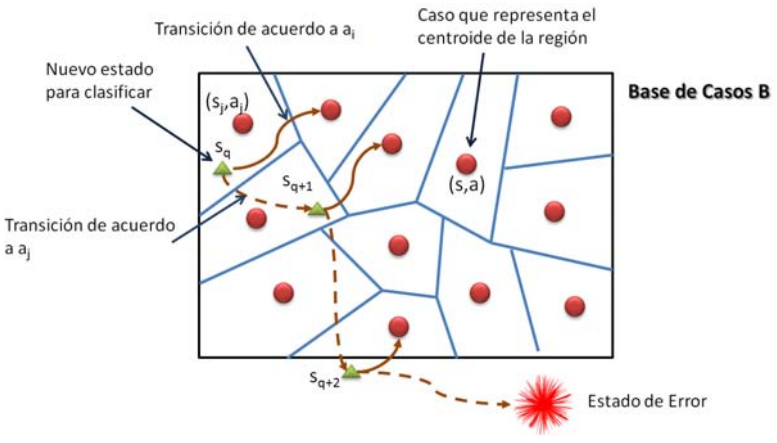


Figura 6.5: Efecto de almacenar todos los casos derivados del comportamiento base, π_T.

Si se utiliza la política previamente aprendida, π_B, cuando llega un nuevo estado s_q, se ejecuta la acción a_j correspondiente al caso $c_j = (s_j, a_j)$ donde la distancia Euclídea $dist(s_q, s_j)$ es mínima con respecto a cualquier otro caso almacenado. No obstante, si se utiliza la política π_T para proporcionar una acción en esta situación, s_q, se selecciona la acción a_i en lugar de la acción a_j con $a_i \neq a_j$. En este punto se dice que la política π_B clasifica el estado s_q como la **clase obtenida** a_j, mientras la política π_T clasifica el estado s_q como la **clase deseada** a_i, (puesto que la política π_T es la política que se trata de imitar), con $|a_i - a_j| > 0$. Además, se dice que $|a_i - a_j|$ es el **error de clasificación**. Las acciones a_j y a_i siempre serían exactamente las mismas, con $dist(s_q, s_j) = 0$ y $|a_i - a_j| = 0$, si la base de casos almacenara todos los posibles pares (s_i, a_i) que la política π_T es capaz de generar en el dominio, pero en un dominio continuo, estocástico y de grandes dimensiones, almacenar todos estos casos es imposible. La suma de todos estos errores de clasificación en un episodio conducen al agente a regiones inexploradas del espacio de casos, es decir, regiones donde el nuevo estado s_q recibido del entorno tiene una distancia Euclídea $dist(s_q, s_j) >> \theta$ con respecto al caso más cercano $c_j = (s_j, a_j)$ en B. Cuando se visitan estas regiones inexploradas, hay una gran diferencia entre la clase obtenida que se deriva de π_B y la clase deseada de π_T, i.e., $|a_i - a_j| >> 0$, y las probabilidades de visitar estados de error se incrementan considerablemente.

Por lo tanto, almacenar todos los casos $c = (s, a)$ generados mediante π_T no es suficiente para imitar correctamente su comportamiento. Por esto se introduce el algoritmo de la Tabla 6.1.

	Aproximación CBR para Imitación del Comportamiento $(\pi_T, \theta, \eta, K) \rightarrow B$
00	Dado el comportamiento base, π_T
01	Dado el umbral de densidad, θ
02	Dado el número máximo de casos, η
03	Dado un número máximo de episodios de aprendizaje a ejecutar, K
04	1. Inicializar la base de casos $B = \emptyset$
05	2. **Repetir**
06	Establecer $k = 0$
07	**Mientras** $k < K$ **hacer**
08	Calcular el caso $< s_c, a_c, 0 >$ más cercano al estado actual s_k
09	**Si** $\varrho^{\pi_B^\theta}(s_k) = 0$ **entonces** // *Mediante la ecuación 6.2*
10	Establecer $a_k = a_c$
11	**en otro caso**
12	Seleccionar a_k mediante el **comportamiento base** π_T
13	Crear un nuevo caso $c^{new} = (s_k, a_k, 0)$
14	$B := B \cup c^{new}$
15	Ejecutar a_k, y recibir el nuevo estado s_{k+1}
16	Establecer $k = k + 1$
17	**fin mientras**
18	**Si** $\|B\| > \eta$ **entonces**
19	Eliminar los $\eta - \|B\|$ casos menos frecuentemente usados de B
20	**hasta** que el criterio de parada se haga verdadero
21	3. **Devolver** B de la cual se deriva la **Política Segura Basada en Casos**, π_B^θ

Tabla 6.1: Algoritmo CBR para Imitación del Comportamiento.

En este primer paso del algoritmo la función de valor estado $V^{\pi_B^\theta}(s_i)$ se inicializa a 0 (línea 08 del algoritmo en la Tabla 6.1). El valor $V^{\pi_B^\theta}(s_i)$ para cada caso se inicializa en el

segundo paso del algoritmo que se describe en la sección 6.3.2. Además, este paso utiliza la función de riesgo basada en casos (ecuación 6.2) para identificar si un nuevo estado s_k se considera un estado de riesgo (línea 09). Si un nuevo estado no es de riesgo (i.e., se trata de un estado conocido, $s_k \in \Omega$), se utiliza una estrategia 1-NN (1-*Nearest Neighbor*) para determinar el caso del que se recupera la acción a_c que se ejecuta (línea 10). Si se trata de un estado desconocido el algoritmo selecciona la acción a_k utilizando el comportamiento base π_T, y se construye un nuevo caso, $c^{new} = (s_k, a_k, 0)$, que se añade a la base de casos (línea 14). El algoritmo utiliza las acciones que tiene almacenadas en la base de casos siempre que se encuentre en estados conocidos. Cuando se encuentra en un estado desconocido, solicita al comportamiento base, π_T, que genere una acción, y seguirá solicitando acciones al comportamiento base hasta que no regrese a una situación de estados conocidos. De esta forma, el algoritmo de la figura 6.1 continuamente incrementa la competencia de la base de casos almacenando nuevas experiencias. Transcurridos unos cuantos episodios, el algoritmo ejecutará prácticamente todas las acciones derivadas de la base de casos, y apenas se ejecutarán acciones suministradas por el comportamiento base, π_T, como se describirá detalladamente en la sección 6.5.

La base de casos tiene establecido un número máximo de casos η que puede almacenar. Existen una serie de razones por las cuales es conveniente limitar el número de casos almacenados. Las bases de casos con un número elevado de elementos incrementan el tiempo requerido para recuperar el caso más cercano cuando se presenta un nuevo estado. Este problema se puede solucionar parcialmente mediante la utilización de técnicas que reducen el tiempo de recuperación de casos, como por ejemplo, mediante la utilización de *kd-trees*, que también han sido utilizados en esta Tesis. No obstante, los *kd-trees* no logran reducir los costes de almacenamiento. Existen diferentes aproximaciones para eliminar los casos lo suficientemente *inútiles* de la base de casos durante el entrenamiento, por ejemplo los algoritmos IBx propuestos por Aha [Aha and Kibler, 1991] o cualquier aproximación basada en el prototipo más cercano [Fernandez and Isasi, 2008]. En cualquier caso, cuando el número de casos almacenados en B excede algún valor crítico, , $\|B\| > \eta$, de forma que la recuperación de casos dentro de cierta cantidad de tiempo no se puede garantizar es inevitable eliminar algunos casos. Una aproximación eficiente para abordar este problema es eliminar los casos menos frecuentemente utilizados en B (línea 18).

El resultado de este paso es una base de casos restringida, B, describiendo la Política Segura Basada en Casos, π_B^θ, que imita casi perfectamente el comportamiento base, π_T (línea 20). Formalmente, sea $U(\pi_T)$ una estimación de la utilidad del comportamiento base π_T, calculada mediante el promedio de la suma de refuerzos obtenidos en N_T episodios. Entonces $U(\pi_B^\theta) \leq U(\pi_T)$.

6.3.2. Segundo Paso: Mejora del Comportamiento Base Aprendido

En este paso del algoritmo PI-SRL se trata de mejorar, mediante la exploración segura del espacio de estados y de acciones, la Política Segura Basada en Casos π_B^θ aprendida en el paso anterior. En primer lugar, para cada caso $c_i \in B$, se calcula la función de valor-estado, $V^{\pi_B^\theta}(s_i)$, asociada a cada estado siguiendo la aproximación de Monte Carlo (MC) que se muestra en la Tabla 6.2.

El algoritmo presentado en la tabla 6.2 es similar en espíritu al algoritmo *First-Visit MC* para V^π [Sutton and Barto, 1998], aunque ha sido adaptado en esta Tesis para su funcionamiento con una base de casos. En el algoritmo mostrado en la tabla 6.2, los *returns* para cada estado $s_i \in B$ son acumulados y promediados, siguiendo la política de compor-

Algoritmo MC adaptado a CBR $(B, K) \rightarrow B$ *con* $V^{\pi^{\theta}_B}(s_i)$ *inicializados*	
00	Dada la base de casos, B
01	Dado un número máximo de episodios de aprendizaje a ejecutar, K
02	1. Inicializar, **Para cada** $c^i \in B$
03	$V(s) \leftarrow$ *arbitrariamente*
04	$Returns(s) \leftarrow$ *lista vacía*
05	2. **Mientras** $k < K$
06	Generar un episodio utilizando π^{θ}_B
07	**Para cada** s que aparece en el episodio con $< s, a, V(s) > \in B$
08	$R \leftarrow$ return siguiendo la primera ocurrencia de s
09	Añadir R to *Returns(s)*
10	*V(s)* $\leftarrow$ *promedio(Returns(s))*
11	Establecer $k = k + 1$
12	3. **Devolver** B

Tabla 6.2: Algoritmo de Monte Carlo para calcular la función de valor-estada de cada caso.

tamiento π^{θ}_B derivada de la base de casos B (línea 10 en el algoritmo de la tabla 6.2). El término "return" siguiendo la primera ocurrencia del estado s en el algoritmo, se refiere al retorno esperado, i.e., se refiere a la suma de refuerzos con descuento a futuro que se obtiene empezando desde ese estado s y ejecutando la política π^{θ}_B. El término "Return" en el algoritmo de la tabla 6.2 se refiere a una lista compuesta de los "return" del estado s en los diferentes episodios ejecutados. Una de las razones principales de utilizar el algoritmo de MC propuesto es que nos permite calcular fácil y rápidamente estimaciones para cada caso $c_i \in B$ de la función de valor-estado $V^{\pi^{\theta}_B}(s_i)$. Además, los métodos de MC para aprendizaje por refuerzo han demostrado su efectividad en una amplia variedad de tareas [Sutton and Barto, 1998]. Una vez que la función de valor-estado $V^{\pi^{\theta}_B}(s_i)$ se ha estimado para cada caso $c_i \in B$, se añade ruido aleatorio siguiendo una distribución Gaussiana a las acciones generadas mediante la Política Segura Basada en Casos π^{θ}_B. De esta forma, se lleva a cabo una exploración de los espacios de estados y acciones que permite encontrar nuevas políticas de comportamiento. El algoritmo utilizado para mejorar el comportamiento de la Política Segura Basada en Casos aprendida en el paso anterior se muestra en la tabla 6.3. El segundo paso del algoritmo PI-SRL está compuesto, a su vez, de cuatro pasos que se describen detalladamente a continuación.

- (a) **Inicialización**: En este paso se inicializa la lista que se utiliza para almacenar los casos que ocurren durante un episodio, y se inicializa el valor máximo del refuerzo acumulado obtenido en un episodio.

- (b) **Generación de Casos**: El algoritmo construye un caso por cada paso en un episodio. Por cada nuevo estado s_k recibido del entorno, se calcula el caso más cercano dentro de la base de casos empleando para ello la distancia Euclídea 6.1 (línea 10 en el algoritmo de la tabla 6.3). Para determinar el grado de riesgo del nuevo estado s_k percibido del entorno, se emplea la **Función de Riesgo Basada en Casos** (línea 11). Si $\varrho^{\pi^{\theta}_B}(s_k) = 0$, entonces $s_k \in \Omega$ (i.e., se trata de un estado **conocido**). En este caso, se ejecuta la acción a_k calculada mediante la ecuación 6.4, y se construye el nuevo caso $c^{new} =< s, a_k, V(s) >$ que se añade a la lista de casos ocurridos durante el episodio (línea 14). Cabe destacar que el nuevo caso $< s, a_k, V(s) >$ se construye reemplazando la acción a en el caso más cercano $< s, a, V(s) > \in B$ por la nueva

Algoritmo de Mejora de la Política $(B, \eta, \pi_T, \Theta, K) \rightarrow B$ *mejorada*

00 Dada la base de casos, B, y el máximo número de casos, η
01 Dado el comportamiento base, π_T
02 Dado el umbral de actualización de la base de casos, Θ
03 Dado un número máximo de episodios de aprendizaje a ejecutar, K
04 1. Establecer $maxTotalRwEpisode = 0$, el máximo de refuerzo acumulado obtenido en un episodio
05 2. **Repetir**
06 (a) ***Inicialización***:
07 **Establecer** $k = 0$, $listCasesEpisode \leftarrow \emptyset$, $totalRwEpisode = 0$
08 (b) ***Generación de Casos***:
09 **Mientras** $k < K$ **hacer**
10 Buscar el caso $< s, a, V(s) > \in B$ más cercano al estado actual s_k
11 **Si** $\varrho^{\pi_B^\theta}(s_k) = 0$ **entonces** // *estado conocido*
12 Seleccionar una acción a_k mediante la ecuación 6.4
13 Ejecutar la acción a_k
14 Crear un nuevo caso $c^{new} := (s, a_k, V(s))$
15 **en otro caso** // *estado desconocido*
16 Seleccionar la acción a_k mediante π_T
17 Ejecutar la acción a_k
18 Crear un nuevo caso $c^{new} := (s_k, a_k, 0)$
19 $totalRwEpisode := totalRwEpisode + r(s_k, a_k)$
20 $listCasesEpisode := listCasesEpisode \cup c^{new}$
21 Establecer $k = k + 1$
22 (c) ***Calcular la función de valor-estado para los estados desconocidos***:
23 **Para cada** caso c_i en $listCasesEpisode$
24 **Si** $\varrho^{\pi_B^\theta}(s_i) = 1$ **entonces** // *unknown state*
25 $return(s_i) := \sum_{j=n}^{k} \gamma^{j-n} r(s_j, a_j)$ // *n se refiere a la primera ocurrencia de* s_i *en el episodio*
26 $V(s_i) := return(s_i)$
27 (d) ***Actualizar los casos en B utilizando la experiencia recuperada***:
28 **Si** $totalRwEpisode > (maxTotalRwEpisode - \Theta)$ **entonces**
29 $maxTotalRwEpisode := \boldsymbol{max}(maxTotalRwEpisode, totalRwEpisode)$
30 **Para cada** caso $c_i =< s_i, a_i, V(s_i) >$ en $listCasesEpisode$
31 **Si** $\varrho^{\pi_B^\theta}(s_i) = 0$ **entonces** // *estado conocido*
32 Calcular el caso $< s_i, a, V(s_i) > \in B$ correspondiente al estado s_i
33 Calcular $\delta = r(s_i, a_i) + \gamma V(s_{i+1}) - V(s_i)$
34 **Si** $\delta > 0$ **entonces**
35 Reemplazar el caso $< s_i, a, V(s_i) > \in B$ con el caso $< s_i, a_i, V(s_i) > \in listCasesEpisode$
36 $V(s_i) = V(s_i) + \alpha\delta$
37 **en caso contrario** // *estado desconocido*
38 $B := B \cup c_i$
39 **Si** $\|B\| > \eta$ **entonces**
40 Eliminar los $\eta - \|B\|$ menos frecuentemente usados en B
41 **hasta** que el criterio de parada se haga verdadero
42 3. **Devolver** B

Tabla 6.3: Descripción del segundo paso del algoritmo PI-SRL

acción a_k resultante de aplicar ruido aleatorio siguiendo una distribución Gaussiana a la acción a mediante la ecuación 6.4. De esta forma, el algoritmo sólo produce cambios suaves en los casos de la base de casos B, puesto que $a_k \sim a$. Si por el contrario $\varrho^{\pi_B^\theta}(s_k) = 1$, el estado $s_k \in \Upsilon$, i.e., s_k es un estado **desconocido** (línea 15).

Cuando el estado se encuentra en un estado **desconocido**, se ejecuta la acción a_k sugerida por el **comportamiento base**, π_T, que define un comportamiento seguro aunque subóptimo (línea 16). Se construye un nuevo caso $< s_k, a_k, 0 >$ que se añade a la lista de casos que ocurren durante la ejecución del episodio. Hasta que el agente no se encuentre en un estado **conocido**, se mantiene ejecutando el **comportamiento base**, π_T. Por último, se calcula el refuerzo acumulado total obtenido en el episodio, donde $r(s_k, a_k)$ es el refuerzo inmediato que se recibe cuando se ejecuta la acción a_k en el estado s_k (línea 19).

- (c) **Calcular la función de valor-estado para los estados desconocidos**. En este paso se calcula el valor de la función de valor-estado para los estados etiquetados como estados **desconocidos** en el paso anterior. En el paso anterior, la función de valor-estado para estos estados se inicializaba a 0 (línea 18). Para calcular este valor, el algoritmo procede de forma similar que en el algoritmo *First Visit MC* mostrado en la tabla 6.2. No obstante, en este caso se calcula el retorno (*return*) para cada estado **desconocido** aunque no se promedia, puesto que estamos considerando únicamente un episodio (líneas 25 y 26). El retorno de cada s_i se calcula teniendo en cuenta la primera visita al estado s_i en el episodio (cada ocurrencia del estado s_i en el episodio se llama visita al estado s_i), aunque este estado s_i podría aparecer más veces en el resto del episodio considerado.

- (d) **Actualizar los casos en B utilizando la experiencia recuperada**. Las actualizaciones en la base de casos B se realizarán considerando únicamente los episodios con un refuerzo total acumulado similar al mejor episodio encontrado hasta el momento empleando para ello el umbral Θ (línea 28). De esta forma, a este paso del algoritmo se le suministran buenas secuencias de experiencias puesto que se ha demostrado que buenas secuencias causan que el agente converja a una política estable y útil, mientras que malas secuencias pueden causar que el agente converja a una política pobre e inestable [Wyatt, 1997]. Esta selección de *buenos* episodios para las actualizaciones previene la degradación del comportamiento inicial de la base de casos B calculada en el primer paso del algoritmo, evitando realizar actualizaciones con *malos* episodios o episodios con fallos. De nuevo, el algoritmo itera por cada caso $c_i = (s_i, a_i, V(s_i)) \in listCasesEpisode$ (línea 30). Si s_i es un **estado conocido** (línea 31), se calcula el caso $< s_i, a, V(s_i) > \in B$ correspondiente al estado s_i (línea 32). Cabe destacar que el caso $c_i = (s_i, a_i, V(s_i)) \in listCasesEpisode$ fue construido en la línea 14 del algoritmo reemplazando la acción a correspondiente al caso $< s_i, a, V(s_i) > \in B$ por la nueva acción a_i resultante de aplicar ruido aleatorio siguiendo una distribución Gaussiana a la acción a mediante la ecuación 6.4.

  Posteriormente se calcula el error de diferencia temporal (*TD-error*), δ (línea 33). Si $\delta > 0$, la ejecución de la acción a_i ha resultado en un cambio positivo en el valor del estado, y entonces esta acción podría conducir a un mayor valor de la función de valor-estado y por lo tanto a una mejor política. van Hasselt and Wiering [van Hasselt and Wiering, 2007] también actualizan la función de valor y las acciones únicamente utilizando acciones que potencialmente conducen a un mayor valor de la función de valor-estado $V(s_i)$. Si el error de diferencia temporal δ es positivo, se juzga que la acción a_i es una buena acción y se refuerza la selección de esta acción en este estado para que sea tenida en cuenta en futuros pasos del algoritmo. En nuestro caso, esto se hace mediante la actualización de la salida del caso $< s_i, a, V(s_i) > \in B$ a a_i (línea

35). Por lo tanto, las actualizaciones de los casos en la base de casos sólo se producen cuando el error de diferencia temporal es positivo. Esta forma de proceder es similar a las actualizaciones *reward-inaction* en aprendizaje de autómatas [Narendra and Thathachar, 1974; Narendra and Thathachar, 1989], que utilizan el signo del error de diferencia temporal como medida de "éxito". PI-SRL sólo actualiza la base de casos cuando realmente se observan mejoras potenciales de la política. Esto evita que el aprendizaje se vea ralentizado con la aparición de *plateaus* en el espacio de la función de valor y el error de diferencia temporal es pequeño. Se ha demostrado, incluso, que esta forma de actualizaciones puede dar lugar a mejores políticas que cuando el tamaño del paso depende de la magnitud del error de diferencia temporal [van Hasselt and Wiering, 2007].

Destacar además, que estas actualizaciones sólo producen cambios suaves en los casos de la base de casos B puesto que una acción a sólo se reemplaza si a_i produce un mayor $V(s_i)$, y $a_i \sim a$. Estas actualizaciones también pueden ser vistas como una forma de *risk-seeking* puesto que sólo se premian las transiciones a estados que prometen un mayor valor de la función de valor-estado que en media [Mihatsch and Neuneier, 2002]. Por último, de esta forma también se previene la degradación de los casos en la base de casos B asegurando que los reemplazos sólo se llevan a cabo cuando una acción potencialmente conduce a un mayor $V(s_i)$.

Si por el contrario s_i es un estado desconocido, el caso c_i asociado se añade a la base de casos B (línea 38). Por último, en caso de que sea necesario se eliminan algunos casos de la base de casos B (línea 40). En la literatura se han propuesto complejos cálculos para determinar qué casos son los más susceptibles de ser eliminados. Forbes y Andre [Forbes and Andre, 2002] sugieren eliminar aquellos casos que contribuyen en menor medida en la aproximación de la función. Driessens and Ramon [Driessens and Ramon, 2003] sugieren una aproximación basada en el error, que propone la eliminación de los casos que contribuyen en mayor medida al error de predicción de otros ejemplos. El problema principal de todos estos cálculos sofisticados es la complejidad. La determinación de qué caso(s) deben ser eliminados involucra el cálculo de una "puntuación" para cada caso $c_i \in B$, que requiere al menos una recuperación y una regresión respectivamente, para cada $c_j \in B$, $j \neq i$. Este proceso repetido barre completamente la base de casos cada vez, lo que induce una enorme carga computacional. Gabel y Riedmiller [Gabel and Riedmiller, 2005] calculan una "puntación" para cada caso $c_i \in B$, que requiere calcular el conjunto de k vecinos más cercanos a c_i. Por lo tanto, estas aproximaciones no son aconsejables en sistemas con unas necesidades temporales limitadas, en dominios con espacios de estados y acciones continuos y de grandes dimensiones, que normalmente requerirán grandes bases de casos, como es el caso de los dominios utilizados en esta Tesis. En cambio, en esta Tesis se propone eliminar nuevamente los casos menos frecuentemente utilizados de la base de casos. Esta idea parece intuitiva, puesto que los casos menos utilizados normalmente contendrán peores estimaciones correspondientes a la función de valor de un estado, aunque esta estrategia también podría llegar a "olvidar" algunos casos relevantes adquiridos en el pasado. No obstante, el algoritmo PI-SRL funciona correctamente en los dominios de experimentación propuestos cuando se emplea esta estrategia como se demuestra en la sección 6.5. Por lo tanto, la habilidad de "olvidar" estados conocidos inútiles descrita en la sección 6.2.2 se debe a que el algoritmo elimina los $\|B\| - \eta$ casos menos frecuentemente usados en B.

6.4. Configuración de Parámetros

Dado que se requiere minimizar el número de daños, es fundamental minimizar el tuneado de parámetros mediante la definición de valores estándar, o mediante mecanismos que permitan definir sin riesgo valores adecuados. En el caso del algoritmo PI-SRL esto se traduce en decidir un conjunto apropiado de valores para el umbral θ, el parámetro de riesgo σ, el umbral de actualización Θ, y el máximo número de casos permitidos en la base de casos, η. En el caso del parámetro θ, un valor demasiado pequeño puede producir que la base de casos se llene por completo muy rápido, forzando la eliminación de casos casi constantemente. Por el contrario, un valor demasiado elevado del parámetro θ puede restringir demasiado la adición de nuevos casos a la base de casos, lo que puede influir negativamente en el rendimiento final de la política aprendida. En el caso del parámetro de riesgo σ, valores muy elevados pueden causar daños en el agente continuamente, mientras que la utilización de valores pequeños, aunque seguros, pueden no ser suficientes para explorar los espacios de estados y acciones y encontrar nuevos y mejores comportamientos. Al contrario que los parámetros θ y σ, el parámetro referido al umbral de actualización Θ no está relacionado directamente con el concepto de riesgo, aunque sí está directamente relacionado con el rendimiento final del algoritmo. El parámetro Θ se utiliza para determinar cómo de bueno debe ser un episodio con respecto al mejor episodio encontrado hasta el momento, puesto que sólo los mejores episodios son utilizados para realizar actualizaciones sobre la base de casos B (influyendo en la velocidad de convergencia y en el rendimiento del algoritmo). Si se establece un valor muy pequeño para Θ, el número de actualizaciones en la base de casos puede ser insuficiente para mejorar el rendimiento del comportamiento base. Por último, un valor muy elevado del parámetro η permite la existencia de bases de casos muy grandes que incrementan la carga computacional en las recuperaciones, degradando la eficiencia final del sistema. Por el contrario, un valor muy pequeño de η puede restringir excesivamente el tamaño de la base de casos, y afectar negativamente al rendimiento final del algoritmo. En esta sección se ofrece una perspectiva sólida para definir automáticamente el valor de todos estos parámetros. La configuración de los parámetros que se propone se describe como un conjunto de heurísticas que han demostrado que funcionan correctamente en una amplia variedad de dominios de diversa naturaleza (Section 6.5).

6.4.1. Parámetro θ

Se trata de un parámetro dependiente del dominio, relacionado con el *tamaño medio* de las acciones. En esta Tesis, el valor de este parámetro se establece mediante el cálculo de la distancia media entre estados durante una interacción del comportamiento base, π_T, con el entorno. La ejecución de la política de comportamiento π_T suministra una secuencia de estados-acciones de la forma $s_1 \rightarrow a_1 \rightarrow s_2 \rightarrow a_2 \rightarrow \ldots \rightarrow s_n$. El parámetro θ se calcula utilizando esta secuencia mediante la ecuación 6.5.

$$\theta = \frac{dist(s_1, s_2) + \ldots + dist(s_{n-1}, s_n)}{n-1} \tag{6.5}$$

Cabe destacar que el valor de este parámetro se calcula a partir de ejecuciones seguras, i.e., no se producen daños para su estimación, aunque su cálculo sí conlleva un coste de ejecución.

6.4.2. Parámetro σ

Varios autores coinciden en que es imposible evitar por completo todos los accidentes en tareas con riesgo [Moldovan and Abbeel, 2012; Geibel and Wysotzki, 2005]. El algoritmo PI-SRL es completamente seguro si sólo se ejecuta el primer paso correspondiente al modelado del comportamiento base mediante CBR, o si se establece un valor de $\sigma = 0$ durante el segundo paso de mejora del comportamiento base aprendido. No obstante, si se procede de esta forma, el rendimiento final del algoritmo estaría fuertemente limitado por el comportamiento base utilizado. Para mejorar el rendimiento del comportamiento base, es inevitable llevar a cabo el proceso de exploración definido en el segundo paso del algoritmo, que permite mejorar el rendimiento del agente más allá del comportamiento base. Puesto que no se tiene un conocimiento completo del dominio y su dinámica, es inevitable durante este segundo paso visitar regiones desconocidas del espacio de estados donde el agente podría alcanzar estados de error. No obstante, es posible ajustar el parámetro de riesgo, σ, para determinar el nivel del riesgo que se está dispuesto a asumir durante este proceso de exploración. En esta Tesis se propone comenzar con valores bajos de σ, que suponen asumir poco riesgo, e ir incrementando gradualmente su valor, lo que supone incrementar las probabilidades de visitar estados de error. En esta Tesis se comienza con un valor de $\sigma = 9 \times 10^{-7}$, y este valor se incrementa iterativamente hasta que se obtiene una política óptima o cercana a la óptima, o hasta que el número de fallos sea elevado.

6.4.3. Parámetro Θ

El valor de este parámetro se establece de forma relativa al máximo refuerzo acumulado obtenido en el mejor episodio que se ha llevado a cabo hasta el momento. En esta Tesis, se utiliza un valor del $\Theta = 5\,\%$ del máximo refuerzo acumulado obtenido.

6.4.4. Parámetro η

La estimación del número máximo de casos permitidos en la base de casos, η, se establece como la estimación del número de casos requeridos para imitar correctamente el comportamiento base, π_T. En esta sección, se explica el proceso llevado a cabo para calcular este valor. La Figura 6.6 muestra las trayectorias (secuencias de estados) generadas mediante el comportamiento base, π_T, en tres dominios con un grado de estocasticidad diferente: un dominio determinista, un dominio ligeramente estocástico, y un dominio muy estocástico. Para cada dominio, se muestran las secuencias de estados producidas en diferentes episodios mediante π_T. Estas secuencias son $\{s_{00}, s_{01}, s_{02}, \ldots, s_{0n}\}$, $\{s_{00}, s_{11}, s_{12}, \ldots\ , s_{1n}\}$,..., $\{s_{00}, s_{m1}, s_{m2}, \ldots, s_{mn}\}$, donde s_{ji} se corresponde con el estado i-th en la secuencia de estados resultante en el episodio j. En el dominio determinista, la ejecución de m episodios diferentes utilizando π_T siempre genera la misma secuencia de estados. En este caso, se establece el número máximo de casos, η, como $\eta = n$, donde n es la longitud del episodio, i.e., se asume que todos los casos que se producen en el episodio son almacenados. Cabe destacar que procediendo de esta forma en este dominio se tendría que $U(\pi_B^\theta) = U(\pi_T)$, es decir, la política π_B^θ consigue imitar perfectamente el comportamiento de la política π_T.

En dominios ligeramente estocásticos, las trayectorias generadas en m episodios diferentes, son ligeramente diferentes. En este caso, se procede de la siguiente manera. Se considera que al principio la base de casos está vacía. Además, se considera que todos los estados $\{s_{00}, s_{01}, s_{02}, \ldots, s_{0n}\}$ correspondientes a la primera secuencia de estados producidos en el dominio se almacenan en la base de casos. Para cada dominio, se ejecutan m episodios

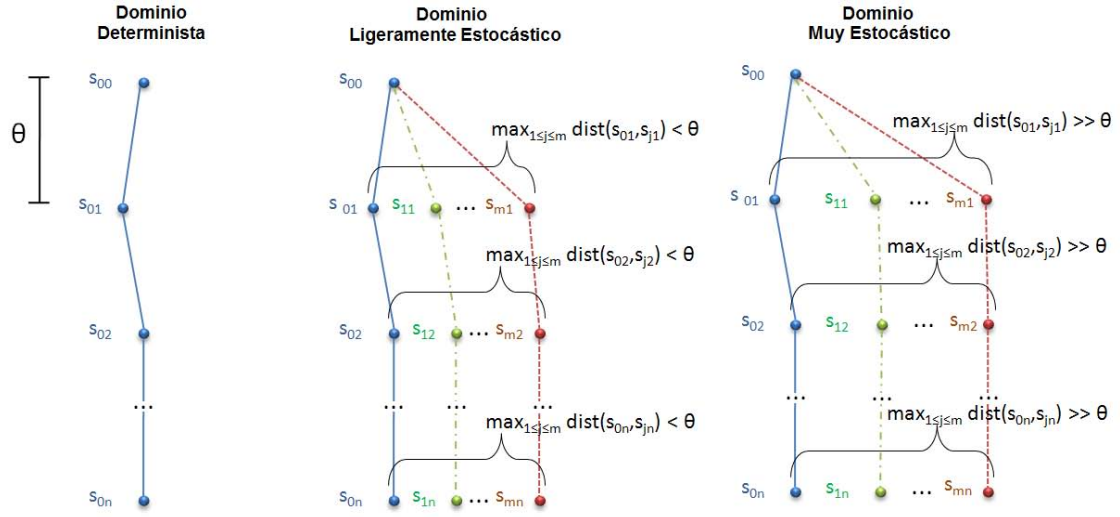


Figura 6.6: Trayectorias generadas mediante el comportamiento base π_T en dominios con diferentes niveles de estocasticidad.

diferentes, obteniendo m secuencias diferentes de estados. Después de la ejecución de estos m episodios diferentes, se calcula la distancia máxima entre el estado i-th correspondiente a la primera secuencia de estados (añadida previamente a la base de casos), y el estado i-th producido en la secuencia de estados j, i.e., $\text{máx}_{1\leq j\leq m}\, d(s_{0i}, s_{ji})$. En este dominio ligeramente estocástico, esta distancia máxima no excede el umbral θ en ningún caso, i.e., $\text{máx}_{1\leq j\leq m}\, d(s_{0i}, s_{ji}) < \theta$. En este dominio, se puede considerar que el estado i-th en la secuencia de estados j tiene, al menos, un vecino con distancia menor que θ (el correspondiente al estado s_{0i}) y, por lo tanto, el estado i-th en la secuencia de estados j no se añade a la base de casos. En este dominio por tanto, también se establece el número máximo de casos permitidos, η, como $\eta = n$, siendo n la longitud de la secuencia de estados de un episodio.

Por el contrario, en dominios muy estocásticos, esta máxima distancia excede enormemente el umbral θ en todos los pasos de la secuencia de estados, i.e., $\text{máx}_{1\leq j\leq m}\, d(s_{0i}, s_{ji}) >> \theta$, y por lo tanto se considera que nuevos casos son añadidos a la base de casos. En este caso, se estima el número total de casos que serán añadidos a la base de casos de la siguiente forma. Para cada estado i-th en la primera secuencia de estados, se estima el número de casos que serán añadidos como $\left\lfloor \frac{\text{máx}_{1\leq j\leq m}\, d(s_{0i}, s_{ji})}{\theta} \right\rfloor$, es decir, se calcula el número de intervalos en el rango $[0, \text{máx}_{1\leq j\leq m}\, d(s_{0i}, s_{ji})]$ con ancho θ (el parámetro utilizado para determinar si un caso es añadido o no a la base de casos). Por lo tanto, el número estimado de casos añadidos a la base de casos teniendo en cuenta todos los estados de la secuencia de estados se calcula como $\sum_{i=0}^{n} \left\lfloor \frac{\text{máx}_{1\leq j\leq m}\, d(s_{0i}, s_{ji})}{\theta} \right\rfloor$. Por último, el número máximo de casos permitidos en la base de casos se calcula mediante la ecuación 6.6.

$$\eta = n + \left(\sum_{i=0}^{n} \left\lfloor \frac{\text{máx}_{1\leq j\leq m}\, d(s_{0i}, s_{ji})}{\theta} \right\rfloor \right) \tag{6.6}$$

En el dominio determinista el elemento de la ecuación 6.6 $\sum_{i=0}^{n} \left\lfloor \frac{\text{máx}_{1\leq j\leq m}\, d(s_{0i}, s_{ji})}{\theta} \right\rfloor$ es igual a 0, y por lo tanto $\eta = n$. El valor de este elemento se incrementa a medida que

se incrementa la estocasticidad del entorno, puesto que se asume que la estocasticidad del entorno incrementa el número de casos que son requeridos. Por último, si el número de casos necesario fuera muy elevado o cercano al infinito, se puede incrementar el valor del umbral θ para hacer más restrictiva la adición de nuevos casos a la base de casos. No obstante, este incremento puede tener efectos adversos en el rendimiento final del algoritmo.

Por otro lado, también es posible determinar el máximo número de casos que se pueden procesar dentro de un paso de aprendizaje. En dominios como la RoboCup, y los dominios empleados en la evaluación del algoritmo PI-SRL en esta Tesis, el agente lleva a cabo una acción dentro de un intervalo discreto de tiempo, llamado paso o ciclo cuya duración se establece a t_c. Si el agente no ejecuta una acción en cada uno de estos ciclos, entonces estará en desventaja con respecto a los agentes que sí lo hacen. Durante cada uno de estos ciclos el agente lleva a cabo tres tareas fundamentales:

1. Percibir el estado en el cual se encuentra.
2. Llevar a cabo las tareas de aprendizaje y decidir qué acción llevar a cabo en esta nueva situación.
3. Ejecutar la acción seleccionada.

La tarea número 2 es la más importante, pero dado que no es la única tarea que se debe llevar a cabo dentro de un ciclo, se le asigna la mitad del tiempo disponible para cada ciclo, $\frac{t_c}{2}$ [Floyd *et al.*, 2008]. En esta tarea, gran parte del tiempo se corresponde con el tiempo t_r que se dedica a recuperar el caso más cercano correspondiente al nuevo estado percibido del entorno. De esta forma, se puede establecer el tamaño máximo de la base de casos como el máximo número de casos, max_c, que hace que se siga cumpliendo la condición $t_r < \frac{t_c}{2}$. De esta forma, si el número de casos estimado, η, requerido para imitar el comportamiento base es $\eta > max_c$, es probable que el algoritmo PI-SRL no sea aplicable al dominio en cuestión, debido a la gran cantidad de casos necesarios que hay que almacenar en la base de casos. En este caso, se puede hacer más restrictiva la adición de casos a la base de casos como se comentaba anteriormente, o se puede emplear algún método de selección de características como el presentado en el capítulo anterior con el fin de reducir el tiempo de recuperación, t_r. No obstante, la utilización de técnicas *kd*-tree, también empleadas en esta Tesis, hacen que el valor max_c sea un valor lo suficientemente grande en la mayoría de los dominios como para garantizar la amplia aplicabilidad del algoritmo PI-SRL. De hecho, en los dominios de evaluación empleados en la sección 6.5, el valor max_c superaba ampliamente el valor de η utilizado como límite en el número de casos en la base de casos.

6.5. Experimentos y Resultados

En esta sección se reportan los resultados obtenidos por el algoritmo PI-SRL en cuatro dominios de experimentación diferentes que crecen en complejidad en cuanto al número de variables de estado y de acciones: el *automatic car parking problem* (Sección 4.2.1), el *Cart-Pole* (Sección 4.2.2), el *helicopter* (Sección 4.2.3), y el simulador empresarial SIMBA (Sección 4.2.5). El algoritmo PI-SRL trata de encontrar las políticas óptimas o políticas cercanas al óptimo a la vez que trata de minimizar durante los procesos de aprendizaje el número de colisiones del coche, los desbalanceos del péndulo, las colisiones del helicóptero y las bancarrotas de la compañía gestionada, respectivamente en los cuatro dominios de experimentación propuestos. Estos cuatro dominios son dominios estocásticos. El dominio

del *helicopter* y del simulador empresarial, SIMBA, son dominios estocásticos por sí mismos, y además son dominios generalizados. En el caso del *automatic car-parking problem* y el *Cart-Pole*, se ha añadido intencionadamente ruido aleatorio siguiendo una distribución Gaussiana a las acciones y la función de refuerzo, para transformarlos en dominios estocásticos. Los resultados del algoritmo PI-SRL en estos cuatro dominios son comparados con los resultados de una aproximación basada en aprendizaje por refuerzo evolutivo [Martín and de Lope Asiaín, 2009] (ganador de la competición de aprendizaje por refuerzo edición 2009 en el dominio del helicóptero) y una aproximación sensible al riesgo (*Risk-Sensitive*) [Geibel and Wysotzki, 2005]. La técnica evolutiva introduce varios comportamientos base libres de fallos aunque subóptimos en la población inicial de redes de neuronas, lo que garantiza la rápida convergencia del algoritmo a políticas cercanas al óptimo e, indirectamente, se minimiza el número de daños que sufren los agentes. De hecho, el ganador de la competición en el dominio del *helicopter* es el agente capaz de obtener el mayor refuerzo acumulado, teniendo en cuenta que cada colisión involucra un refuerzo negativo muy elevado.

Por el contrario, la aproximación sensible al riesgo define el riesgo como la probabilidad $\rho^\pi(s)$ de visitar un estado de error (por ejemplo, el helicóptero se choca y el control del agente finaliza) empezando a partir de algún estado inicial s. En este caso, se define una nueva función de valor-estado: la suma ponderada de la probabilidad de riesgo y la función de valor-estado original (Ecuación 6.7).

$$V_\xi^\pi(s) = \xi V^\pi(s) - \rho^\pi(s) \tag{6.7}$$

El parámetro $\xi \geq 0$ determina la influencia de los valores $V^\pi(s)$ comparados con los valores $\rho^\pi(s)$. Para $\xi = 0$, V_ξ^π se corresponde con el cálculo de las políticas de riesgo mínimo, puesto que sólo la función de riesgo se tiene en cuenta en el cálculo de la nueva función de valor definida. Para valores más grandes de ξ la función de valor original multiplicada por ξ domina la suma ponderada. Conviene destacar que en la descripción del algoritmo original, los autores sólo consideran dominios con un conjunto finito (discreto) de acciones. Por este motivo, en esta Tesis se ha adaptado su algoritmo para que funcione también en dominios con estados de acciones continuos. La adaptación consiste en utilizar nuevamente técnicas CBR para aproximar tanto la función de riesgo como la función de valor-estado original, que intervienen en la definición de la nueva función de valor V_ξ^π. Además, en cada dominio se han utilizado tres valores diferentes del parámetro ξ para modificar la influencia de los valores V^π sobre los valores ρ^π.

Por último, conviene mencionar, que uno de los comportamiento base utilizado para inicializar la población inicial del algoritmo basado en aprendizaje por refuerzo evolutivo es exactamente el mismo que el utilizado para llevar a cabo el primer y segundo paso del algoritmo PI-SRL. También conviene destacar que la base de casos utilizada por la aproximación sensible al riesgo no parte de un desconocimiento absoluto de la tarea que se trata de aprender, sino que es inicializada con la Política Segura Basada en Casos π_B^θ construida en el primer paso del algoritmo PI-SRL. Esto hace que la comparación de los rendimientos de los algoritmos sea lo más equilibrada posible, pero siempre teniendo en cuenta que se trata de diferentes algoritmos y cada uno hace su propio uso del comportamiento base. En cada dominio, se han establecido diferentes niveles de riesgo modificando el valor del parámetro de riesgo, σ, siguiendo el procedimiento descrito en la sección 6.4.

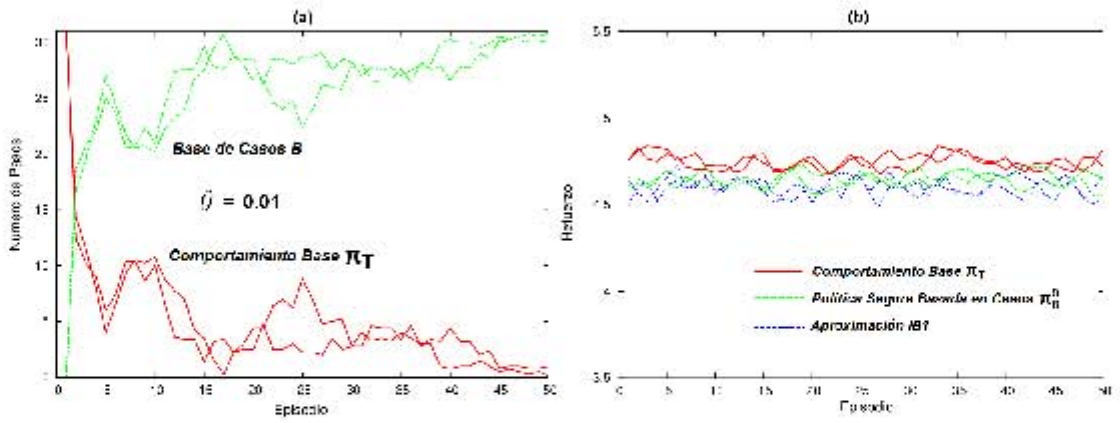


Figura 6.7: (a) Número de pasos por episodios ejecutados mediante la base de casos B y el comportamiento base π_T. (b) Refuerzo total acumulado por episodio obtenido por π_T, la Política Segura Basada en Casos π_B^θ y la aproximación *IBL*.

6.5.1. *Automatic Car Parking Problem*

El dominio del *automatic car parking problem* se presentó en la sección 4.2.1. Como se describió entonces, se ha añadido ruido aleatorio a los actuadores y a la función de refuerzo siguiendo una distribución Gaussiana con una desviación de 0.1, para convertir el dominio en un dominio estocástico. Es importante aclarar que el ruido añadido a los actuadores para convertir el dominio en estocástico es independiente del ruido aleatorio (parámetro de riesgo) utilizado para explorar los espacios de estados y acciones en el segundo paso del algoritmo PI-SRL. En este caso, el ruido aleatorio siguiendo una distribución Gaussiana con desviación σ utilizado para explorar se sumará al ruido previamente añadido a los actuadores.

Para aprender de formar segura una política de comportamiento adecuada en este dominio, en primer lugar se ejecuta el primer paso del algoritmo PI-SRL, el paso de *Modelado del Comportamiento Base* mediante CBR. En este paso se trata de aprender la Política Segura Basada en Casos, π_B^θ, a partir de las demostraciones de la tarea realizadas por el comportamiento base, π_T, tal como se describió en la sección 6.3.1. Los parámetros θ y η en este paso se calculan siguiendo los procedimientos descritos en la sección 6.4, y sus valores son 0.01 y 207 respectivamente. En este dominio se utiliza un comportamiento base, π_T, cuyo refuerzo total acumulado por episodio tiene un promedio de 4.75. La figura 6.7 (a) muestra gráficamente la ejecución del primer paso del algoritmo *Modelado del Comportamiento Base*.

La gráfica en la figura 6.7 (a) muestra dos procesos de aprendizaje diferentes y para cada uno, muestra el número de pasos ejecutados mediante el comportamiento base, π_T (líneas sólidas rojas), y el número de pasos por episodio que son ejecutados utilizando los casos de la base de casos B (líneas discontinuas verdes). Al principio de la ejecución de este paso de *Modelado del Comportamiento Base*, la base de casos B se encuentra vacía, y todos los pasos son ejecutados utilizando el comportamiento base, π_T. A medida que el proceso de aprendizaje avanza, se van añadiendo nuevos casos a la base de casos B y se va aprendiendo la Política Segura Basada en Casos π_B^θ. Alrededor de los episodios 40-50, prácticamente todos los pasos son ejecutados utilizando los casos almacenados en la base de casos B, y prácticamente se dejan de solicitar acciones del comportamiento base

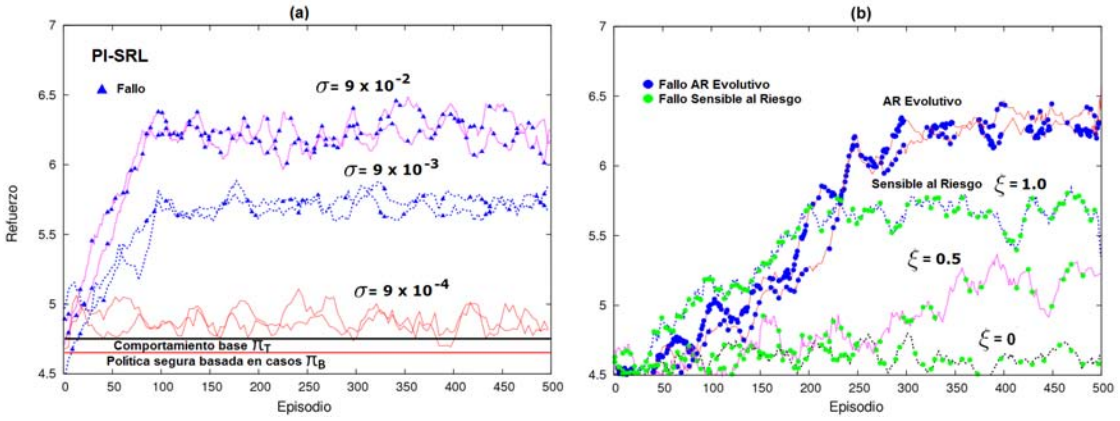


Figura 6.8: (a) Refuerzo acumulado total por episodio obtenido por PI-SRL mediante diferentes configuraciones de riesgo (σ). (b) Refuerzo acumulado total por episodio obtenido mediante la aproximación basada en aprendizaje por refuerzo evolutivo y la aproximación sensible al riesgo. En todos los casos, se marca cuando el episodio finaliza con fallo.

π_T, lo que significa que la base de casos segura se ha aprendido correctamente. Conviene destacar que los procesos de aprendizaje que se muestran en la figura 6.7 (a) se han llevado a cabo sin fallos (i.e., sin colisiones del coche contra la pared o el obstáculo). En otras palabras, se ha logrado imitar correctamente el comportamiento base, π_T, de forma segura sin colisiones. La figura 6.7 (b) muestra el refuerzo acumulado total por episodio obtenido mediante tres políticas de comportamientos diferentes. La primera (línea sólida roja) se trata del comportamiento base, π_T, la segunda (línea discontinua verde) se corresponde con la Política Segura Basada en Casos π_B^θ que se deriva de la base de casos B que se ha aprendido, y la tercera (líneas discontinuas azules) pertenece a una aproximación basada en técnicas IBL que consisten en almacenar casos en memoria. En concreto, en las técnicas IBL, los nuevos elementos recibidos se clasifican examinando los casos almacenados en memoria para determinar el caso más similar de acuerdo a alguna métrica de similitud (en esta Tesis se emplea la distancia Euclídea). La clasificación de ese vecino más cercano se utiliza también para clasificar el nuevo elemento recibido [Aha and Kibler, 1991]. En la aproximación IBL presentada en la figura 6.7 (b), el proceso de construcción de la base de casos se ha llevado a cabo simplemente almacenando todos los casos producidos durante una interacción entre el comportamiento base, π_T, y el entorno de 50 episodios. Por este motivo, se considera que esta aproximación de IBL pertenece a la clase de algoritmos IB1 en el sentido de que todos los casos son almacenados durante el proceso de entrenamiento [Aha and Kibler, 1991]. La figura 6.7 (b) muestra que la Política Segura Basada en Casos, π_B^θ, imita casi perfectamente al comportamiento base π_T. En este dominio, la aproximación IBL utilizada también alcanza un rendimiento similar a las anteriores.

La figura 6.8 (a) muestra los resultados para diferentes valores del parámetro de riesgo, σ, obtenidos por el algoritmo PI-SRL en el segundo paso de *Mejora del Comportamiento Base Aprendido* 6.3.2. Para cada valor del parámetro de riesgo se muestran dos procesos de aprendizaje diferentes. Cuando un episodio finaliza con fallo (i.e., el coche ha colisionado con una pared o con el obstáculo), se indica mediante los triángulos azules. Los procesos de aprendizaje en la figura 6.8 (a) demuestran que el número de fallos se incrementan conforme se incrementa el parámetro de riesgo σ. Para un valor bajo de riesgo ($\sigma = 9 \times 10^{-4}$) no se producen fallos, aunque apenas se consigue mejorar el rendimiento del comportamiento

base empleado. Experimentos adicionales demuestran que incrementado el valor de σ a valores más elevados de $\sigma = 9 \times 10^{-2}$ se obtiene un número más elevado de fallos, aunque el rendimiento final del algoritmo no se mejora. La figura 6.8 (b) muestra los resultados obtenidos mediante la aproximación basada en aprendizaje por refuerzo evolutivo, y la aproximación sensible al riesgo para diferentes valores del parámetro ξ. El número de fallos obtenidos por la aproximación basada en aprendizaje por refuerzo evolutivo es mayor que el obtenido por el algoritmo PI-SRL, aunque el rendimiento final es muy similar. En el caso de la aproximación sensible al riesgo, el rendimiento es más elevado cuando se utiliza un valor elevado de ξ ($\xi = 1{,}0$) lo que equivale a maximizar la función de valor-estado sin prácticamente tener en cuenta el valor de la función de riesgo ρ, aunque el agente está continuamente colisionando con las paredes o el obstáculo.

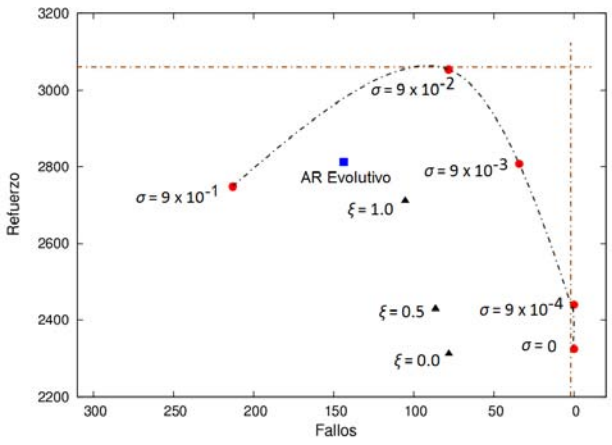


Figura 6.9: Número de fallos (colisiones del coche) y el refuerzo total acumulado durante 500 episodios obtenidos por cada aproximación.

La figura 6.9 muestra el número medio de fallos y el refuerzo total acumulado obtenido por cada aproximación durante 500 episodios. Los círculos rojos se corresponden con el algoritmo PI-SRL con diferentes niveles de riesgo, los triángulos negros con la aproximación sensible al riesgo con los diferentes valores del parámetro ξ utilizados y, por último, el cuadrado azul pertenece a la aproximación basada en aprendizaje por refuerzo evolutivo. En la figura 6.9 se muestran dos asíntotas. La asíntota horizontal se establece en el refuerzo acumulado obtenido cuando se utiliza el valor más elevado de σ. Esta asíntota indica que valores más elevados del parámetro de riesgo, σ, incrementan el número de fallos, pero el refuerzo acumulado total obtenido no se mejora, incluso se podría empeorar debido al mayor número de fallos. La asíntota vertical que se establece en $fallos = 0$, indica que reduciendo el valor de σ no se consigue un menor número de fallos. La figura 6.9 muestra dos configuraciones de riesgo adicionales con respecto a la figura 6.8 (a): un nivel muy alto de riesgo ($\sigma = 9 \times 10^{-1}$), y un nivel muy bajo de riesgo ($\sigma = 0$). Cuando se establece un nivel muy bajo de riesgo, $\sigma = 0$, no se añade ruido aleatorio siguiendo una distribución Gaussiana a las acciones, lo que produce que el agente no colisione el coche, aunque el rendimiento no se mejora con respecto a la Política Segura Basada en Casos, π_B^θ, aprendida en el primer paso del algoritmo. El algoritmo PI-SRL con un nivel medio de riesgo ($\sigma = 9 \times 10^{-4}$) tampoco produce colisiones, aunque el rendimiento sólo se consigue mejorar ligeramente con respecto al comportamiento base. Cuando se utiliza un valor de riesgo alto ($\sigma = 9 \times 10^{-2}$) se obtiene el mayor refuerzo acumulado, 3053.37, con un número medio de fallos de 78.8. No obstante,

cuando el parámetro de riesgo se establece a un nivel más alto ($\sigma = 9 \times 10^{-1}$), el número de fallos se incrementa considerablemente y, por tanto, el refuerzo acumulado obtenido se reduce. El algoritmo PI-SRL con un nivel de riesgo alto ($\sigma = 9 \times 10^{-2}$) y la aproximación basada en aprendizaje por refuerzo evolutivo alcanzan un rendimiento similar al final del proceso de aprendizaje como se muestra en la figura 6.8 (a), aunque el algoritmo PI-SRL converge más rápidamente. Por este motivo, en la figura 6.9 el algoritmo PI-SRL con riesgo alto tiene un mayor refuerzo acumulado.

El criterio de comparación de Pareto puede ser utilizado para comparar las soluciones en la figura 6.9. Utilizando este principio, una solución y^* domina estrictamente (o "es preferida a") una solución y si cada parámetro de y^* no es estrictamente peor que el correspondiente parámetro de y y al menos un parámetro es estrictamente mejor. Esto se escribe como $y^* \succ y$ para indicar que y^* domina estrictamente a y. De acuerdo con el principio de Pareto, se puede asumir que todos los puntos correspondientes al algoritmo PI-SRL, excepto el correspondiente a un valor muy elevado de riesgo, se encuentran en la frontera de Pareto, puesto que estos puntos no son estrictamente dominados por ninguna otra aproximación (i.e., ninguna otra aproximación tiene al mismo tiempo un refuerzo acumulado mayor y un menor número de fallos que las soluciones de PI-SRL). Además, se puede decir que en este dominio, la solución de PI-SRL con nivel medio de riesgo domina estrictamente (o "es preferida a") las soluciones propuestas por la aproximación sensible al riesgo, i.e., PI-SRL $\sigma = 9 \times 10^{-3} \succ$ Aproximación sensible al riesgo, y la solución PI-SRL correspondiente a un nivel elevado de riesgo domina estrictamente a la solución obtenida mediante la aproximación basada en aprendizaje por refuerzo evolutivo, i.e., PI-SRL $\sigma = 9 \times 10^{-2} \succ$ AR Evolutivo.

En cualquier caso, decidir qué solución de las presentadas en la figura 6.9 es la mejor depende de lo que se esté tratando de conseguir. Si minimizar el número de fallos es el criterio de optimización más importante (independientemente de la mejora obtenida con respecto al comportamiento base, π_T), la mejor aproximación es el algoritmo PI-SRL con un nivel bajo de riesgo ($\sigma = 9 \times 10^{-4}$). Si por el contrario, maximizar el refuerzo acumulado total obtenido es el criterio de optimización más importante (independientemente del número de fallos), la mejor aproximación es PI-SRL con un nivel alto de riesgo ($\sigma = 9 \times 10^{2}$).

La figura 6.10 muestra la evolución de los casos en la base de casos B (espacio conocido) en diferentes episodios para un proceso de aprendizaje con un nivel de riesgo alto. Cada gráfica muestra el conjunto de estados conocidos, Ω (región verde), el conjunto de estados de error, Φ (región roja), el conjunto de estados desconocidos, Υ (región amarilla), y el conjunto de estados de no error, Γ_Ω (círculos naranjas). El algoritmo PI-SRL adapta el espacio conocido con el fin de encontrar mejores políticas seguras para completar la tarea. La figura 6.10 (a) muestra la situación inicial de la base de casos B (correspondiente a la Política Segura Basada en Casos, π_B^θ, aprendida en el primer paso del algoritmo). Esta política es robusta en el sentido de que nunca produce colisiones del coche con la pared o el obstáculo, pero su rendimiento está lejos del óptimo (selecciona el camino más largo para aparcar el coche dentro del garaje bordeando la parte superior del obstáculo). A medida que el proceso de aprendizaje avanza (figura 6.10 (b)), PI-SRL encuentra un camino más corto para aparcar el coche en el garaje (incrementándose el rendimiento), aunque pasando muy cerca del obstáculo (incrementándose las probabilidades de golpear el coche). En la figura 6.10 (c), PI-SRL encuentra un nuevo camino más corto para aparcar el coche bordeando la parte inferior del obstáculo. No obstante, en la base de casos B aún se mantienen los casos correspondientes al viejo camino que bordeaba la parte superior del obstáculo. Por este motivo, la figura 6.10 (c) muestra los dos caminos para aparcar el coche. Por último, en la

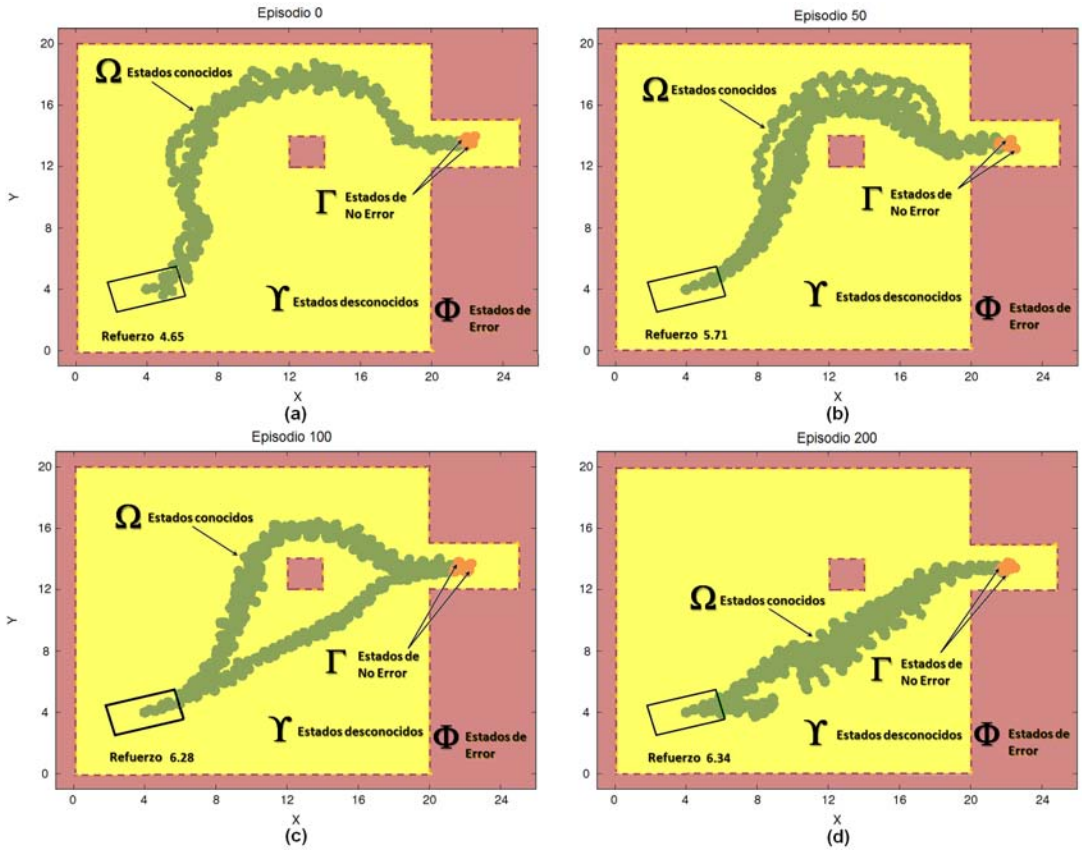


Figura 6.10: Evolución del espacio de estados conocido en los episodios $T = 0$ (a), $T = 50$ (b), $T = 100$ (c) and $T = 200$ (d) en un proceso de aprendizaje con un nivel alto de riesgo ($\sigma = 9 \times 10^{-2}$). Cada gráfico se corresponde con la situación del espacio de estados de acuerdo a la base de datos B en el episodio T.

figura 6.10 (d) los casos correspondientes al camino subóptimo bordeando la parte superior del obstáculo han sido eliminados de la base de datos B, y han sido reemplazados por los nuevos casos correspondientes al camino más corto y seguro bordeando la parte inferior del obstáculo. En otras palabras, el algoritmo PI-SRL adapta el espacio conocido explorando el espacio desconocido con el fin de encontrar nuevos y mejores comportamientos. Durante este proceso de ajuste del espacio conocido a las regiones del espacio donde se encuentra otros comportamientos seguros y mejores, el algoritmo "olvida" algunos de los estados conocidos e inútiles aprendidos anteriormente.

El siguiente experimento se ha llevado ha cabo para demostrar que si tratamos de abordar un dominio muy ruidoso, incluso no tomando riesgos (i.e., no añadiendo ruido adicional a los actuadores para explorar, $\sigma = 0$), el agente podría tener un rendimiento muy pobre produciendo constantemente fallos. Este experimento también será utilizado para explicar por qué sólo el ruido del propio dominio no puede ser utilizado para explorar eficientemente el espacio. Para ello, se ha añadido más ruido a los actuadores y se ha ejecutado de nuevo el segundo paso del algoritmo PI-SRL pero sin tomar riesgos (i.e., $\sigma = 0$). En este experimento, se ha añadido un ruido aleatorio siguiendo una distribución Gaussiana a los actuadores con desviación estándar de 0.3 en lugar de la desviación estándar de 0.1 utilizada en todos los experimentos anteriores como se describía en la sección 4.2.1. La figura 6.11 muestra dos

procesos de aprendizaje correspondientes a dos ejecuciones independientes del segundo paso del algoritmo PI-SRL, *Mejora del Comportamiento Base Aprendido*. El eje de x muestra los episodios, mientras que el eje y muestra el refuerzo total acumulado por episodio. Los fallos (i.e., las colisiones del coche con la pared o el obstáculo) están marcados como anteriormente mediante un triángulo azul. En los experimentos en la figura 6.8 (a), la Política Segura Basada en Casos π_B^θ, asociada a la base de casos B, con un nivel bajo de riesgo ($\sigma = 9 \times 10^{-4}$) nunca produce fallos. Por el contrario, en los experimentos que se muestran en la figura 6.11, la misma política asociada a la base de casos, continuamente colisiona el coche con la pared a pesar de que el parámetro de riesgo se mantiene a 0 ($\sigma = 0$). Además, también se puede apreciar un ligero incremento en el rendimiento del algoritmo.

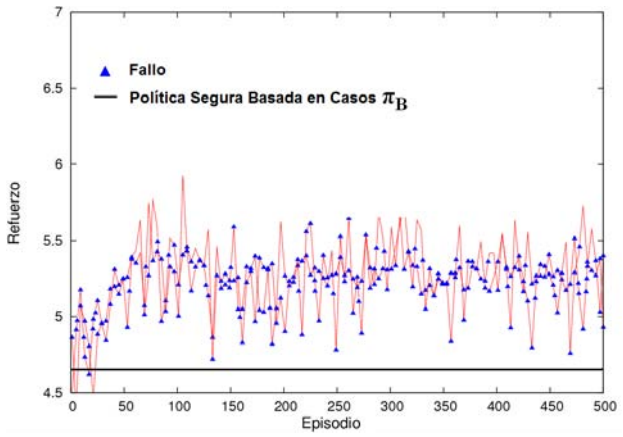


Figura 6.11: Ejecución del segundo paso del algoritmo PI-SRL incrementado el ruido de los actuadores y con $\sigma = 0$

El incremento del ruido de los actuadores en el segundo paso del algoritmo con respecto al primer paso (la Política Segura Basada en Casos, π_B^θ, se ha aprendido en el primer paso del algoritmo utilizando un ruido aleatorio siguiendo una distribución Gaussiana con una desviación estándar de 0.1, y en este experimento el segundo paso se ha ejecutando utilizando ruido con una desviación estándar de 0.3), lleva al agente más allá del espacio de estados conocidos en la base de casos B aprendida en el primer paso, y es capaz de encontrar nuevas trayectorias para aparcar el coche el garaje. En esta nueva situación, el proceso de exploración se lleva a cabo de la siguiente forma. Si se alcanza un nuevo estado, el agente ejecuta la acción a recuperada de la base de casos B sin la adición de ruido aleatorio Gaussiano puesto que el parámetro de riesgo es 0 ($\sigma = 0$) (línea 11 del algoritmo de la tabla 6.3). Si por el contrario el agente llega a un estado desconocido, se ejecuta la acción a aconsejada por el comportamiento base, π_T (línea 15 del algoritmo de la tabla 6.3). Utilizando este proceso de exploración, si se encuentra una trayectoria mejor para aparcar el coche en el garaje, los casos resultantes del episodio correspondientes a estados desconocidos se añaden a la base de casos (línea 37 del algoritmo de la tabla 6.3), incrementándose ligeramente el rendimiento en la figura 6.11. Conviene destacar que los reemplazos de casos (línea 34 del algoritmo de la tabla 6.3), en caso de producirse, no producen cambios en las acciones de la base de casos B, puesto que las acciones en B son reemplazadas por la misma acción previamente recuperada de B más cierta cantidad de ruido aleatorio Gaussiano con desviación estándar σ (línea 11 del algoritmo de la tabla 6.3). Pero puesto que no se añade ruido a las acciones con el fin de explorar, $\sigma = 0$, las acciones que se recuperan de la base

de casos no se cambian. En cualquier caso, este proceso de exploración con $\sigma = 0$ (i.e., sin tomar riesgos), no permite aprender comportamientos óptimos porque:

1. Las acciones ejecutadas en estados desconocidos, y añadidos a la base de casos B, son seleccionadas mediante el comportamiento base, π_T, que tiene un rendimiento subóptimo (definición del comportamiento base).

2. Las acciones en la base de casos B no se reemplazan por acciones mejores. El ruido aleatorio Gaussiano con desviación estándar σ se emplea para explorar acciones diferentes y mejores que las suministradas por la base de casos, B, y por el comportamiento base, π_T, pero, en este caso, el parámetro de riesgo se estable a $\sigma = 0$, y no es posible encontrar nuevas y mejores acciones.

Experimentos adicionales han demostrado que PI-SRL se comporta aún peor cuando se incrementa el nivel del ruido que se añade a los actuadores (el coche colisiona contra la pared o el obstáculo en prácticamente todos los episodios). Este experimento demuestra que un valor de $\sigma = 0$, implica ejecutar siempre las mismas acciones suministradas por la base de casos, B, y el comportamiento base, π_T, e implica, por tanto, no encontrar nuevas y mejores acciones que ejecutar en situaciones conocidas. En PI-SRL, los reemplazos llevados cabo se realizan hacia las acciones más prometedoras, en nuestro caso, hacia las acciones que prometen un mayor refuerzo acumulado. Esta es la razón por la cuál es necesario ejecutar un proceso de exploración que permita explorar el espacio en busca de un comportamiento óptimo: sin la exploración no se descubren nuevas y mejores acciones y el rendimiento del algoritmo PI-SRL está fuertemente limitado por el rendimiento de la Política Segura Basada en Casos, π_B^θ aprendida en el primer paso, y el comportamiento base, π_T, cuyo rendimiento es subóptimo.

6.5.2. *Cart-Pole*

El dominio del *Cart-Pole* se explica en detalle en la sección 4.2.2. De forma similar al dominio anterior, se ejecuta el primer paso del algoritmo PI-SRL, *Modelado del Comportamiento Base* (sección 6.3.1), para aprender la Política Segura Basada en Casos, π_B^θ, a partir de las demostraciones del comportamiento base, π_T. En este caso también se considera que se dispone de un comportamiento base, π_T, cuyas demostraciones de la tarea son seguras aunque subóptimas, y su refuerzo acumulado total por episodio es de 9292. Los parámetros θ y η se calculan siguiendo los procesos de cómputo descritos en la sección 6.4 a partir de los cuáles se establecen sus valores de 0.02 y 12572 respectivamente.

Para cada proceso de aprendizaje mostrado en la figura 6.12 (a), se muestran el número pasos por episodio ejecutados por el comportamiento base, π_T (líneas sólidas rojas), y el número de pasos por episodio ejecutando la base de casos, B (líneas discontinuas verdes). Al principio del proceso de aprendizaje, la base de casos B está vacía y todos los pasos se ejecutan utilizando el comportamiento base, π_T. A medida que el proceso de aprendizaje avanza, la base de casos B se va llenando y la Política Segura Basada en Casos, π_B^θ, se va aprendiendo. Al final del proceso de aprendizaje (alrededor de los episodios 40-50), casi todos los pasos son ejecutados utilizando los casos en la base de casos B, y apenas se solicitan acciones al comportamiento base, π_T. Nuevamente el paso de Modelado del Comportamiento Base se lleva a cabo sin ningún fallo (i.e., sin desbalancear en ningún momento el péndulo o sacando el carro fuera de la pista). La figura 6.12 (b) muestra tres ejecuciones diferentes utilizando: i) la Política Segura Basada en Casos, π_B^θ, aprendida

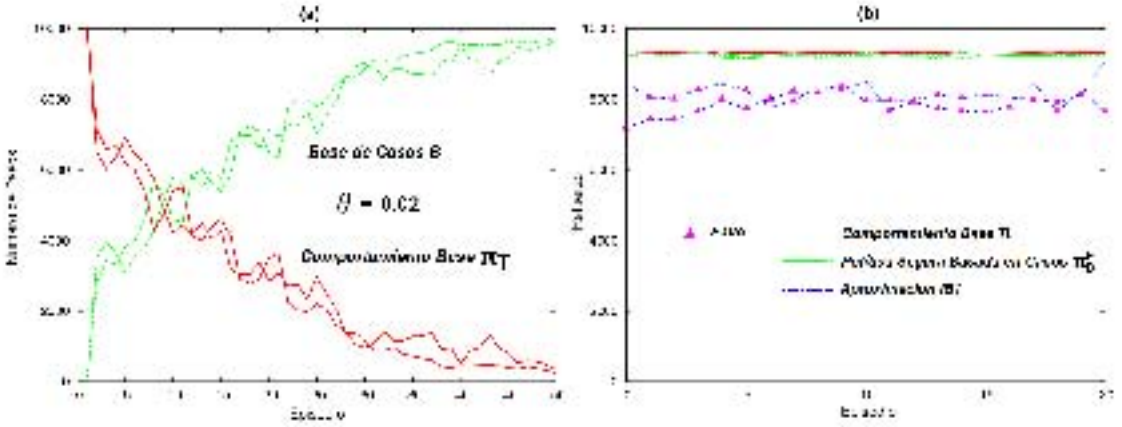


Figura 6.12: (a) Número de pasos por episodios ejecutados mediante la base de casos B y el comportamiento base π_T. (b) Refuerzo total acumulado por episodio obtenido por π_T, la Política Segura Basada en Casos π_B^θ y la aproximación *IBL*.

anteriormente, ii) el comportamiento base, π_T, y iii) una aproximación basada en IBL [Aha and Kibler, 1991]. El refuerzo acumulado total por episodio obtenido por la Política Segura Basada en Casos aprendida, π_B^θ, es de 9230 (líneas discontinuas verdes). También se aprecia que la Política Segura Basada en Casos, π_B^θ casi imita perfectamente el rendimiento del comportamiento base, π_T. Por otro lado, la aproximación IB1 (líneas discontinuas azules) en muchos episodios produce fallos, i.e., desbalanceos del péndulo o que el carro salga fuera de la pista establecida, obteniendo un refuerzo total acumulado por episodio de 8055.

La figura 6.13 (a) muestra los resultados del segundo paso del algoritmo PI-SRL, *Mejora del Comportamiento Base Aprendido*, para diferentes configuraciones del parámetro de riesgo.

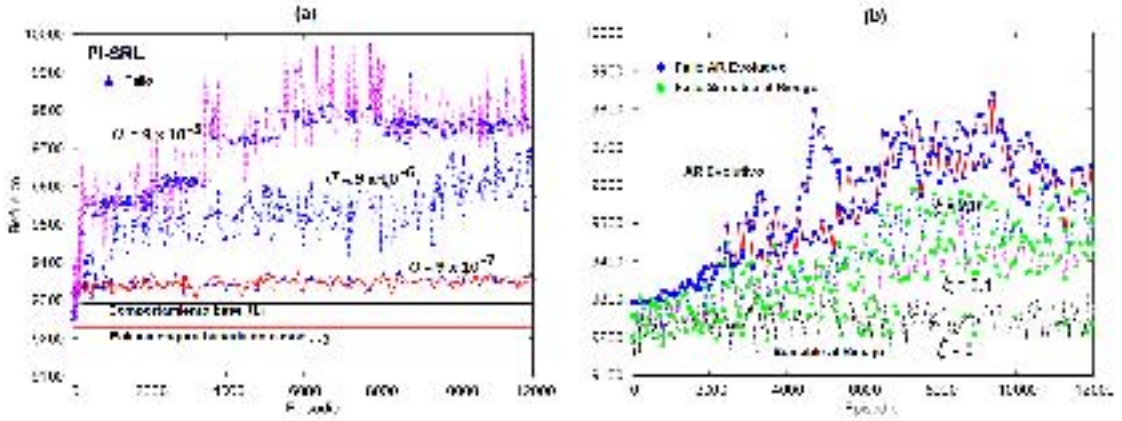


Figura 6.13: (a) Refuerzo acumulado total por episodio obtenido por PI-SRL mediante diferentes configuraciones de riesgo (σ). (b) Refuerzo acumulado total por episodio obtenido mediante la aproximación basada en aprendizaje por refuerzo evolutivo y la aproximación sensible al riesgo. En todos los casos, se marca cuando el episodio finaliza con fallo.

Para cada configuración del parámetro de riesgo, la figura 6.13 (a) muestra dos curvas de aprendizaje. Además, cuando un episodio finaliza en fallo se indica mediante los triángulos

azules que aparecen en la figura. Los experimentos demuestran que incrementar el nivel de ruido incrementa la probabilidad de fallos, pero la política obtenida mejora en términos de refuerzo acumulado total por episodio. No obstante, cuando se ejecutan experimentos con valores superiores a $\sigma = 9 \times 10^{-5}$, se incrementa el número de fallos obtenidos pero no se consigue mejorar el rendimiento. La figura 6.13 (b) muestra los resultados obtenidos en el dominio por la aproximación basada en aprendizaje por refuerzo evolutivo y la aproximación sensible al riesgo utilizando diferentes valores del parámetro ξ. El algoritmo de aprendizaje por refuerzo evolutivo es claramente la aproximación con el mayor número de fallos. En la aproximación sensible al riesgo con $\xi = 2$ (maximización de la función de valor), el algoritmo selecciona acciones que conducen a un mayor refuerzo acumulado, aunque también a un mayor riesgo. Por el contrario, en el caso de $\xi = 0$ (minimización del riesgo), cuando el agente aproxima correctamente la función de riesgo (alrededor del episodio 6000), selecciona acciones con un riesgo menor y el número de fallos se reduce, aunque el rendimiento del algoritmo es bastante pobre. Con un valor de $\xi = 0{,}1$ se genera una política intermedia. Por lo tanto, PI-SRL con un nivel de riesgo alto obtiene políticas mejores y con menor número fallos que la aproximación basada en aprendizaje por refuerzo evolutivo y la aproximación sensible al riesgo.

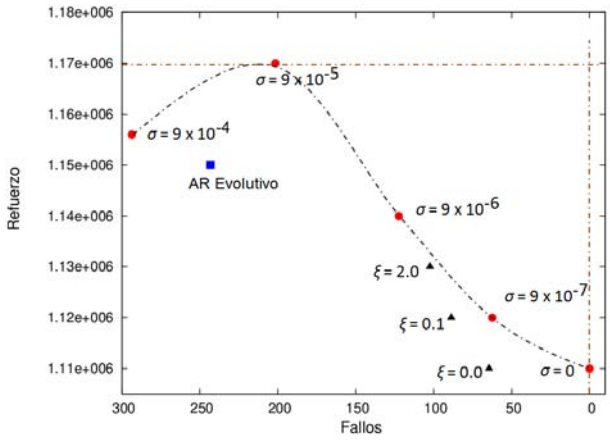


Figura 6.14: Número medio de fallos (desbalanceos del péndulo y el carro fuera de la pista) y refuerzo acumulado total durante 12000 episodios obtenidos por cada aproximación.

La figura 6.14 refuerza las anteriores conclusiones. Esta figura muestra el número medio de fallos y el refuerzo acumulado total obtenido durante 12000 episodios. Los círculos rojos pertenecen al algoritmo PI-SRL, los triángulos negros a la aproximación sensible al riesgo, y el cuadrado azul se corresponde con la aproximación basada en aprendizaje por refuerzo evolutivo. La figura 6.14 también muestra el rendimiento del algoritmo PI-SRL cuando se emplean dos niveles de riesgo adicionales con respecto a la figura 6.13 (a): un nivel de riesgo muy alto ($\sigma = 9 \times 10^{-4}$), y un valor de riesgo muy bajo ($\sigma = 0$). El refuerzo acumulado total y el número de fallos se incrementan hasta el nivel de riesgo correspondiente a $\sigma = 9 \times 10^{-5}$. Este nivel de riesgo representa un punto de inflexión en el cuál el número de fallos se sigue incrementando, pero el refuerzo acumulado no se mejora. De hecho, el nivel de riesgo muy alto ($\sigma = 9 \times 10^{-4}$) produce una reducción brusca en el refuerzo acumulado total obtenido. De nuevo, se puede utilizar el principio de Pareto para comparar las diferentes soluciones ofrecidas por los algoritmos en la figura 6.14. En este dominio, la solución de PI-SRL con un nivel medio de riesgo domina estrictamente a la aproximación sensible al riesgo con $\xi = 0{,}0$ y

$\xi = 0{,}1$ (i.e., PI-SRL $\sigma = 9 \times 10^{-7} \succ$ Aproximación sensible al riesgo con $\xi = 0{,}0$ y $\xi = 0{,}1$). La solución PI-SRL con un nivel alto de riesgo domina estrictamente a la aproximación basada en aprendizaje por refuerzo evolutivo (i.e., PI-SRL $\sigma = 9 \times 10^{-5} \succ$ AR Evolutivo).

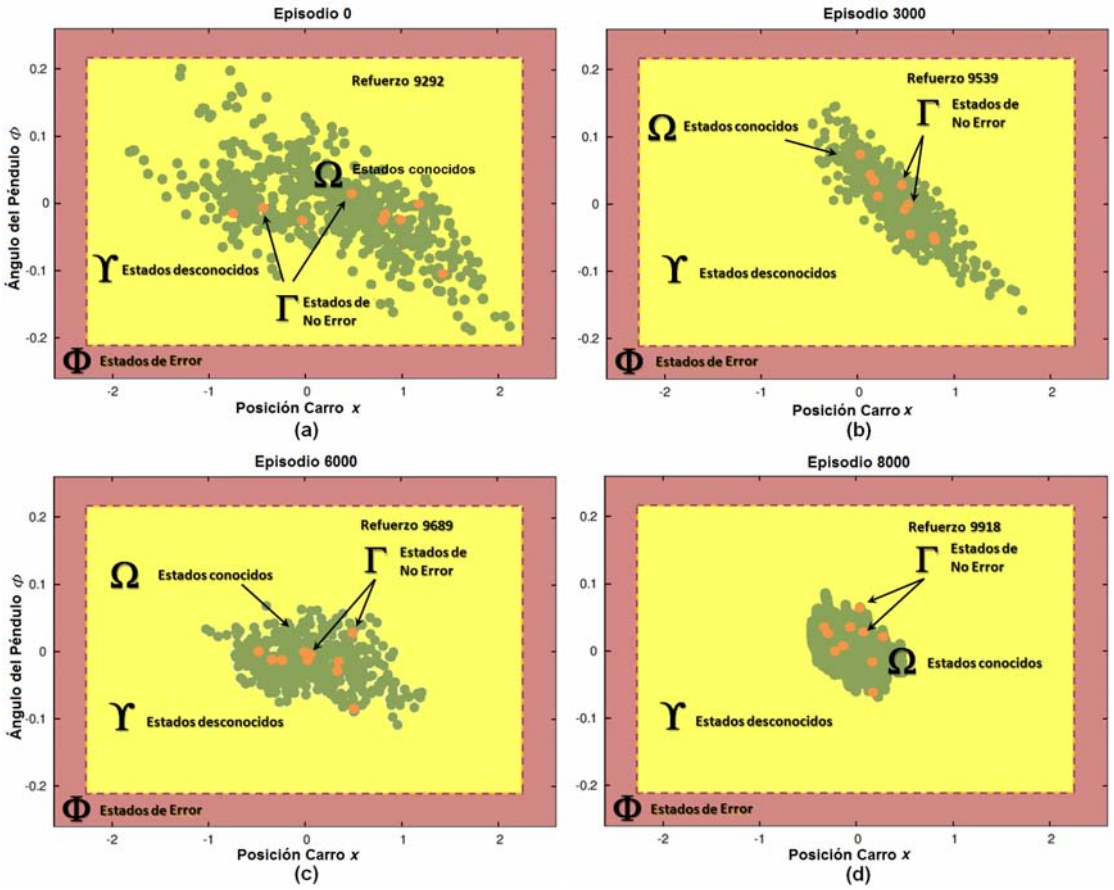


Figura 6.15: Evolución del espacio conocido en los episodios $T = 0$ (a), $T = 3000$ (b), $T = 6000$ (c) and $T = 8000$ (d) en un proceso de aprendizaje con un nivel de riesgo alto ($\sigma = 9 \times 10^{-5}$). Cada gráfica se corresponde con la situación del espacio de estados de acuerdo a la base de casos B en el episodio T.

Por último, la figura 6.15 muestra la evolución del espacio conocido derivado de la base de casos B en diferentes episodios durante un proceso de aprendizaje con un nivel de riesgo alto. En cada gráfica se muestra el conjunto de estados de error, Φ (región roja), el conjunto de estados desconocidos, Υ (región amarilla), el conjunto de estados conocidos, Ω (región verde), y el conjunto de estados de no error Γ_Ω (círculos naranjas). El espacio conocido, Ω, en cada gráfica se ha calculado tomando los casos de la base de casos B en los episodios $T = 0, 3000, 6000$ y 8000. Para cada gráfico, los estados de no-error, Γ_Ω, se han calculado a partir de 10 ejecuciones independientes de la base de casos B correspondiente al episodio T (los círculos naranjas representan los estados terminales para cada una de estas ejecuciones independientes). La primera gráfica (figura 6.15 (a)) presenta la situación de la base de casos B resultante del paso de *Modelado del Comportamiento Base*. La evolución que se aprecia en la figura 6.15 muestra dos características diferentes del algoritmo en este dominio. En primer lugar, el algoritmo PI-SRL adapta progresivamente el espacio de estados conocidos a regiones donde se encuentran mejores comportamientos. En este caso, el espacio

de estados conocido tiende a comprimirse en el centro de coordenadas. Este hecho se debe a que el refuerzo es mayor cuanto más vertical está el péndulo encima del carro y cuanto más centrado está el carro en la pista, es decir, si el ángulo ϕ del péndulo y la posición del carro x son 0. En segundo lugar, la probabilidad de fallo en el dominio del *Cart-Pole* es mayor en los episodios iniciales del proceso de aprendizaje. Al principio del proceso de aprendizaje, (figura 6.15 (a), $T = 0$), algunas zonas del espacio conocido están cerca del espacio de error. En esta situación, pequeñas modificaciones en las acciones producen que se visiten estados pertenecientes al espacio de error (desbalanceos del péndulo, y movimientos del carro fuera de la pista). A medida que el proceso de aprendizaje avanza (figuras 6.15 (b), (c) y (d)), el espacio conocido se comprime en el origen de coordenadas, alejándose del espacio de error. En este momento, la probabilidad de visitar estados de error se reduce. Para contrastar este hecho, se comprueba que en los experimentos de la figura 6.13 (a), los procesos de aprendizaje con un nivel alto de riesgo producen el 52 % de los fallos (126) en los primeros 4000 episodios, y el restante 48 % (117) en los últimos 8000 episodios.

6.5.3. *Helicopter*

La descripción del dominio del *Helicopter Control Task* se ha presentado en la sección 4.2.3. La ejecución del algoritmo PI-SRL requiere la existencia de un comportamiento base, π_T, previamente definido en el dominio. En este dominio, el comportamiento base, π_T, considerado tiene un refuerzo total por episodio de -78035.93. En primer lugar, se ejecuta el primer paso de M*odelado del Comportamiento Base*, con el fin de aprender la Política Segura Basada en Casos, π_B^θ, capaz de imitar al comportamiento base, π_T.

La figura 6.16 (a) muestra dos ejecuciones del paso de *Modelado del Comportamiento Base*.

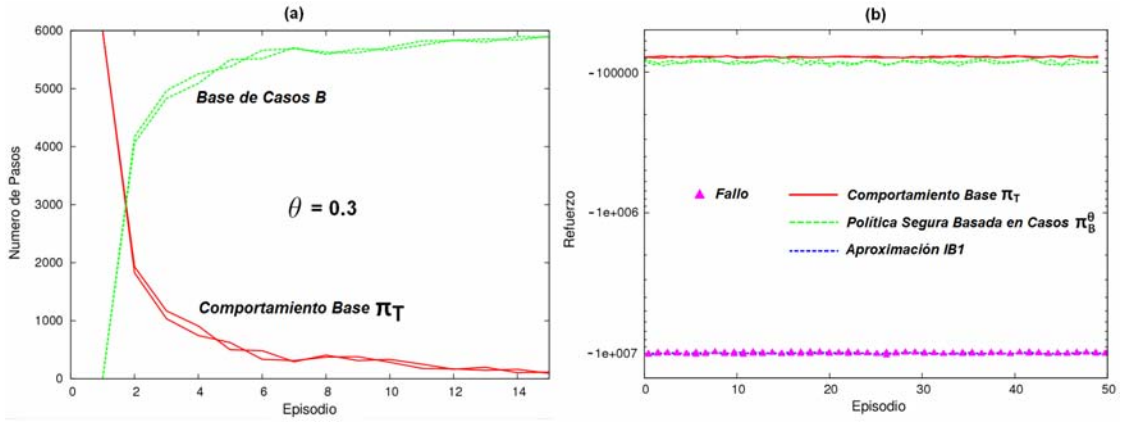


Figura 6.16: (a) Número de pasos por episodios ejecutados mediante la base de casos B y el comportamiento base π_T. (b) Refuerzo total acumulado por episodio obtenido por π_T, la Política Segura Basada en Casos π_B^θ y la aproximación *IBL*.

Como en los anteriores dominios, esta gráfica muestra cómo a medida que el proceso de aprendizaje avanza, se reduce el número de pasos que son ejecutados con acciones obtenidas por el comportamiento base, π_T, y el número de pasos ejecutados mediante acciones extraídas de la base de casos B se incrementa. Al final del proceso de aprendizaje, la base de casos B almacena la Política Segura Basada en Casos, π_B^θ. La figura 6.16 (b) compara el rendimiento (en términos de refuerzo total acumulado por episodio) del comportamien-

to base, π_T, la Política Segura Basada en Casos aprendida, π_B^θ, y la aproximación *IB1*. El refuerzo total por episodio obtenido por el comportamiento base, π_T, es de -78035.93, mientras el obtenido por la política, π_B^θ, es de -85130.11. Aunque la política π_B^θ, no imita perfectamente el comportamiento de la política, π_T, es una política segura que no produce colisiones del helicóptero. En este caso, el proceso de entrenamiento del algoritmo IB1 se ha llevado a cabo almacenando todos los casos producidos durante 15 episodios mediante el comportamiento base, π_T. La figura 6.16 (b) demuestra que procediendo de esta forma el helicóptero colisiona constantemente y el rendimiento está muy lejos del alcanzado por la Política Segura Basada en Casos aprendida, π_B^θ. Posteriormente, la política π_B^θ aprendida se mejora mediante la exploración segura de los espacios de estados y acciones llevada a cabo en el segundo paso del algoritmo PI-SRL.

La figura 6.17 (a) muestra los resultados para diferentes configuraciones de riesgo.

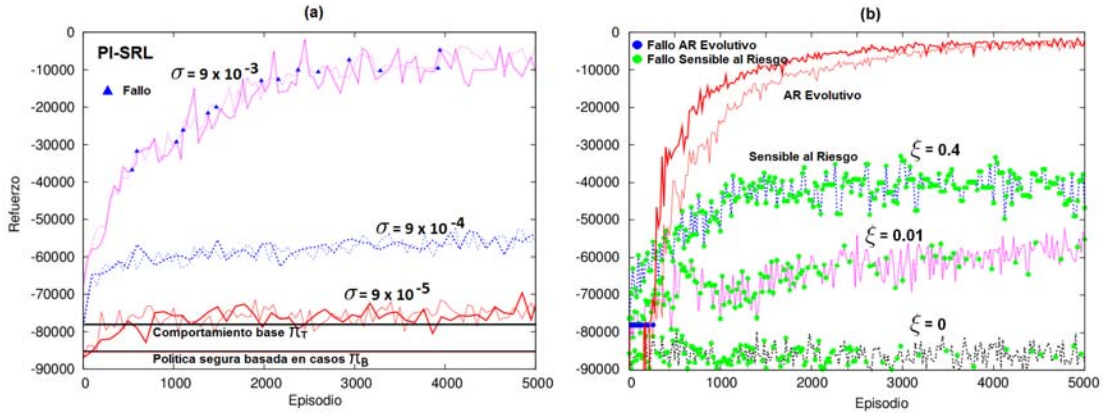


Figura 6.17: (a) Refuerzo acumulado total por episodio obtenido por PI-SRL mediante diferentes configuraciones de riesgo (σ). (b) Refuerzo acumulado total por episodio obtenido mediante la aproximación basada en aprendizaje por refuerzo evolutivo y la aproximación sensible al riesgo. En todos los casos, se marca cuando el episodio finaliza con fallo.

En este caso, niveles bajos y medios de riesgo no producen colisiones del helicóptero aunque no se consiguen mejoras significativas en el rendimiento. El nivel de riesgo alto establecido produce una política de comportamiento cercana al óptimo con un número reducido de colisiones. Una experimentación extensa llevada a cabo utilizando niveles de riesgo superiores a $\sigma = 9 \times 10^{-3}$, muestra que con más riesgo se producen más colisiones, aunque el refuerzo acumulado total no se mejora. La figura 6.17 (b) muestra los resultados obtenidos por el ganador de la competición de aprendizaje por refuerzo en la última edición celebrada el año 2009 [Martín and de Lope Asiaín, 2009], utilizando la aproximación basada en aprendizaje por refuerzo evolutivo. La figura 6.17 (b) también muestra los resultados de la aproximación sensible al riesgo cuando se utilizan diferentes valores del parámetro ξ. La comparación de resultados muestra que el algoritmo PI-SRL con un nivel alto de riesgo es capaz de obtener un rendimiento similar que el obtenido por el algoritmo de aprendizaje por refuerzo evolutivo, aunque el número colisiones producido por PI-SRL es mucho menor. Además, las colisiones del aprendizaje por refuerzo evolutivo se producen todas en los episodios iniciales del proceso de aprendizaje, mientras que las colisiones del algoritmo PI-SRL ocurren de una forma más uniforme. En el caso del algoritmo sensible al riesgo, para valores $\xi = 0$ y $\xi = 0{,}01$ la función de riesgo se logra aproximar alrededor del episodio 3000. En este momento, el agente selecciona acciones con un nivel de riesgo menor y el número de

colisiones comienza a reducirse considerablemente. Con un valor $\xi = 0{,}4$ el agente selecciona acciones que resultan en valores de la función de valor superiores, sin tener prácticamente el riesgo. En este caso, el rendimiento mejora aunque el número de colisiones se incrementa. En cualquier caso, sea cual sea el valor del parámetro ξ, el número de colisiones es mayor y el rendimiento inferior que el obtenido por el algoritmo PI-SRL.

La figura 6.18 proporciona una representación gráfica del rendimiento de cada aproximación.

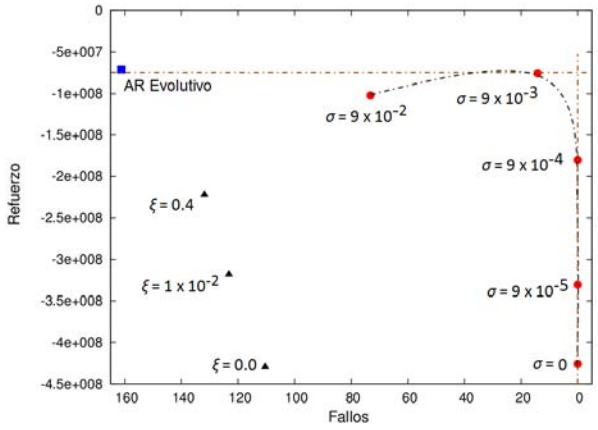


Figura 6.18: Número medio de fallos (colisiones del helicóptero) y refuerzo acumulado total durante 5000 episodios obtenidos por cada aproximación.

Esta información se puede utilizar para completar las conclusiones sobre el rendimiento del algoritmo PI-SRL frente al resto de algoritmos. En particular, la figura 6.18 muestra el número medio de fallos y el refuerzo acumulado total obtenido durante 5000 episodios. Estos datos se han calculado a partir de 10 ejecuciones independientes para cada aproximación y para cada configuración. Como en los dominios anteriores donde se proporcionaba una gráfica similar, los círculos rojos pertenecen al algoritmo PI-SRL, los triángulos negros a la aproximación sensible al riesgo, y el cuadrado azul corresponde a la aproximación basada en aprendizaje por refuerzo evolutivo. Además, la figura 6.18 muestra el rendimiento de dos configuraciones adicionales del parámetro de riesgo σ para el algoritmo PI-SRL: un nivel de riesgo muy alto ($\sigma = 9 \times 10^{-2}$), y un nivel de riesgo muy bajo ($\sigma = 0$). La figura demuestra que la aproximación evolutiva obtiene el mayor refuerzo acumulado total, $-7{,}13 \times 10^7$, aunque seguida muy de cerca por el algoritmo PI-SRL con un nivel de riesgo alto con $-7{,}57 \times 10^7$. La otra aproximación está muy alejada de estos resultados. En cuanto al número de fallos (i.e., colisiones del helicóptero), PI-SRL con nivel de riesgo muy bajo ($\sigma = 0$), con riesgo bajo ($\sigma = 9 \times 10^{-5}$) y con riesgo medio ($\sigma = 9 \times 10^{-4}$) no produce colisiones, aunque PI-SRL con un nivel de riesgo medio es la solución preferida de entre estas tres, puesto que es la que mayor refuerzo acumulado alcanza, $-18{,}01 \times 10^7$. Utilizando nuevamente el criterio de comparación de Pareto, se puede establecer que la solución propuesta por PI-SRL con un nivel elevado de riesgo domina estrictamente a las soluciones propuestas por el algoritmo sensible al riesgo para las configuraciones utilizadas, PI-SRL $\sigma = 9 \times 10^{-3} \succ$ Aproximación Sensible al Riesgo. Además, se puede afirmar que las soluciones propuestas por nuestro algoritmo no son estrictamente dominadas por ninguna otra solución.

Como en los anteriores dominios, la figura 6.19 muestra la evolución del espacio conocido de acuerdo a la situación de la base de casos B en diferentes episodios en un proceso de

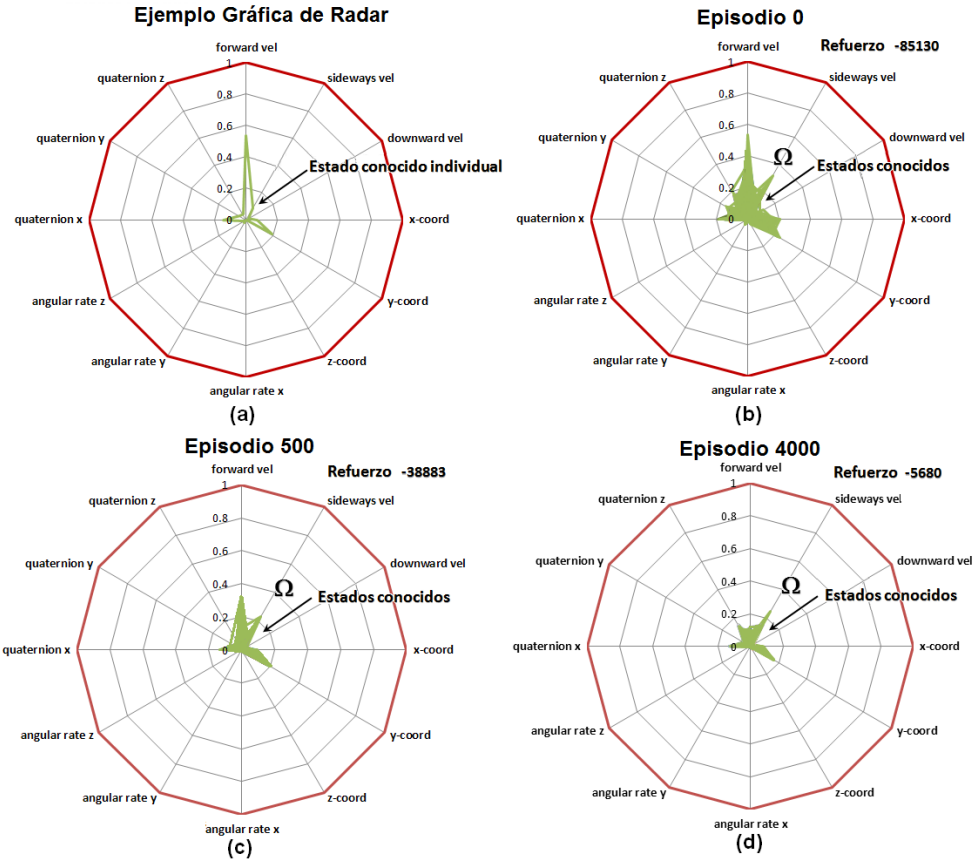


Figura 6.19: (a) Ejemplo de representación de un sólo estado conocido en un gráfico de radar. (b), (c), y (d) Estados conocidos en los episodios $T = 0$, $T = 500$, and $T = 4000$ respectivamente en un proceso de aprendizaje con un nivel alto de riesgo ($\sigma = 9 \times 10^{-3}$). Cada gráfica se corresponde con la situación del espacio de estados de acuerdo a la base de casos B en el episodio T.

aprendizaje correspondiente a un nivel elevado de riesgo. En este caso, debido al gran número de variables de estado, se utilizan gráficas de radar. Una gráfica de radar es un método para representar datos multi-variables utilizando dos dimensiones. En nuestro caso, cada eje de las gráficas presentadas en la figura 6.19 muestra una variable de estado. Por simplicidad en la representación, las gráficas en la figura 6.19 se han generado normalizando los valores absolutos de los variables de estado entre 0 y 1. La figura 6.19 (a) muestra un ejemplo de representación de un único estado conocido utilizando esta forma de representación. El valor de cada eje se corresponde con el valor de una variable individual del estado correspondiente. Se dibuja una línea que conecta los valores de cada variable de estado correspondientes a cada eje, de forma que la línea en la figura 6.19 (a) representa a un estado individual. La figura 6.19 (b), (c) y (d) muestra el espacio conocido de acuerdo a la base de casos B en los episodios 0, 500, y 4000 respectivamente. En estas gráficas, no se representa un único estado, sino todos los estados contenidos en la base de casos B en el episodio correspondiente. Por este motivo, en cada gráfica en la figura 6.19 (b), (c) y (d), se marca el conjunto de estados conocidos, Ω (región verde). Un estado se considera un estado de error si el valor de una

variable individual para ese estado es mayor que 1 (límite marcado mediante una línea roja en las gráficas). Estos límites se han calculado teniendo en cuenta que el helicóptero colisiona si la velocidad a lo largo de cualquiera de los ejes supera los 5 m/s, la posición del helicóptero está más alejada de $20m$ de la posición inicial, si el *angular rate* alrededor de cualquiera de los ejes principales excede 2×2 rad/s o la orientación del helicóptero es mayor a 30 *grados* con respecto a la posición inicial. Como en el dominio anterior, la figura 6.19 muestra dos características diferentes. En primer lugar, a medida que el proceso de aprendizaje avanza, el espacio conocido que se deriva de la base de casos B se ajusta a regiones del espacio de estados con mejores políticas. En este dominio, el agente trata de mantener el helicóptero lo más cerca posible de una posición objetivo que en este caso coincide con el origen del coordenadas. Por lo tanto, el refuerzo es mayor cuanto más cerca vuela el helicóptero del origen de coordenadas. De esta forma, el espacio conocido comienza expandido (figura 6.19 (b)), y progresivamente se concentra en el origen de coordenadas (figura 6.19 (c) y (d)). En segundo lugar, la probabilidad de colisionar el helicóptero es baja desde el comienzo del proceso de aprendizaje puesto que ya desde el principio el espacio conocido aparece bastante concentrado en el origen de coordenadas alejado del espacio de error (figura 6.19 (b)). En otras palabras, desde el principio todas las variables del espacio conocido están bastante alejadas de los límites del espacio de error, reduciéndose desde el principio las probabilidades de visitar estados de error.

En los experimentos anteriores en este dominio, el segundo paso del algoritmo PI-SRL se ha ejecutado utilizando una base de casos B, libre de fallos y, desde el principio, bastante alejada del espacio de estados de error, obtenida mediante la ejecución del primer paso de *Modelado del Comportamiento Base*. Los siguientes experimentos muestran cuál es el rendimiento del segundo paso del algoritmo PI-SRL cuando se emplean políticas iniciales diferentes a la utilizada anteriormente.

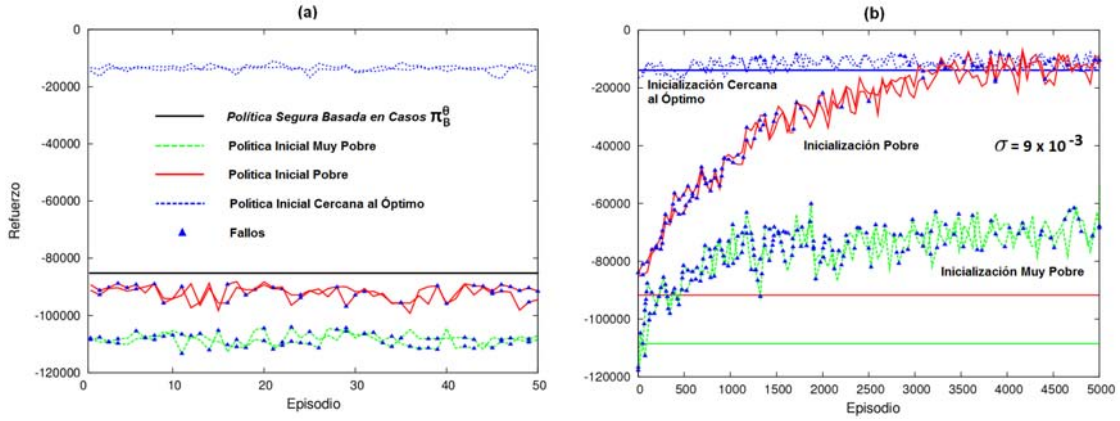


Figura 6.20: (a) El refuerzo acumulado por episodio obtenido por diferentes políticas el dominio del *Helicopter*. (b) El rendimiento de diferentes ejecuciones del segundo paso del algoritmo PI-SRL cada una empezando de una base de casos B conteniendo una política diferente: una política muy pobre, una política pobre, y una política cercana al óptimo.

La figura 6.20 (a) muestra el rendimiento de cada una de estas políticas utilizadas. La línea continua negra muestra el rendimiento de la Política Segura Basada en Casos, π_B^θ, utilizada en los experimentos anteriores antes de ejecutar el segundo paso del algoritmo PI-SRL. El refuerzo acumulado de esta política por episodio es de -85130.11. El resto de líneas corresponden al rendimiento de tres inicializaciones diferentes de la base de casos

B utilizadas en estos nuevos experimentos, antes de ejecutar el segundo paso de PI-SRL. Las líneas discontinuas verdes pertenecen a una política inicial muy pobre. Esta política produce colisiones del helicóptero constantemente y el refuerzo acumulado por episodio que obtiene es de -108548.03. Las líneas continuas rojas se corresponden con una política inicial pobre. Cuando se utiliza esta política el helicóptero se cae ocasionalmente y el refuerzo acumulado obtenido por episodio es de -91723.89. Por último, las líneas discontinuas azules pertenecen a una política cercana a la óptima donde además el helicóptero no colisiona nunca. El refuerzo acumulado total por episodio obtenido por esta última política es de -13940.1.

La figura 6.20 (b) muestra el rendimiento del segundo paso del algoritmo PI-SRL, *Mejora del Comportamiento Base Aprendido*, cuando se parte de una base de casos correspondiente a la política muy pobre, la política pobre y la política cercana a la óptima presentadas en la figura 6.20 (a). En la figura 6.20 (b), las líneas discontinuas azules se corresponden a la utilización de la base de casos B que contiene la política cercana a la óptima; las líneas continuas rojas pertenecen a la utilización de la base de casos B que contiene la política pobre; y las líneas discontinuas verdes pertenecen a la utilización de la base de casos B que contiene la política muy pobre. Todos los experimentos en la figura 6.20 (b) se han llevado a cabo utilizando un nivel alto de riesgo, i.e., $\sigma = 9 \times 10^{-3}$. Las líneas discontinuas azules muestran que utilizando una política cercana al óptimo como política inicial, y un nivel de riesgo alto, no se degrada el rendimiento de la base de casos B. De hecho, se puede observar como se consigue mejorar ligeramente. Como se comentó en la descripción del segundo paso del algoritmo 6.3.2, se previene la degradación de la base de casos B, evitando las actualizaciones con casos provenientes de malos episodios o episodios con fallos. En otras palabras, las actualizaciones en B se harán a partir de casos provenientes de episodios con un refuerzo acumulado total similar al mejor episodio encontrado hasta el momento, utilizando el umbral Θ (el valor de Θ se ha establecido al 5 % del episodio con mayor refuerzo acumulado obtenido hasta el momento). Por ejemplo, si el refuerzo acumulado total del mejor episodio es de -13940.1, sólo los episodios con un refuerzo acumulado superior a -14637 serán utilizados para realizar actualizaciones en la base de casos, descartando de esta forma los malos episodios o los episodios con fallos. De esta forma, se suministra al proceso de actualización de la base de casos con buenas secuencias de experiencia puesto que se ha probado que las buenas secuencias pueden ayudar al agente a converger a una política estable y útil, mientras que las malas secuencias podrían producir que el agente convergiera a una política inestable y pobre [Wyatt, 1997]. Las líneas continuas rojas en la figura 6.20 (b) muestra que utilizando una política pobre con fallos como política inicial, se produce un número de fallos más elevado que cuando se utiliza un política inicial sin fallos. No obstante, el segundo paso del algoritmo PI-SRL es capaz de aprender una política cercana al óptimo a pesar de esta pobre inicialización, de igual forma que cuando se utiliza una política inicial libre de fallos para inicializar la base de casos B, como se muestra en los experimentos correspondientes a un nivel alto de riesgo, $\sigma = 9 \times 10^{-3}$, en la figura 6.17 (a). Por último, las líneas discontinuas verdes en la figura 6.20 (b) demuestran que utilizando una política muy pobre como política inicial con muchos fallos, el segundo paso del algoritmo PI-SRL reduce su rendimiento y produce un número de fallos muy elevado, aunque es capaz de obtener un comportamiento mejor. En este caso, el algoritmo cae en un mínimo local, probablemente sesgado por la inicialización tan pobre. En ambos casos, tanto cuando se utiliza una pobre y una muy pobre inicialización, el número de fallos es mayor al principio del proceso de aprendizaje y se va reduciendo a medida que avanza. Al principio del proceso de aprendizaje, las políticas iniciales pobre y muy pobre están muy cerca del espacio de error,

al contrario que la política inicial mostrada en la figura 6.19 que ya desde el inicio aparecía concentrada en el origen de coordenadas, bastante alejada del espacio de error. A medida que el proceso de aprendizaje avanza, estas políticas se comprimen alejándose cada vez más del espacio de error, por lo que se reduce el número de fallos que se producen.

6.5.4. SIMBA

En esta sección se presentan los resultados obtenidos en el dominio del simulador empresarial SIMBA (Sección 4.2.5). En este dominio, un fallo viene dado por la bancarrota de una compañía, situación que se produce si la compañía sufre una pérdidas superiores al 10 % de los activos netos disponibles. Además, en este dominio se utiliza el mismo espacio de estados con selección de características y el acciones indicadas en la tabla 5.9, y utilizadas en la sección 5.3.3.

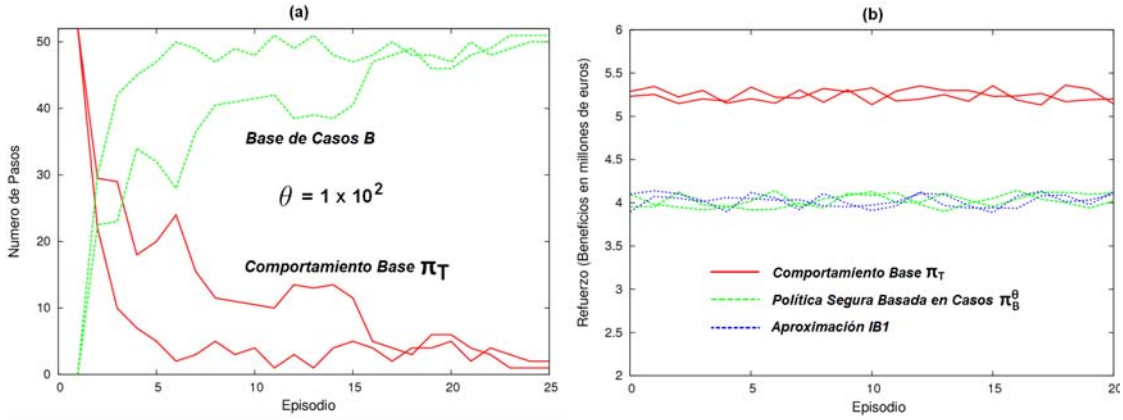


Figura 6.21: (a) Número de pasos por episodios ejecutados mediante la base de casos B y el comportamiento base π_T. (b) Refuerzo total acumulado por episodio obtenido por π_T, la Política Segura Basada en Casos π_B^θ y la aproximación *IBL*.

La figura 6.21 (a) muestra la evolución del número de pasos ejecutados por el comportamiento base, π_T, y la base de casos, B, durante dos ejecuciones independientes del paso *Modelado del Comportamiento Base* del algoritmo PI-SRL. Los valores de los parámetros θ y η se han calculado siguiendo los pasos detallados en la sección 6.4, y sus respectivos valores son 1×10^2 y 513. Con la ejecución de unos pocos episodios (alrededor de 25), se consigue aprender la Política Segura de la Base de Casos, π_B^θ. La figura 6.21 (b) muestra el rendimiento de la Política Segura Basada en Casos, π_B^θ, aprendida anteriormente, el comportamiento base, π_T y la aproximación IB1. En este caso, los beneficios medios por episodio obtenidos por la política π_T son 5.24 millones de euros, mientras que los obtenidos por la política π_B^θ, son 4.02 millones de euros. En el caso de la aproximación IB1, se han almacenado todos los casos generados por el comportamiento base, π_T, en una interacción con el entorno durante 25 episodios. Los experimentos demuestran que en el dominio del simulador empresarial SIMBA, al contrario que en los otros dominios, almacenar todos los casos durante un número suficiente de episodios permite obtener una política segura con un rendimiento similar a la política, π_B^θ con unos beneficios medios por episodio de 3.98 millones de euros. Una vez que la Política Segura Basada en Casos, π_B^θ, se ha aprendido correctamente, se ejecuta el paso de *Mejora del Comportamiento Base Aprendido*.

La figura 6.22 (a) muestra como PI-SRL con un nivel bajo y un nivel medio de riesgo

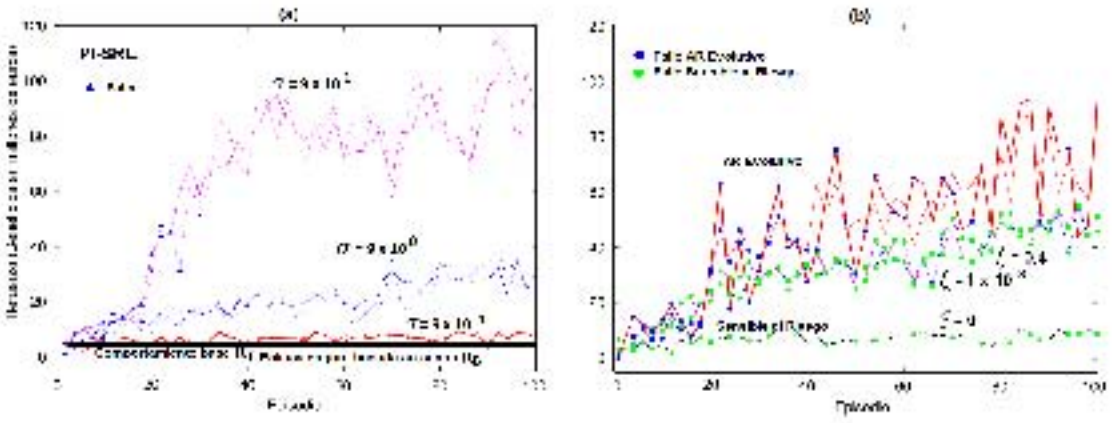


Figura 6.22: (a) El beneficio medio por episodio obtenido mediante diferentes configuraciones de riesgo (σ) por un agente PI-SRL jugando contra cinco agentes π_T. (b) El beneficio medio por episodio obtenido mediante la aproximación basada en aprendizaje por refuerzo evolutivo y la aproximación sensible al riesgo. En todos los casos, se marca cuando un episodio finaliza con fallo.

no producen bancarrotas de la compañía, aunque el rendimiento crece sólo ligeramente con respecto al rendimiento del comportamiento base, π_T. En cambio, PI-SRL con un nivel alto de riesgo aprende una política con un rendimiento muy superior al comportamiento base, π_T, obteniendo un número bajo de fallos durante el proceso de exploración. La figura 6.22 (b) muestra los resultados de algoritmo basado en aprendizaje por refuerzo evolutivo y la aproximación sensible al riesgo. La aproximación evolutiva es claramente la que obtiene un número mayor de fallos. En el caso de la aproximación sensible al riesgo, el número de fallos es insuficiente para aproximar correctamente la función de riesgo, ρ, necesaria para poder evitar estas situaciones en el futuro. La comparativa de los resultados en la figura 6.22 muestran que PI-SRL con $\sigma = 9 \times 10^1$ obtiene mejores políticas y con menos fallos que el resto de aproximaciones.

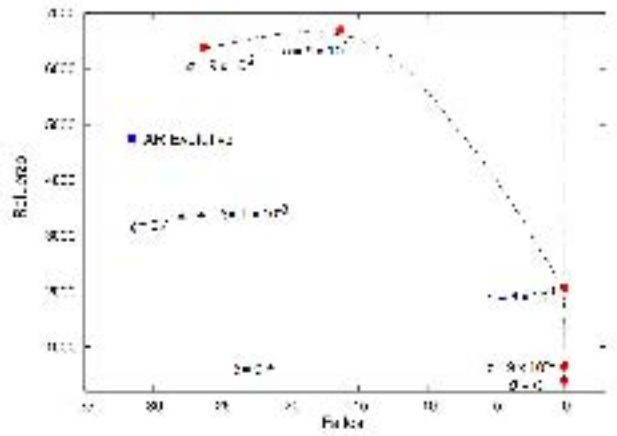


Figura 6.23: Número medio de fallos (bancarrotas de la compañía) y refuerzo acumulado total durante 100 episodios obtenidos por cada aproximación.

La figura 6.23 muestra una representación gráfica de las soluciones propuestas por las

diferentes aproximaciones en este dominio. En particular, la figura 6.23 muestra el número medio de fallos y el refuerzo acumulado total (los beneficios) obtenidos durante 100 episodios. Los datos mostrados en la figura 6.23 se han calculado a partir de la ejecución de 10 procesos de aprendizaje independientes para cada aproximación. Los círculos rojos se corresponden con las soluciones ofrecidas por el algoritmo PI-SRL con diferentes niveles de riesgo, los triángulos negros son las soluciones del algoritmo sensible al riesgo con diferentes valores del parámetro ξ y, por último, el cuadrado azul es la solución correspondiente al algoritmo basado en aprendizaje por refuerzo evolutivo. Como en los anteriores dominios, esta gráfica recoge el rendimiento del algoritmo PI-SRL cuando se utilizan dos niveles de riesgo adicionales no recogidos en la figura 6.22 (a): un nivel muy alto de riesgo ($\sigma = 9 \times 10^2$), y un nivel muy bajo de riesgo ($\sigma = 0$). Los experimentos en la figura 6.23 demuestran que PI-SRL con un nivel alto de riesgo ($\sigma = 9 \times 10^1$) obtiene el mayor beneficio total, 6693.58. Además, las soluciones de PI-SRL con un nivel muy bajo de riesgo ($\sigma = 0$), un nivel medio de riesgo ($\sigma = 9 \times 10^{-1}$) y un nivel medio de riesgo ($\sigma = 9 \times 10^0$), son las soluciones con un menor número de fallos, 0.0. No obstante, entre estas soluciones, la solución ofrecida por PI-SRL con un nivel medio de riesgo se prefiere sobre el resto puesto que alcanza un rendimiento superior. PI-SRL con un nivel muy alto de riesgo ($\sigma = 9 \times 10^2$) incrementa el número de fallos que se obtienen y reduce ligeramente los beneficios obtenidos con respecto a un nivel de riesgo alto ($\sigma = 9 \times 10^1$). Utilizando el criterio de comparación de Pareto, se puede afirmar que la solución de PI-SRL con un nivel alto de riesgo domina estrictamente a cualquier otra solución del resto de aproximaciones, PI-SRL $\sigma = 9 \times 10^1 \succ$ Aproximación Sensible al Riesgo y PI-SRL $\sigma = 9 \times 10^1 \succ$ AR Evolutivo, mientras que los soluciones propuestas por PI-SRL no son estrictamente dominadas por ninguna otra solución.

En este caso, no se proporcionan las gráficas con la evolución del espacio conocido debido a la elevada dimensionalidad de los espacios de estados y acciones para este dominio, que dificultan su representación.

6.6. Discusión

En este capítulo se ha descrito el algoritmo PI-SRL para el aprendizaje de políticas óptimas o cercanas a las óptimas en dominios con riesgo. El algoritmo permite la mejora de políticas mediante aprendizaje por refuerzo con exploración segura de los espacios de estados y acciones, reduciendo el número de veces que se visitan los estados de error. Las principales contribuciones de este algoritmo son la definición de una *Función de Riesgo Basada en Casos* y la utilización de un comportamiento base para realizar una exploración segura de los espacios de estados y acciones. La utilización de la función de riesgo definida en esta Tesis es posible porque la política se almacena en una base de casos, lo que facilita la distinción entre el espacio conocido por el agente y el espacio desconocido. La extracción de esta información no es posible si se emplean otros tipos de representación del conocimiento como el caso de las aproximaciones basadas en aprendizaje por refuerzo evolutivo [Martín and de Lope Asiaín, 2009; Koppejan and Whiteson, 2011], donde es difícil inferir cualquier conocimiento a partir de los pesos de las redes de neuronas empleadas. Se ha presentado una nueva definición de riesgo, completamente diferente de las que habitualmente se encuentran en el literatura. La noción de riesgo presentada en este capítulo, es independiente de la varianza del refuerzo acumulado, independiente de la función de refuerzo, y no requiere identificar estados de error ni aprender funciones de riesgo. Koppejan et al. [Koppejan and Whiteson, 2011] también utilizan una función para identificar los estados peligrosos pero, a diferencia de nuestra

función de riesgo, su definición requiere un fuerte conocimiento previo del dominio y su dinámica. Por otro lado, muchas de las aproximaciones que consideran el riesgo presentes en la literatura abordan problemas que no son completamente continuos [Geibel and Wysotzki, 2005], o sólo reportan resultados en un único dominio de experimentación [Koppejan and Whiteson, 2011], haciendo difícil creer que la aproximación sea fácilmente generalizable a otros dominios.

En este capítulo se ha estudiado en detalle el rendimiento del algoritmo PI-SRL y se ha demostrado su efectividad en cuatro dominios de experimentación diferentes completamente continuos y que crecen en complejidad en cuanto al número de dimensiones de los espacios de estados y acciones: el *automatic car parking problem*, el *Cart-Pole*, el *helicopter*, y el simulador empresarial SIMBA. Los experimentos presentados en este capítulo demuestran diferentes características acerca de las capacidades de aprendizaje del algoritmo PI-SRL.

1. El algoritmo obtiene soluciones de mejor calidad comparado con otro tipo de técnicas como demuestran los experimentos en la sección 6.5. Salvo en el dominio del *helicopter*, PI-SRL obtiene en todos los casos el refuerzo acumulado por episodio más elevado, y el menor número de fallos. Además, utilizando el principio de optimalidad de Pareto se puede afirmar, excluyendo la configuración de riesgo muy alto en el dominio del *Automatic Car Parking*, que las soluciones obtenidas por el algoritmo PI-SRL no son estrictamente dominadas por ninguna otra aproximación.

2. El algoritmo PI-SRL ajusta el espacio inicial conocido a regiones donde se encuentran mejores políticas seguras. El espacio inicial conocido resultante del primer paso del algoritmo se ajusta durante el segundo paso, mejorando de esta forma el comportamiento base aprendido durante el primer paso. Además, los experimentos demuestran que este proceso de ajuste puede comprimir el espacio conocido alejándolo del espacio de error (e.g., en el *Cart-Pole*, subsección 4.2.2, y en el *Helicopter*, subsección 4.2.3) o, en otras ocasiones, se puede requerir que el espacio conocido se ajuste cerca del espacio de error (e.g., *en el Automatic Car Parking*, subsección 4.2.1) puesto que las mejores políticas se encuentran en regiones cercanas al espacio de error.

3. PI-SRL funciona bien en dominios con muy diversa estructura y donde la función de valor puede variar brúscamente. Los dominios del *Automatic Car Parking*, el *Cart-Pole*, el *Helicopter* y el simulador empresarial SIMBA son dominios de muy diversa estructura. No obstante, el algoritmo PI-SRL demuestra comportarse de manera adecuada en cada uno de ellos. De hecho, el algoritmo alcanza un rendimiento similar a los mejores resultados publicados en el literatura como en el caso del dominio del *Helicopter* (subsección 4.2.3) y es capaz de aprender políticas cercanas al óptimo minimizando el número de fallos o, al menos, obteniendo un número menor de fallos que otras aproximaciones basadas en aprendizaje por refuerzo evolutivo. Además, incluso en dominios como el *Automatic Car Parking* donde la función de valor varía brúscamente debido a la presencia del obstáculo, los resultados demuestran que el algoritmo es capaz manejar adecuadamente esta dificultad.

4. El número de fallos depende de la distancia entre el espacio conocido y el espacio de error. Los experimentos en el *Cart-Pole* y en el *Helicopter* demuestran que el número de fallos depende de cómo de cerca el espacio conocido se encuentre del espacio de error. Debido a la estructura de estos dominios, el segundo paso del algoritmo tiende a concentrar el espacio conocido en el origen de coordenadas alejado del espacio de

error. Cuanto mayor es la distancia entre el espacio conocido y el espacio de error, menor es el número de fallos que se producen. Además, en el dominio del *Helicopter*, el espacio conocido está alejado desde el inicio del proceso de aprendizaje del espacio de error, y por lo tanto el número de fallos que se producen es bajo desde el principio. Por lo tanto, la distribución inicial del comportamiento base π_T en el espacio influye en el número de fallos que se obtienen posteriormente en el segundo paso del algoritmo.

5. PI-SRL es completamente seguro si sólo se ejecuta el primer paso del algoritmo. No obstante, procediendo de esta forma, el rendimiento del algoritmo se ve fuertemente limitado por las capacidades del comportamiento base. Si se desea mejorar el rendimiento del comportamiento base, es necesario ejecutar el subsecuente proceso exploratorio llevado a cabo durante el segundo paso del algoritmo. Puesto que no se posee un conocimiento completo del dominio y su dinámica, es inevitable que, durante este proceso de exploración, se visiten regiones desconocidas del espacio donde el agente podría alcanzar estados de error.

6. El parámetro de riesgo σ permite al usuario configurar el nivel de riesgo asumido. Geibel et al. [Geibel and Wysotzki, 2005] definen la función de valor como la suma ponderada del riesgo y el valor de forma $V_\xi^\pi(x) = \xi V^\pi(x) - \rho^\pi(x)$. El parámetro ξ puede ser considerado como su parámetro de riesgo en el sentido que determina la influencia de los valores V^π comparados con los valores ρ^π. Valores elevados de ξ significan una mayor relevancia de los valores V^π (incrementando la probabilidad de incurrir en daños), mientras valores bajos de ξ implican una mayor relevancia de la función de riesgo ρ^π (reduciendo la probabilidad de producir daños). De forma similar en nuestro algoritmo, se propone ir incrementando el valor del parámetro σ, teniendo en cuenta que valores más grandes conllevan una exploración más agresiva de los espacios de estados y acciones (incrementando de esta forma las posibilidades de encontrar mejores políticas de comportamiento), pero incrementando a la vez las posibilidades de dañar al agente.

7. PI-SRL funciona correctamente incluso cuando se utiliza una política inicial pobre con fallos. Los experimentos en la figura 6.20 en el dominio del *Helicopter* demuestran que el algoritmo PI-SRL es capaz de aprender una política cercana a la óptima a pesar de la inicialización pobre, como cuando se emplea una política libre de fallos para inicializar la base de casos. No obstante, la figura 6.20 también demuestra que si la política que se emplea en la inicialización genera muchos fallos, el algoritmo PI-SRL reduce su rendimiento y se produce un mayor número de fallos, aunque se logra aprender una política de comportamiento con un rendimiento superior. En este caso el algoritmo cae en un mínimo local, seguramente sesgado por la inicialización tan pobre.

A continuación, se discute la aplicabilidad del método PI-SRL, permitiendo al lector comprender apropiadamente los escenarios donde la aproximación propuesta puede ser aplicada. La aplicabilidad del algoritmo PI-SRL queda limitada a dominios que cumplen ciertas características:

1. En primer lugar, es necesario que el dominio satisfaga las dos asunciones que se describieron en la Sección 6.2. Por un lado, los estados similares en el dominio tienen que tener acciones óptimas similares. Por otro lado, acciones similares en estados similares

deberían producir efectos similares sobre el entorno. El hecho de que acciones similares deban conducir a estados similares significa asumir cierto grado de continuidad en la dinámica del sistema, que no todos los dominios disponen. No obstante, como se explica en la sección 6.2, ambas asunciones se consideran como asunciones lógicas derivadas de los principios de generalización en aprendizaje por refuerzo [Kaelbling *et al.*, 1996; Jiang, 2004], y que por tanto, requieren la mayoría de algoritmos.

2. La aplicabilidad del algoritmo PI-SRL está limitada por el tamaño de la base de casos B requerida para imitar el comportamiento base. No es posible aplicar la aproximación propuesta en tareas donde el primer paso del algoritmo PI-SRL, *Modelado del Comportamiento Base*, requiere un número prohibitivo de casos para imitar el comportamiento base. En este caso, se puede incrementar el umbral θ para hacer más restrictiva la adición de nuevos casos a la base de casos. No obstante, este incremento puede afectar negativamente al rendimiento final del algoritmo. En cualquier caso, los experimentos llevados a cabo en la sección 6.5 demuestran que comportamientos base relativamente simples se pueden imitar casi perfectamente empleando un número manejable de casos. Además, Floyd et al. [Floyd *et al.*, 2008; Floyd and Esfandiari, 2010] también han demostrado que las técnicas CBR pueden utilizarse con éxito en la imitación de comportamiento simples.

3. El algoritmo PI-SRL requiere la presencia de un comportamiento base. El método propuesto requiere la presencia de un comportamiento base que demuestre de manera segura la tarea que se trata de aprender. Este comportamiento base puede ser llevado a cabo por un experto humano, un agente programado a mano, un controlador automático. No obstante, la presencia de este comportamiento base no puede ser garantizado en todos los dominios.

Parte IV

Conclusiones y Líneas Futuras

Capítulo 7

Conclusiones

En este capítulo se ofrece un resumen de la Tesis desarrollada con el fin de establecer las principales conclusiones que se han obtenido. Este capítulo está compuesto de dos partes. La primera presenta los puntos claves extraídos de los apartados de discusión que se han presentado en esta Tesis. La segunda parte recoge las aportaciones que realiza esta Tesis en el ámbito científico.

7.1. Resumen

En aprendizaje por refuerzo, son pocos los trabajos que aborden problemas con espacios de estados y acciones continuos y de grandes dimensiones. En estos trabajos, se suele aproximar el valor de la función de valor-estado (o la función de valor-acción) mediante el uso de alguna técnica de aproximación de funciones. La técnica de aproximación de funciones más utilizada son las redes de neuronas [Prokhorov and Wunsch, 1997; van Hasselt and Wiering, 2007], aunque también se suelen utilizar técnicas de razonamiento basados en casos, CBR [Santamaría *et al.*, 1998; Smart and Kaelbling, 2000]. El principal inconveniente de todas estas técnicas es extraer información (de la red de neuronas o de la base de casos) sobre cuál es la mejor acción a ejecutar en cada momento. En algunos trabajos se utiliza otro método de aproximación de funciones, normalmente otra red de neuronas, para representar la política de comportamiento dando lugar a las arquitecturas Actor-Crítico que se presentaron en la sección 2.3.1. El inconveniente de estas aproximaciones es que es necesario aproximar dos funciones en lugar de sólo una, y sólo se ha probado con éxito en dominios de dimensionalidad reducida [van Hasselt and Wiering, 2007]. En otros trabajos, en cambio, se itera por un conjunto de acciones discretos obteniendo el valor Q asociado, y seleccionando aquella acción con un valor Q mayor [Santamaría *et al.*, 1998]. Las técnicas evolutivas revisadas [Martín and de Lope Asiaín, 2009; Koppejan and Whiteson, 2011] son una forma eficaz de buscar directamente sobre el espacio de políticas aunque el crecimiento del problema hace crecer la complejidad de las redes de neuronas que se tratan de evolucionar dificultando la convergencia a políticas de comportamiento óptimas o cercanas a las óptimas. En general, abordar un problema de estas características sin ningún tipo de conocimiento inicial plantea dos grandes inconvenientes para todas estas técnicas:

1. El alto número de dimensiones del espacio de acciones dificulta cualquier tipo de exploración en busca de las acciones óptimas o cercanas a las óptimas. Una estrategia

de exploración típica como $\epsilon - greedy$ podría ralentizar excesivamente el proceso de aprendizaje mediante la búsqueda aleatoria en el inmenso espacio de acciones. Además, una exploración aleatoria o inadecuada produce en la mayor parte de las ocasiones acciones incoherentes, que aún siendo válidas, no conducen al agente a explorar las regiones del espacio de estados más "interesantes", es decir, aquellas zonas del espacio de estados que resultan útiles para posteriormente aprender una política de comportamiento óptima o cercana a la óptima.

2. Por otro lado, abordar directamente un problema continuo y con un gran número de dimensiones sin ningún tipo de simplificación requiere la utilización de complejas estructuras con un alto número de parámetros que requieren ser ajustados y que dificultan la convergencia de los algoritmos (e.g., redes de neuronas de gran complejidad).

Los algoritmos G-VQQL y CMAC-VQQL propuestos en esta Tesis solucionan los problemas detectados anteriormente cuando nos encontramos con dominios continuos de grandes dimensiones:

1. La exploración del espacio mediante la utilización de un comportamiento base en dominios donde la exploración aleatoria no es aplicable o no es adecuada permite recoger experiencia de las zonas más "interesantes" de los espacios de estados y acciones, a partir de las cuales se construyen las discretizaciones. Las discretizaciones resultantes mantienen las mismas relaciones estadísticas existentes entre las características de las acciones presentes en el comportamiento base, lo que permite no tener que explorar regiones inútiles de los espacios de estados y acciones acelerando considerablemente la convergencia de los algoritmos. Por lo tanto, de algún modo, este tipo de exploración incorpora cierto conocimiento inicial necesario sobre el problema que se trata de resolver y que difícilmente se podría resolver si él.

2. Los métodos que aplican discretización en los espacios de estados y acciones o sólo en el espacio de acciones, y hacen uso de una correcta selección de características para los estados, reducen la complejidad de problemas de grandes dimensiones y continuos, que serían muy difíciles de abordar directamente sin ningún tipo de simplificación. Estos métodos presentan una clara ventaja sobre los métodos anteriores, puesto que permiten determinar cuál es la mejor acción a ejecutar dado un estado con un coste computacional muy pequeño, y sin necesidad de aproximar funciones adicionales, o de realizar cálculos complejos.

Queda de manifiesto en esta Tesis que los algoritmos G-VQQL y CMAC-VQQL presentados constituyen una forma eficaz de simplificar problemas continuos de grandes dimensiones. Además, los algoritmos propuestos han demostrado también comportarse correctamente en dominios de baja dimensionalidad donde un comportamiento aleatorio es suficiente para explorar de forma adecuada el espacio de estados y el espacio de acciones para construir los cuantificadores (sección 5.3.1.1), o en dominios de mayores dimensiones donde el comportamiento aleatorio permite realizar algún tipo de exploración del espacio (sección 5.3.3.1). El éxito de ambos algoritmos radica en la exploración adecuada que se haga del espacio para obtener discretizaciones apropiadas del espacio de estados y el espacio de acciones. En cuanto a las características deseables de los algoritmos propuestos en esta Tesis listados en la sección 3.2, ambos métodos cumplen todas las propiedades salvo la de *Continuidad* debido a la discretización que se hace de los espacios de estados y acciones en el caso del algoritmo G-VQQL, y del espacio de acciones en el caso del algoritmo

CMAC-VQQL. Estas discretizaciones hace que existan fronteras bruscas en la función de valor-acción aproximada. Además, la utilización de técnicas de discretización en el espacio de estados pueden producir la pérdida de la propiedad de Markov [Moore and Atkeson, 1993; Justin Boyan, 1995], problema que se ve acentuado puesto que las técnicas propuestas también discretizan el espacio de acciones. No obstante, este problema puede entenderse como un problema de introducción de estocasticidad en el dominio que hará que no sea posible alcanzar políticas de acción óptimas. En cualquier caso, los experimentos que se han llevado a cabo demuestran la efectividad de los modelos propuestos en dominios complejos.

En cuanto al aprendizaje por refuerzo en dominios con riesgo, el algoritmo PI-SRL ha obtenido mejores resultados que otro algoritmo sensible al riesgo [Geibel and Wysotzki, 2005] y que un algoritmo basado en aprendizaje por refuerzo evolutivo [Martín and de Lope Asiaín, 2009] en dominios de experimentación de muy distinta naturaleza, y dónde la función de valor-estado puede cambiar bruscamente. El algoritmo de Geibel et al. [Geibel and Wysotzki, 2005] necesita identificar estados de error, y aproximar previamente una función de riesgo, ρ, antes de poder evitar que el agente o el sistema de aprendizaje sufra daños. Por lo tanto, requiere visitar continuamente estados de error con el fin de aproximar la función de riesgo. En cuanto al algoritmo de Martin et al. [Martín and de Lope Asiaín, 2009], parece un algoritmo *ad-hoc* difícilmente generalizable a otros dominios de experimentación [García *et al.*, 2011], sobre todo si éstos contienen un gran número de variables para los estados y las acciones (lo que complicaría la topología de las redes de neuronas y dificultaría el proceso de búsqueda de cada vez mejores comportamientos). Los experimentos demuestran que el algoritmo PI-SRL es completamente seguro si sólo se ejecuta el primer paso de *Modelado del Comportamiento Base*. No obstante el algoritmo padece el mismo problema que las aproximaciones basadas en Aprendizaje por Demostración clásicas: el rendimiento final del algoritmo se ve fuertemente limitado por las demostraciones que es capaz de realizar el comportamiento base utilizado, cuyo rendimiento se considera subóptimo. En el caso de que se quiera mejorar el rendimiento del agente más allá de las demostraciones del comportamiento base, es inevitable ejecutar algún proceso de exploración de los espacios de estados y acciones. En nuestro caso, esta exploración se realiza de forma segura con el apoyo del comportamiento base en el segundo paso del algoritmo PI-SRL, *Mejora del Comportamiento Base Aprendido*. Al principio de este proceso de exploración el agente parte de un espacio conocido en el cuál es capaz de ejecutar un comportamiento seguro aunque subóptimo. El proceso de exploración conduce al agente a los límites de este espacio conocido y le fuerza a ir más allá, al espacio de estados desconocidos, donde se recurre al comportamiento base para volver a una situación de estados conocidos. Poco a poco el agente adapta su espacio conocido a regiones donde se encuentran mejores políticas seguras, mientras olvida las regiones del espacio conocido que ya no le son útiles. No obstante, puesto que no se tiene un conocimiento completo del dominio y su dinámica, es imposible evitar por completo durante este proceso de exploración visitar algunos estados de error. El parámetro de riesgo σ permite al usuario configurar el nivel de riesgo asumible. En los experimentos, se propone ir incrementando el valor de este parámetro, teniendo en cuenta que valores más grandes conllevan una exploración más agresiva de los espacios de estados y acciones (incrementando de esta forma las posibilidades de encontrar mejores políticas de comportamiento), pero incrementando a la vez las posibilidades de dañar al agente. En cualquier caso, los experimentos demuestran que PI-SRL es capaz de reducir el número de veces que se visitan en comparación a una aproximación basada en aprendizaje por refuerzo evolutivo, y una aproximación sensible al riesgo.

7.2. Aportaciones

Se consideran aportaciones de la presente Tesis en el ámbito científico a aquellos métodos desarrollados que proporcionan un avance al estado del arte. Esta Tesis contribuye en el campo del aprendizaje por refuerzo en dominios continuos y de grandes dimensiones en los puntos que se describen a continuación:

1. El modelo G-VQQL [García *et al.*, 2012] para la representación de espacios de estados y acciones de naturaleza continua.

   a) A lo largo de esta Tesis se ha planteado la utilización de técnicas del vecino más cercano para realizar una discretización tanto del espacio de estados como del espacio de acciones.

   b) Mediante la técnica propuesta se ha investigado el efecto de la utilización de técnicas de discretización sobre el espacio de estados y el espacio de acciones, incorporando de esta forma información valiosa al estado del arte que se había centrado principalmente en la discretización del espacio de estados únicamente [Fernández and Borrajo, 1999; Stone *et al.*, 2005].

   c) Se ha demostrado que es posible aplicar con éxito el algoritmo G-VQQL en dominios estocásticos, continuos y de grandes dimensiones, obteniendo políticas de comportamiento cercanas al óptimo.

   d) El algoritmo funciona de forma adecuada en dominios donde un comportamiento aleatorio permite explorar de forma eficiente el espacio de estados. No obstante, en dominios donde esta exploración no es posible, se ha demostrado que se puede realizar una exploración mediante un comportamiento base, la cual permite obtener discretizaciones de zonas relevantes de los espacios de estados y acciones, evitando explorar zonas inútiles, y acelerando enormemente la convergencia del algoritmo.

2. El modelo CMAC-VQQL [García *et al.*, 2010; García *et al.*, 2012] para la representación de espacios de estados y acciones de naturaleza continua.

   a) Este método combina con éxito técnicas de aproximación de funciones, CMAC, con técnicas de discretización basadas en cuantificación vectorial, VQ.

   b) Al igual que el algoritmo G-VQQL, se ha demostrado que CMAC-VQQL permite obtener políticas de comportamiento de alto rendimiento, en dominios estocásticos, continuos y de grandes dimensiones.

   c) Como en el caso del algoritmo G-VQQL, se ha demostrado que la utilización de comportamientos base ayuda a realizar una exploración sobre zonas relevantes del espacio, que después facilitan el proceso de aprendizaje.

Además, esta Tesis contribuye en el campo del aprendizaje por refuerzo en entornos con riesgo en los siguientes puntos:

1. El algoritmo **P**olicy **I**mprovement through **S**afe **R**einforcement **L**earning [García and Fernández, 2011], PI-SRL, que permite alcanzar rendimientos cercanos al óptimo minimizando el número de daños que sufre el agente o el sistema de aprendizaje. El rendimiento del algoritmo PI-SRL se ha comparado satisfactoriamente con otras técnicas de aprendizaje por refuerzo sensibles al riesgo, y con algoritmos de aprendizaje

por refuerzo evolutivos que incorporan a la población inicial varios comportamientos expertos.

2. La definición del algoritmo PI-SRL aporta dos grandes contribuciones al campo del aprendizaje por refuerzo con Riesgo y que permiten llevar a cabo una exploración segura de los espacios de estados y acciones.

   a) La definición de *Función de Riesgo de la Base de Casos* (*Case Base Risk Function*), que permite determinar cuándo un estado es de riesgo y, por tanto, es mejor ejecutar acciones conservadoras, de cuándo no lo es y se pueden ejecutar comportamientos más agresivos con el fin de realizar una exploración de los espacios de estados y acciones mayor. El uso de esta función es posible porque la política de comportamiento se almacena en una base de casos, con lo cual se puede calcular la distancia entre el *espacio conocido* y el *espacio desconocido*.

   b) La definición de *Comportamiento Base* (*Baseline Behavior*), que permite regresar el sistema al espacio de estados conocidos cuando se han alcanzado estados desconocidos.

3. Se ha aportado una nueva definición de *riesgo* que no tiene relación con las tradicionales definiciones basadas en la varianza del refuerzo acumulado [Heger, 1994; Coraluppi and Marcus, 1999], que es independiente de la función de refuerzo [Hans, Alexander *et al.*, 2008] y que no requiere identificar estados de error ni aprender previamente funciones de riesgo [Geibel, 2001; Geibel and Wysotzki, 2005]. La noción de riesgo descrita en esta Tesis se basa en la distancia entre el *espacio conocido* y el *espacio desconocido* por el agente.

4. La interpretación de la evolución del espacio conocido en cada uno de los dominios enriquece el análisis presentado y permite relacionar el número de fallos con la distancia entre el espacio conocido y el espacio de error.

5. Se ha estudiado en detalle el rendimiento del algoritmo y se ha probado su efectividad en cuatro dominios con espacios de estados y acciones continuos y que crecen en complejidad (i.e., crece el número de variables que describen los espacios de estados y acciones) probando de esta forma que el algoritmo es aplicable a dominios de naturaleza muy diversa: el Automatic Car Parking Problem, el dominio *Cart-Pole*, el dominio del *Helicopter*, y el simulador de gestión empresarial SIMBA. Muchas de las aproximaciones de aprendizaje por refuerzo con riesgo existentes en la literatura abordan, o bien problemas que no son completamente continuos [Geibel and Wysotzki, 2005], o solo reportan resultados en un único dominio [Koppejan and Whiteson, 2011] (sin demostrar que la aproximaxión generalice fácilmente a dominios arbitrarios).

Además la presente Tesis aporta un nuevo dominio para la investigación en aprendizaje por refuerzo, el simulador empresarial SIMBA. La adaptación de RL-Glue [Tanner and White, 2009] permite considerar el dominio como un marco de investigación multi-agente donde, además, participantes humanos pueden competir contra agentes software (apéndice A). Como se describió en la Sección 4.2.5, es un dominio generalizado donde cientos de parámetros pueden modificar el comportamiento del entorno de un episodio al siguiente. Al considerarlo como un dominio multi-agente, pueden surgir comportamientos competitivos

o cooperativos (dando lugar a oligopolios) no establecidos a priori. Por todo esto, el dominio SIMBA podría ser incluido en la colección de dominios de evaluación que utiliza la comunidad.

Capítulo 8

Líneas Futuras

El problema de aprendizaje por refuerzo en dominios con espacios de estados y acciones continuo y de grandes dimensiones sigue siendo un problema abierto, por lo que la evolución de las técnicas presentadas en esta Tesis y otras técnicas de aprendizaje plantean nuevas líneas de investigación. Las limitaciones que se han encontrado en los algoritmos G-VQQL y CMAC-VQQL también plantean trabajos futuros para conseguir mejoras en el rendimiento de los algoritmos. En este sentido las propuestas de trabajo futuro son:

1. Convertir los algoritmos G-VQQL y CMAC-VQQL a versiones *continuas*. Santamaría et al. [Santamaría *et al.*, 1998] realizan la media ponderada de los valores Q de los estados y acciones vecinos con el fin de calcular el valor Q de un nuevo estado y acción, tal como se describió en la sección 2.3.1. Por otro lado, Millán et al. [Millán *et al.*, 2002] presentan el algoritmo *Continuous Q-Learning*, que de forma similar a Santamaría aunque esta vez para las acciones, ofrece la forma de obtener acciones continuas a partir de los pesos de las acciones discretas vecinas.

2. Uno de los principales inconvenientes que ofrece la cuantificación vectorial es que necesita un fuerte proceso previo de exploración tanto del espacio de estados como del espacio de acciones. En muchos dominios esta exploración inicial se puede realizar de forma aleatoria, permitiendo después a la cuantificación vectorial obtener una buena representación alternativa y reducida de los espacios de estados y acciones. No obstante, en muchos dominios la exploración aleatoria solo conduce a explorar parcialmente los espacios de estados y acciones [Martinez-Gil *et al.*, 2011], produciendo después problemas en la aproximación de la función de valor. En estos casos es posible utilizar, como se ha demostrado en esta Tesis, comportamientos base o predefinidos que permitan guiar el proceso de exploración. Por lo tanto, un posible trabajo futuro sería analizar las diferencias de rendimiento que obtienen los algoritmos, en función de la utilización de distintos comportamientos base (o de distintas técnicas de exploración) para la obtención del conjunto representativo de estados y acciones.

3. Si es posible utilizar comportamientos base para guiar el proceso de exploración para suministrar un conjunto de estados y acciones a la cuantificación vectorial previo al proceso de aprendizaje, quizá también sería posible utilizar este comportamiento base en el proceso de aprendizaje para acelerar la convergencia y mejorar el rendimiento del algoritmo, siguiendo un esquema similar al de otros trabajos [Latzke *et al.*, 2006]. Por ejemplo, durante el proceso de aprendizaje se podrían intercalar episodios de ex-

ploración normales con episodios guiados mediante el comportamiento base [Driessens and Džeroski, 2004].

4. Creación de nuevas combinaciones de algoritmos que permitan abordar problemas con espacio de estados continuos y de grandes dimensiones. En esta Tesis se ha propuesto combinar CMAC con VQ, pero será posible utilizar otros métodos de forma que se siguiera manteniendo el esquema de aprendizaje. Por ejemplo, se podrían utilizar árboles *kd* para discretizar el espacio de estados y seguir manteniendo a la cuantificación vectorial como técnica de discretización del espacio de acciones.

5. Estudiar de forma detenida, y desde un punto de vista económico, las decisiones tomadas por los agentes que utilizan los algoritmos G-VQQL y CMAC-VQQL para aprender nuevas estrategias de toma de decisiones. Los expertos en economía consideran que hay diferentes grupos estratégicos en un mercado, cada uno con un rendimiento diferente [Leask and Parker, 2007]. Un importante campo de investigación, consiste en identificar qué estrategia es mejor que las otras. SIMBA permite crear diferentes configuraciones de mercado con diferentes evoluciones de comportamiento, que pueden ayudar a identificar nuevas estrategias de negocio en situaciones muy diferentes y que podrían ser utilizadas en el mundo real.

En el caso del aprendizaje por refuerzo en entornos con riesgo, las limitaciones puestas de manifiesto en esta Tesis sobre el algoritmo PI-SRL, posibilitan nuevas líneas de investigación. En este caso, se prevé:

1. La graduación automática del parámetro de riesgo σ a lo largo del proceso de aprendizaje. Por ejemplo, podría ser interesante explotar el hecho de que el espacio conocido está alejado del espacio de error, para incrementar el valor de σ, o disminuirlo en el caso de que estén cerca para reducir la exploración y asumir menos probabilidades de daño. Otra estrategia podría ser incrementar el valor del parámetro de riesgo σ, si en los últimos t episodios la tasa de daños (i.e., el número de colisiones, bancarrotas,...) cae por debajo de cierto umbral τ (con el fin de explorar de una forma más agresiva), o disminuir σ en el caso de que la tasa de daños sea más alta que el parámetro τ. No obstante, todas estas aproximaciones añaden nuevos parámetros y el problema inherente de especificar sus valores correctos.

2. La integración del algoritmo PI-SRL en procesos de aprendizaje reales, con robots reales. Esto a su vez plantea nuevos desafíos puesto que uno de los grandes inconvenientes de la utilización de robots es la incertidumbre de las percepciones que tienen del mundo real, debido a sus características mecánicas y eléctricas y a la complejidad del entorno. No obstante, esta incertidumbre puede considerarse como ruido adicional que convierte al dominio en más estocástico de lo que ya era. En esta línea de investigación se analizará si el algoritmo PI-SRL es capaz de aprender minimizando los daños, en entornos reales.

3. La utilización de mejores técnicas de exploración que la exploración aleatoria siguiendo una distribución Gaussiana empleada en esta Tesis. Wyatt [Wyatt, 1997] propone en su Tesis diferentes formas de exploración más inteligentes que podrían ser integradas en al algoritmo PI-SRL para incrementar el rendimiento obtenido.

4. La adición de una nueva fase anterior a las enumeradas en esta Tesis en la cual se aprenda una métrica de similitud específica para el dominio en el cual se trata de aprender. En esta Tesis se ha empleado la distancia Euclídea como métrica de similitud para los estados, aunque el uso de métricas específicas podrían incrementar el rendimiento del algoritmo propuesto [Taylor *et al.*, 2011].

5. Utilización de una función de riesgo de naturaleza continua. La función utilizada en el capítulo 6 tiene dos valores lógicos: 1, y 0. El valor 1 se asocia a una situación de riesgo (estados desconocidos) y el valor 0 se asocia a situaciones sin riesgo (estados conocidos). Una futura línea de investigación puede tratar de extender la función de riesgo para manejar el concepto de riesgo parcial, donde el valor de riesgo puede variar de forma continua en un rango que va desde 1 (completamente arriesgado) a 0 (sin riesgo). La nueva función de riesgo se podría calcular utilizando la ecuación 8.1.

$$\varrho(s_q) = 1 - \frac{1}{1 + \mathrm{e}^{\frac{K}{\theta} \times (\mathrm{mín}_{1 \leq i \leq \eta}\, dist(s_q, s_i) - \theta)}} \tag{8.1}$$

La ecuación 8.1 representa una función sigmoidal con centro en θ. El parámetro K se utiliza para controlar el ancho de esta función sigmoidal. Un valor elevado de K genera una función sigmoidal "estrecha" lo que implica variaciones bruscas del valor de riesgo, mientras que un valor menor de K genera una función sigmoidal más amplia lo que implica variaciones más suaves del valor de riesgo. El valor de la función de riesgo para un nuevo estado percibido del entorno s_q se calcula utilizando la distancia Euclídea entre el estado s_q y el estado más cercano en la base de casos B. El valor de riesgo varía gradualmente entre 1 y 0: una distancia muy alta implica un valor de riesgo cercano a 1, mientras que una distancia menor implica un valor de riesgo cercano a 0. La figura 8.1 muestra la comparación entre la función de riesgo utilizada en el capítulo 6 y la nueva función de riesgo continua presentada en la ecuación 8.1 con $K = 6$ y $\theta = 0{,}3$. Mientras la función de riesgo en la figura 8.1 (a) es una función escalón, la nueva función de riesgo en la figura 8.1 (b) permite obtener transiciones más suaves entre las situaciones sin riesgo (*Risk-free*) y las situaciones consideradas con un nivel alto de riesgo (*Risk*).

Con esta nueva definición de la función de riesgo, cuando se recibe un nuevo estado s_q del entorno, la nueva acción a se calcula como la suma ponderada de la acción derivada del comportamiento base y la acción derivada de la base de casos B como muestra la ecuación 8.2.

$$a = \varrho(s_q)\pi_T + (1 - \varrho(s_q))(\pi_B^{\theta} \times rndGaussian(a_c, \sigma)) \tag{8.2}$$

donde la función $rndGaussian(a_c, \sigma)$ genera números aleatorios siguiendo una distribución Gaussiana centrada en a_c (i.e., la acción actual) con una desviación estándar de σ. Se espera que con esta función de riesgo las acciones en la base de datos se modifiquen de forma más suave que cuando se emplea la función de riesgo discreta, y por lo tanto se reduzca el número fallos. También existen otros métodos que podrían ser integrados en PI-SRL y que son utilizados para determinar cuándo un estado es conocido o desconocido como el *knownness criterion* utilizado por Nouri et al. [Nouri and Littman, 2008] derivado del trabajo de Kakade [Kakade, 2003].

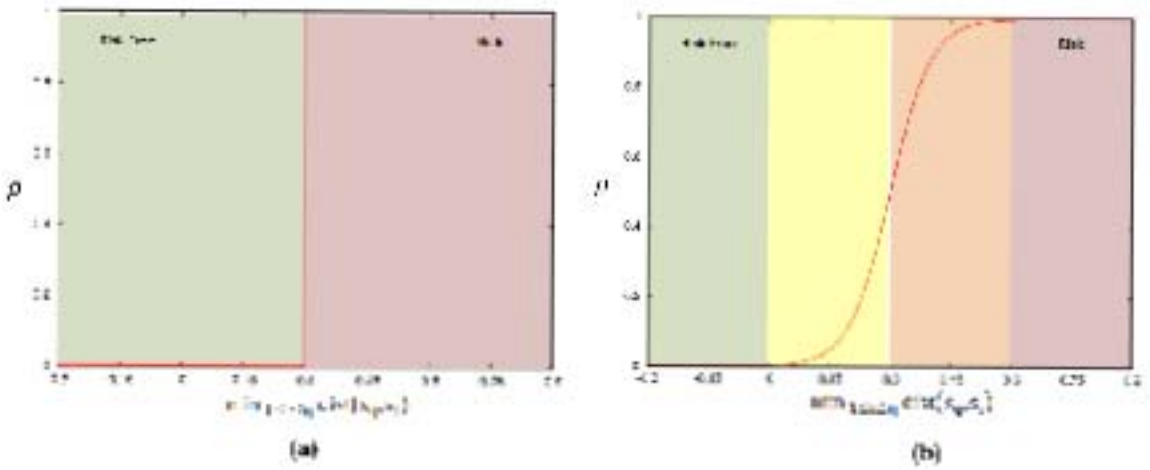


Figura 8.1: (a) La función de riesgo en escalón utilizada en el capítulo 6 (b) La función de riesgo sigmoidal que genera valores continuos entre 0 y 1.

6. Utilización de técnicas de aproximación diferentes a CBR. El inconveniente de esta línea de investigación es que la técnica debe permitir identificar entre el espacio conocido por el agente y el desconocido, con el fin de identificar posibles situaciones de riesgo.

7. Definición de nuevas funciones de riesgo que no requieran conocimiento previo del dominio, que no requieran ser aprendidas previamente, y que sean fácilmente configurables.

Parte V

Apéndices

Apéndice A

Resultados Adicionales en SIMBA

Este capítulo presenta los resultados obtenidos en SIMBA cuando se utiliza como un marco de investigación multi-agente. La evaluación se llevará a cabo empleando diferentes escenarios de juego donde todas las compañías son gestionadas por agentes G-VQQL que aprenden al mismo tiempo. El objetivo es evaluar la influencia de cada escenario en las capacidades de aprendizaje de los agentes, a la vez que se trata de estudiar, desde un punto de vista económico, la naturaleza de sus comportamientos. SIMBA es un entorno multi-agente donde la cooperación y la competición entre los agentes pueden coexistir dependiendo de la estrategia de negocio que se utilice. La competición entre los agentes puede producir estrategias de gestión agresivas y reactivas, mientras que la cooperación puede producir oligopolios. La competición y la cooperación típicamente emergen en el mercado, y en muchos casos, no están preestablecidas. De hecho, en muchos países los acuerdos entre compañías están prohibidos con el fin de preservar la competitividad del mercado.

Finalmente, la sección A.2 presenta el análisis del comportamiento del agente G-VQQL desde un punto de vista económico realizado por un experto. Las siguientes secciones describen en detalle cada uno de estos experimentos. En cada uno de estos experimentos, se ha empleado la estrategia de exploración $\epsilon - greedy$, reduciendo el valor de ϵ desde 1 (comportamiento aleatorio) a 0 (comportamiento totalmente avaricioso) en un valor de 0.015 por simulación. Los parámetros utilizados para el algoritmo G-VQQL son: $\alpha = 0{,}125$ y $\gamma = 0{,}9$. En los experimentos todas las gráficas muestran la misma información; el eje y muestra el resultado del ejercicio (i.e., los beneficios obtenidos por la compañía en millones de euros) obtenidos en la simulación específica definida en el eje x. Cada simulación está compuesta de 52 periodos o pasos de aprendizaje (equivalente a 12 años puesto que cada periodo equivale a un periodo real de 3 meses), mientras que cada proceso de aprendizaje está compuesto por 90 simulaciones o episodios.

A.1. Aprendizaje de Múltiples Agentes

En este caso, todas las compañías son gestionadas por un agente G-VQQL y el aprendizaje se realiza al mismo tiempo. Este escenario se considera un escenario de aprendizaje *multi-agente*, aunque cada agente aprende independientemente de los otros agentes involucrados en la simulación. En este caso, uno puede hablar de aprendizaje multi-agente en la medida en que lo que un agente aprende afecta y es afectado por los agentes vecinos. A partir de este hecho, se pueden esperar dos tipos de comportamientos diferentes. Por un lado, cabe esperar que todos los agentes converjan a políticas de comportamiento similares

obteniendo beneficios similares mediante el uso de estrategias de negocio no agresivas. En este caso, se espera obtener por tanto, algún tipo de equilibrio. Modelos tradicionales en economía soportan esta idea. Por ejemplo, el modelo de competición de Cournot [Cournot, 1927] es un modelo que describe la estructura de un mercado. La asunción esencial de este modelo es que cada compañía tiene como objetivo la maximización de sus beneficios, basándose en la expectativa de que su propia decisión no tendrá un efecto en las decisiones de sus rivales. El equilibrio de Cournot ocurre cuando la producción de cada compañía maximiza sus beneficios relativos a la producción del resto de compañías. Un caso especial de equilibrio de Cournot se produce cuando las compañías dividen el mercado en partes iguales, obteniendo los mismos beneficios. Por otro lado, se puede esperar que los agentes diverjan a diferentes políticas, cada una de las cuales hace obtener al agente correspondiente un determinado beneficio. En otras palabras, algunos agentes adquieren ventaja durante el proceso de exploración lo que les hace aprender mejores políticas en términos de beneficios. Las siguientes subsecciones describen las políticas de comportamiento aprendidas en este escenario multi-agente.

A.1.1. Aprendizaje de los Agentes al Mismo Tiempo

La figura A.1 muestra los resultados obtenidos por diferentes agentes G-VQQL - cada uno manejando una compañía - que están aprendiendo simultáneamente. La figura muestra dos procesos de aprendizaje diferentes. Como se describió anteriormente, todos los agentes utilizan la misma estrategia de exploración/explotación.

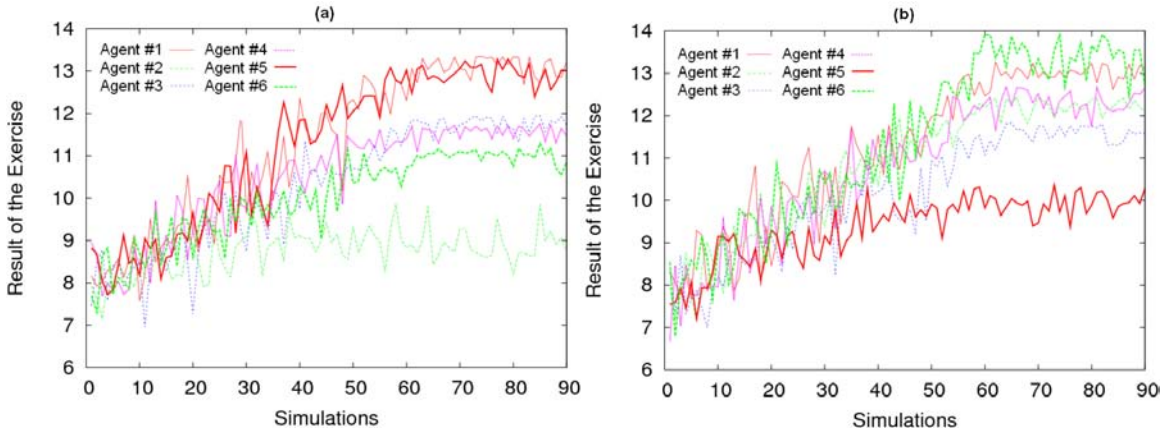


Figura A.1: Esta figura muestra el rendimiento de seis agentes aprendiendo al mismo tiempo y utilizando los mismo parámetros de aprendizaje (i.e., algoritmo, aproximación de generalización y estrategia de exploración).

En ambos casos, los agentes G-VQQL divergen a políticas de comportamiento diferentes con un determinado nivel de beneficios. En otras palabras, algunos agentes toman ventaja durante el proceso de exploración y son capaces de obtener una política mejor en términos de los beneficios obtenidos. Este comportamiento ocurre en ambos procesos de aprendizaje, aunque no son siempre las mismas compañías las que obtienen mejores resultados debido a las diferencias que se producen durante el proceso de exploración estocástico. Por ejemplo, en la gráfica de la izquierda, dos agentes logran obtener unos resultados de aproximada-

mente 13 millones de euros, mientras que otra compañía, incapaz de aprender una buena política de comportamiento, alcanza unos beneficios máximos de sólo 9 millones de euros. No obstante, en la gráfica de la derecha, hay un agente que obtiene más beneficios que ningún otro (i.e., alrededor de 13.5 millones de euros), mientras que el peor agente sólo obtiene alrededor de 10 millones de euros. A partir de estos resultados, se puede decir que entre los agentes emerge un comportamiento competitivo. Los diferentes agentes G-VQQL intentan aprender a ser cada vez más inteligentes que sus oponentes - donde "más inteligente" significa obtener mayores beneficios - creando, de este modo, un mercado altamente competitivo. En este mercado extremadamente competitivo, los diferentes agentes aprenden a mejorar sus beneficios. Si esto no fuera así, cada uno de estos agentes aprenderían políticas de comportamiento similares, con un rendimiento similar (como se demuestra en los resultados de la Figura 5.26).

A.1.2. Transferencia de Políticas Entre Diferentes Escenarios

Para evaluar el rendimiento de los agentes aprendidos anteriormente un mercado no competitivo, se emplea el siguiente escenario de evaluación. En este experimento, no competitivo significa que los agentes no aprenden, es decir, no tratan de mejorar su comportamiento que se mantiene constante durante cada simulación. Los agentes sólo utilizan la política aprendida anteriormente en el mercado extremadamente competitivo donde todos los agentes aprendieron al mismo tiempo. Cada agente G-VQQL tomado de la gráfica de la izquierda de la figura A.1, juega de forma avariciosa y sin modificar su política (i.e., sin aprender) contra 5 agentes pre-programados. La figura A.2 muestra estos 6 procesos de evaluación.

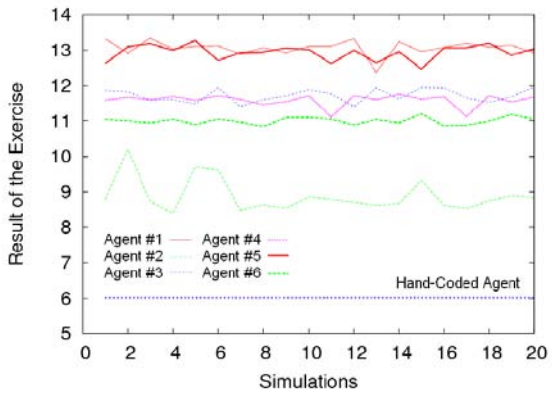


Figura A.2: La figura muestra el rendimiento de las políticas de los agentes aprendidas en el experimento de la figura A.1. Todos los agentes juegan de forma avariciosa sin mejorar su política contra 5 agentes pre-programados.

Como se puede observar, en este mercado no competitivo, los diferentes agentes G-VQQL obtienen los mismos beneficios que los obtenidos en el mercado competitivo. Este experimento demuestra que si el agente G-VQQL aprende un comportamiento específico en un mercado competitivo, el agente mantiene el comportamiento en un entorno no competitivo, donde los oponentes siguen el mismo comportamiento conservador. Por lo tanto, se puede afirmar que las políticas aprendidas se pueden transferir correctamente de un escenario de juego a otro, lo que demuestra la robustez de las políticas cuando se cambia de escenario.

A.1.3. Aprendizaje en un Dominio Generalizado

Como se describe en la sección 4.2.5, SIMBA es un dominio generalizado. Cuenta con un amplio conjunto de parámetros Θ y la modificación de cualquiera de estos parámetros puede modificar completamente el comportamiento del mercado. En el siguiente conjunto de evaluaciones, se modifica la tasa de incremento de ventas de diferentes formas. Además, los agentes G-VQQL en la figura A.3 (a), presentan una situación de partida diferente de la que presentan en la figura A.3 (b). La figura muestra dos procesos de aprendizaje donde la tasa de crecimiento de ventas se incrementa con el tiempo.

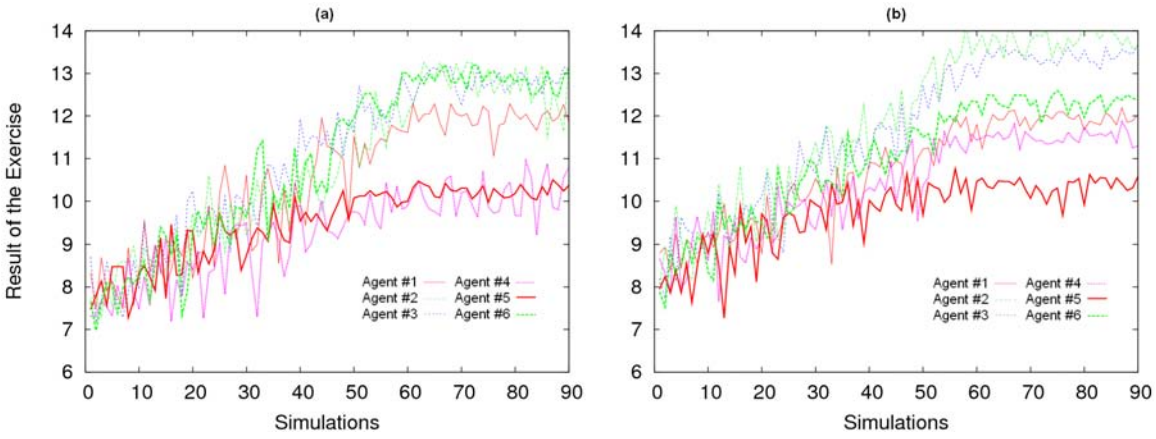


Figura A.3: Aprendizaje múltiple de agentes incrementado la tasa de crecimiento de ventas. Cada gráfica se corresponde con una situación de partida diferente.

En cada caso, como en el experimento anterior, los agentes G-VQQL que participan en el proceso de aprendizaje divergen a políticas de comportamiento diferentes. La figura A.4 muestra dos procesos de aprendizaje cuando la tasa de crecimiento de ventas del mercado se mantiene constante. Por ultimo, la figura A.5 muestra dos procesos de aprendizaje cuando la tasa de crecimiento de ventas decrece con el tiempo.

En algunos casos, los algoritmos se tunean para trabajar en un conjunto reducido de entornos de forma que los resultados obtenidos no son fácilmente generalizable a otros entornos similares. La modificación de la evolución de la tasa de crecimiento de ventas, modifica completamente el comportamiento del mercado de un escenario a otro. No obstante, los experimentos demuestran que, en todos los escenarios de juego propuestos (i.e., con una tasa de crecimiento de ventas creciente, constante y decreciente), los agentes G-VQQL son capaces de adaptarse a la situación y de mejorar sus prestaciones. La evaluación a través de estos escenarios de juego ofrece una evaluación mucho más significativa del algoritmo G-VQQL presentado en esta Tesis.

A.2. Análisis Económico del Comportamiento del Agente G-VQQL

Además, SIMBA ofrece a los humanos la posibilidad de jugar contra agentes software. El principal objetivo del humano involucrado en el siguiente escenario de juego propuesto

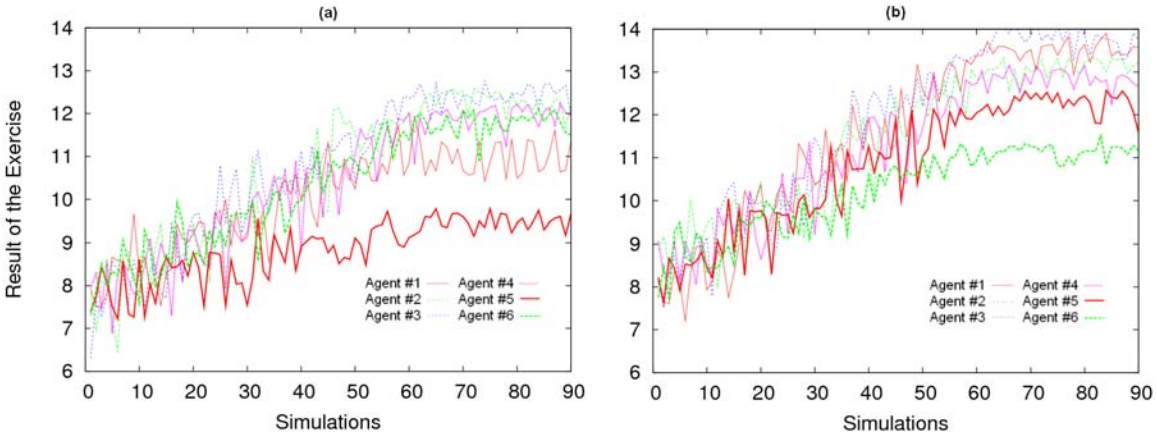


Figura A.4: Aprendizaje múltiple de agentes manteniendo constante la tasa de crecimiento de ventas. Cada gráfica se corresponde con una situación de partida diferente.

es analizar cualitativamente el comportamiento de los agentes G-VQQL desde un punto de vista económico. Para ello, se han ejecutado dos simulaciones de 8 periodos o pasos. Cada una de estas simulación está compuesta de dos tipos de agentes y un humano. El primer agente utiliza el algoritmo G-VQQL descrito en esta Tesis. El segundo agente utiliza una estrategia de aprendizaje completamente diferente; en particular, utiliza una variante de la aproximación k-NN. Este agente k-NN cuenta con una base de datos extremadamente grande con instancias del tipo *(estado,acción,beneficios)* recogidas de una interacción de un agente pre-programado con el simulador. En cada paso, el agente k-NN selecciona las k instancias más cercanas al estado actual y selecciona la mejor acción (i.e., la acción que permitió obtener una mayor cantidad de beneficios) de entre esas k instancias. De esta forma, en cada simulación llevada a cabo, hay un agente G-VQQL, un agente k-NN ($k = 15$) y una compañía gestionada por un humano. El resto de compañías en la simulación son gestionadas por un agente pre-programado. En estas simulaciones, el principal objetivo del humano no es batir a los agentes involucrados en la simulación, sino estudiar el comportamiento del agente G-VQQL desde un punto de vista económico. Este estudio cualitativo demuestra que el agente G-VQQL compite en la misma parcela del mercado (aquella que le asegura los máximos beneficios), utilizando la misma estrategia de negocio todo el tiempo. Para llevar a cabo el análisis del comportamiento, el jugador humano compitió en esta misma parcela del mercado empleando una estrategia similar. No obstante, Porter [Porter, 2008] and Miles [Miles and Snow, 1978] describen diferentes estrategias de negocio para alcanzar y mantener una ventaja competitiva en el mercado. Según estos autores, las mejores estrategias son aquellas que compiten en diferentes parcelas del mercado y utilizan diferentes combinaciones de estrategias de negocio (por ejemplo, bajo coste o diferenciación y especialización). En resumen, el empleo de estrategias más competitivas en diferentes parcelas del mercado, podría mejorar el rendimiento de los agentes y la calidad de las compañías que gestionan.

Por último, independientemente del experimento anterior, queremos resaltar el hecho de que los agentes G-VQQL están siendo utilizados en programas de entrenamiento para estudiantes en económicas. La tabla A.1 presenta algunos resultados preliminares (la media y la desviación estándar) correspondientes a dos simulaciones independientes. En cada una

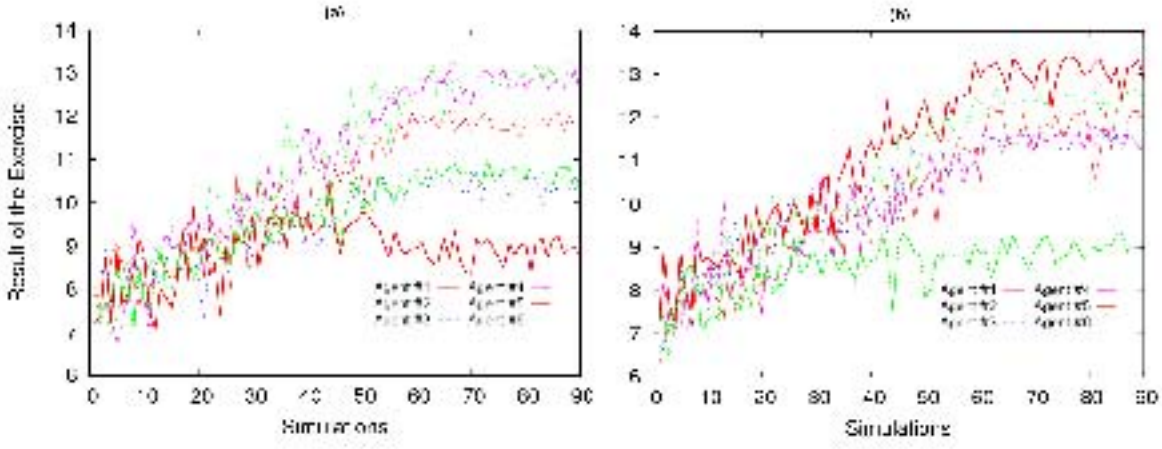


Figura A.5: Aprendizaje múltiple de agentes reduciendo la tasa de crecimiento de ventas. Cada gráfica se corresponde con una situación de partida diferente.

de las simulaciones hay dos agentes pre-programados, dos agentes G-VQQL y dos compañías gestionadas por equipos de estudiantes de cuatro alumnos. En estas simulaciones, el agente G-VQQL obtiene en media unos beneficios mayores (medidos en millones de euros) que una muestra aleatoria de jugadores humanos.

Tabla A.1: Agentes vs. Equipos de Estudiantes (en millones de euros). La tabla muestra la media y la desviación estándar en dos simulaciones independientes.

Simulation	1		2	
Agente	Media	Desviación	Media	Desviación
G-VQQL	**4,94**	0,18	**4,53**	0,34
Pre-programado	1,88	0.79	1,89	0,76
Equipo Humano	3,92	1,21	2,28	0,92

Los resultados son prometedores, no obstante, solo se están presentado resultados de dos simulaciones con dos equipos humanos cada una. Por este motivo, estos resultados no son generalizables al resto de la población. Para poder inferir otro tipo de conclusiones (por ejemplo, que los agentes son lo suficientemente competitivos para jugar contra humanos expertos) se necesitarían obtener resultados a partir de un número mucho mayor de humanos, compitiendo en un número mucho mayor de simulaciones.

A.3. Discusión

El conjunto de experimentos llevados a cabo en este capítulo completan la evaluación del algoritmo G-VQQL presentado en esta Tesis. En todas las situaciones propuestas, las políticas aprendidas mediante el algoritmo G-VQQL son robustas en el sentido de que su rendimiento se mantiene aún cuando se modifican las condiciones del escenario de aprendizaje.

La principal contribución es la demostración de que en un dominio competitivo y multiagente como SIMBA, las características competitivas de los agentes tienen una fuerte in-

fluencia en el resto de agentes. Desde un punto de vista económico, se ha demostrado que los agentes no alcanzan ningún tipo de equilibrio (por ejemplo, dividiendo el mercado en partes iguales y ganando los mismos beneficios). Más bien, dependiendo de los oponentes, se ha visto que las características de las políticas obtenidas pueden variar. Particularmente, cuando varios agentes homogéneos con los mismos parámetros de aprendizaje aprenden al mismo tiempo en un entorno, emerge un comportamiento competitivo, previamente no programado, que hace que cada agente aprenda su propia política de comportamiento con diferente rendimiento. Estos resultados también tienen una utilidad práctica en educación. Como se describe en la sección 4.2.5, SIMBA es un simulador empresarial que es utilizado principalmente en educación.

Un área interesante que puede ser investigada en un futuro consiste en utilizar los agentes obtenidos para la formación de estudiantes en estrategias de negocio. Esta línea de investigación se ve reforzada por el hecho de que es posible seleccionar a agentes G-VQQL con un rendimiento específico como competidores, adaptando de esta forma el grado de competitividad del mercado a las características de un grupo de estudiantes en particular. Otra línea de investigación puede ser el análisis desde un punto de vista económico de la información producida en la interacción entre los agentes con diferentes grados de habilidad en los experimentos propuestos. Esta interacción produce de forma emergente diferentes escenarios económicos que pueden ser analizados con el fin de extraer conocimiento que sea extrapolable al mundo real de los negocios. Dentro de este análisis, se puede estudiar qué decisiones particulares tomaron los agentes en determinadas situaciones con el fin de encontrar nuevas estrategias de negocio. Los expertos en economía consideran que hay diferentes grupos estratégicos dentro de un mercado, cada uno con un rendimiento diferente. Un importante campo de estudio en economía consiste en decidir si la estrategia utilizada por un grupo es mejor que la estrategia de otro [Leask and Parker, 2007]. En nuestros agentes, en cambio, no se identifica una estrategia claramente equiparable con las existentes en la literatura. Por lo tanto, nuestros agentes podrían ser enfrentados a otros agentes que siguen estrategias económicas bien definidas en la literatura (por ejemplo, defensores, prospectores,...), con el fin de identificar nuevas y mejores estrategias, a la vez que se analiza la evolución económica del mercado cuando se utilizan diferentes configuraciones.

Apéndice B

Descripción de los Comportamientos Base

En este capítulo se realiza una descripción de los comportamientos base utilizados en los capítulos 5 y 6. En estos capítulos se ha supuesto la existencia *a priori* de estos comportamientos en los dominios utilizados, aunque en algunos casos, como en los comportamientos base del *Automatic Car Parking Problem* y el *Cart-Pole* presentados en el capítulo 6, han tenido que construirse para comprobar la validez del algoritmo PI-SRL. En el caso del comportamiento base del *Octopus Arm* utilizado en el capítulo 5, el comportamiento base empleado está inspirado en las acciones descritas por Engel et al. [Engel *et al.*, 2006]. En el caso del comportamiento base del *Helicopter*, éste ya viene integrado en el software de la competición de aprendizaje por refuerzo. En esta Tesis se ha utilizado este comportamiento directamente sin ningún tipo de modificación. Por último, el software del simulador empresarial SIMBA trae consigo un agente automático o preprogramado cuya política de comportamiento se ha empleado en los experimentos como comportamiento base. A continuación se describe cómo se ha llevado a cabo el proceso de construcción de estos comportamientos base en algunos casos, o cómo es el comportamiento base que ya traen consigo algunos de los dominios.

B.1. *Automatic Car Parking Problem* y *Cart-Pole*

En ambos casos el comportamiento base empleado en el capítulo 6 se ha construido utilizando el algoritmo G-VQQL descrito en el capítulo 5. En estos dominios, debido al reducido número de características tanto en el espacio de estados como de acciones, un comportamiento aleatorio es suficiente para explorar el espacio de forma que sea posible obtener unas discretizaciones (Tabla 5.1) que permitan posteriormente aprender políticas de comportamiento, que aún siendo subóptimas, permiten completar la tarea de forma segura en ambos casos. En el caso del *Automatic Car Parking Problem*, la configuración del algoritmo G-VQQL empleada es 64×32, es decir, se ha empleado una discretización de 64 centroides para el espacio de estados y una discretización de 32 centroides para el espacio de acciones, con $\alpha = 0{,}125$, y $\gamma = 0{,}9$. La estrategia de exploración utilizada es $\epsilon - greedy$ con $\epsilon = 0$ (completamente avariciosa) desde el inicio del proceso de aprendizaje. Este proceso permite obtener la política del comportamiento base, π_T, con refuerzo total por episodio de 4.75 empleado en los experimentos de la sección 6.5.1. En el caso del *Cart-Pole*, la configuración del algoritmo G-VQQL utilizada es 128×32. Los parámetros de aprendizaje

utilizados son: $\alpha = 0{,}3$, y $\gamma = 1{,}0$. La política de exploración empleada es $\epsilon - greedy$ con $\epsilon = 0$ (completamente avariciosa) desde el inicio del aprendizaje. Este proceso de aprendizaje permite obtener el comportamiento base, π_T, con un refuerzo total por episodio de 9292 empleado en los experimentos de la sección 6.5.2.

B.2. *Octopus Arm*

Dada la complejidad de este dominio, en los procesos de aprendizaje que se pueden encontrar en la literatura, se suelen emplear conjuntos discretos de acciones coherentes. Así por ejemplo, Engel et al. [Engel *et al.*, 2006] utilizan un conjunto de 7 acciones discretas que permiten mover el brazo por el espacio. El comportamiento base empleado en este dominio y utilizado en esta Tesis se basa en algunas de estas acciones, aunque la fuerza que se aplica a cada compartimento es de naturaleza continua en lugar de discreta como en el caso de Engel.

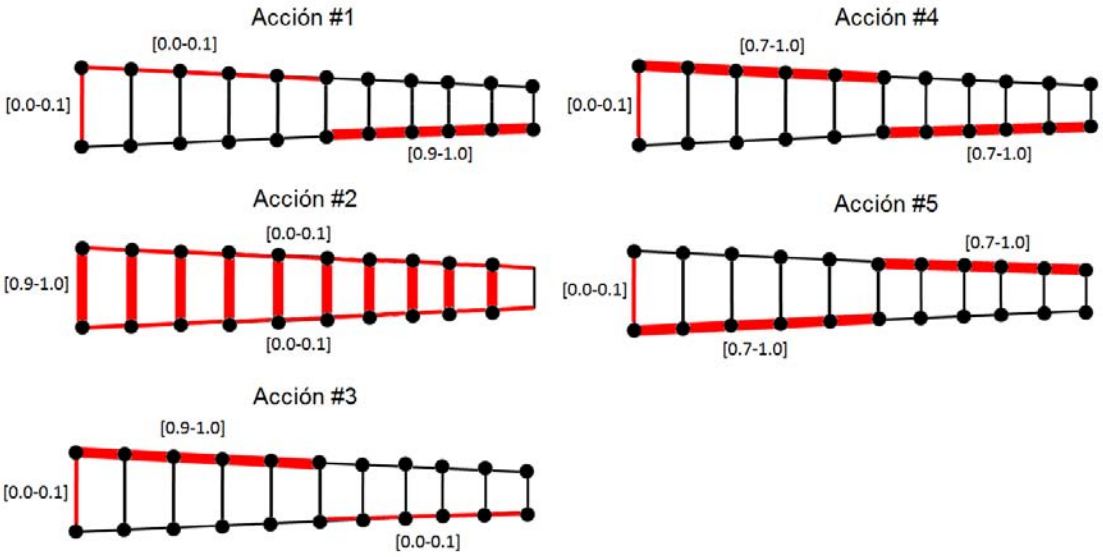


Figura B.1: Descripción de los cinco tipos de acciones del comportamiento base en el dominio del *Octopus Arm*. El grosor de las líneas indica el grado de activación o la fuerza aplicada a cada uno de los compartimentos.

La figura B.1 muestra los cinco tipo de acciones que utiliza el comportamiento base para explorar el entorno en busca de la meta. Cabe destacar que cada tipo de acción representa a un conjunto infinito de posibles acciones debido a que la fuerza que se aplica a cada uno de los compartimentos es de naturaleza continua. Por ejemplo, en el caso de la acción #1 se aplica una fuerza en el rango $[0{,}0 - 0{,}1]$ al compartimento 1 transversal, una fuerza en el rango $[0{,}0 - 0{,}1]$ a los compartimentos del 1 al 5 ventrales y una fuerza en el rango $[0{,}9 - 1{,}0]$ a los compartimentos del 1 al 5 dorsales. La selección de cada una de las acciones y de las fuerzas que se aplican a los compartimentos dependen del estado que determina la posición en la que se encuentra el brazo. Por ejemplo, si se determina que los

compartimentos del brazo tienen forma de "S" invertida, se utilizan las acciones #1, #2 y #4 con mayor frecuencia para deshacer esta situación y dar forma al brazo de "S" normal. Una vez conseguida la forma de "S" normal se trata de volver a la "S" invertida mediante las acciones #2, #3 y #5, y así sucesivamente. De esta forma se consigue que el brazo se mueva en zig-zag, a modo de "serpiente", girando sobre la base, y explorando eficientemente el entorno. La fuerza que se aplica a los compartimentos depende de la amplitud de esta "S". Así, se construye el comportamiento base, π_T, con un refuerzo total por episodio de -1600 utilizado en la sección 5.3.2.

B.3. *Helicopter*

El comportamiento base en el dominio del *Helicopter* viene integrado en el software de la competición de aprendizaje por refuerzo. Se trata de un conjunto de ecuaciones lineales que permiten obtener el valor de cada característica de la acción a partir de los valores de ciertas características de estado (ecuaciones B.1). De esta forma, para el cálculo de la primera característica de una acción, a_0, intervienen las características s_5 y s_2 de estado; para la segunda característica de la acción, a_1, intervienen las características de estado s_0, s_3 y s_{10}; para a_2 se utilizan s_1, s_5 y s_9; por último, para a_3 se emplea s_{11}.

$$\begin{aligned} a_0(t) &= 0{,}0513 \times s_5(t) + 0{,}1348 \times s_2(t) + 0{,}23 \\ a_1(t) &= -0{,}0185 \times s_3(t) + -0{,}0322 \times s_0(t) + 0{,}7904 \times s_{10}(t) \\ a_2(t) &= -0{,}0196 \times s_4(t) + -0{,}0367 \times s_1(t) + -0{,}7475 \times s_9(t) + 0{,}02 \\ a_3(t) &= -0{,}1969 \times s_{11}(t) \end{aligned} \tag{B.1}$$

Cabe destacar que el comportamiento base descrito no emplea las características de estado s_6, s_7 y s_8 por considerarlas irrelevantes para el cálculo de la acción adecuada. Este comportamiento base, π_T, con un refuerzo total por episodio de -78035.93 es el que se emplea en la sección 6.5.3.

B.4. SIMBA

El simulador empresarial SIMBA se trata de un software comercial el cual viene provisto con un agente automático que es capaz de ejecutar acciones coherentes impidiendo que la compañía entre en bancarrota, y ofreciendo cierta resistencia cuando se selecciona para competir con equipos de humanos que gestionan otras compañías. En este agente, el cálculo de las acciones se realiza teniendo en cuenta el valor del IPC del mercado en el episodio actual. El rendimiento de este agente, en cuanto a los beneficios se refiere, puede variar dependiendo del grado de habilidad de los oponentes con los cuales se enfrente. En nuestro caso, este agente se utiliza como el comportamiento base, π_T, utilizado en los capítulos 5 y 6.

Apéndice C

Publicaciones

A continuación se presenta una lista de las publicaciones en congresos o conferencias internacionales que están ligadas al desarrollo de esta Tesis.

Revistas Internacionales Con Factor de Impacto

1. **Autores:** Javier García, Fernando Borrajo y Fernando Fernández
   Título: *Reinforcement Learning for Decision-Making in a Business Simulator*
   Ref. revista/libro: International Journal of Information Technology & Decision Making (JCR del ISI 2010, factor de impacto 3,139 nº 7/108 en Computer Science/Artifical Intelligence; nº 5/126 en Computer Science/Information Systems; nº 7/97 en Computer Science/Interdisciplinary Applications)
   Fecha: Aceptado
   Editorial: World Scientific Publishing Co Pte Ltd

2. **Autores:** Fernando Fernández, Javier García y Manuela Veloso
   Título: *Probabilistic Policy Reuse for Inter-Task Transfer Learning*
   Ref. revista/libro: Robotics and Autonomous Systems (JCR del ISI 2010, factor de impacto 1,313 nº 58/108 en Computer Science/Artificial Intelligence; nº 7/17 en Robotics; nº 24/60 en Automation & Control Systems)
   Fecha: 2010
   Editorial: Elsevier(ISSN: 0921-8890)

3. **Autores:** Fernando Borrajo, Yolanda Bueno, Isidro de Pablo, Begoña Santos, Fernando Fernández, Javier García y Ismael Sagredo
   Título: SIMBA: *A Simulator for Business Education and Research*
   Ref. revista/libro: Decision Support Systems (JCR del ISI 2010, factor de impacto 2.135 nº 25/108 en Computer Science/Artificial Intelligence; nº 22/128 en Computer Science/Information Systems; nº 7/75 en Opertations Research & Management Science)
   Fecha: 2010

Lecture Notes

1. **Autores:** Iván López-Bueno, Javier García y Fernando Fernández
   Título: *Two Steps Reinforcement Learning in Continuous Reinforcement Learning Tasks*

Tipo de Participación: Artículo
Congreso: 10th International Work-Conference on Artificial Neural Networks (IWANN '09)
Publicación: Lecture Notes in Computer Science
Lugar de celebración: Salamanca, Spain
Fecha: 2009

Capítulos de Libros

1. **Autores:** Javier García, Iván López-Bueno, Fernando Fernández y Daniel Borrajo
Título: *A Comparative Study of Discretization Approaches for State Space Generalization in the Keepaway Soccer Task*
Ref. revista/libro: In Progress in Education
Fecha: 2010
Editorial (si libro): Nova Science Publishers

2. **Autores:** Fernando Borrajo, Isidro de Pablo, Javier García y Fernando Fernández
Título: *Business Simulators for Business Education and Research: SIMBA Experience*
Ref. revista/libro: In Business, Technological and Social Dimensions of Computer Games: Multidisciplinary Developments
Editorial (si libro): IGI Global
Fecha: 2011

3. **Autores:** Fernando Borrajo, Yolanda Bueno, Fernando Fernández, Javier García, Isidro de Pablo, Ismael Sagredo y Begoña Santos
Título: *Business Simulation in Business Education*
Ref. revista/libro: In Distance Education
Editorial (si libro): Nova Science Publishers
Fecha: 2010

Contribuciones a Congresos

1. **Autores:** Javier García y Fernando Fernández
Título: *Safe Reinforcement Learning in High-Risk Tasks through Policy Improvement*
Tipo de participación: Artículo
Congreso: IEEE Symposium Series on Computational Intelligence 2011
Publicación: Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning
Lugar de celebración: París, Francia
Fecha: 2011

2. **Autores:** Javier García, Fernando Borrajo y Fernando Fernández
Título: *Learning Virtual Agents for Decision-Making in Business Simulators*
Tipo de participación: Artículo
Congreso: Workshop on Multi-Agent Systems and Simulation (MAS&S)
Publicación: Proceedings of MALLOW (Multi-Agent Logics, Languages, and Organisations Federated Workshops) 2010
Lugar de celebración: Lyon, Francia
Fecha: 2010

3. **Autores:** Raquel Fuentetaja, Silvia de Castro, Javier García, Fernando Fernández, Daniel Borrajo y Fernando Borrajo
   Título: *Un Simulador Empresarial como Herrramienta Práctica para la Asignatura de Aprendizaje Automático*
   Tipo de participación: Artículo
   Congreso: En Jornadas de Enseñanza Universitaria de la Informática (JENUI 2010)
   Publicación: Actas JENUI 2010
   Lugar de celebración: Santiago de Compostela, Spain
   Fecha: 2010

4. **Autores:** Francisco Javier García, Manuela Veloso y Fernando Fernández
   Título: *Reinforcement Learning in the RoboCup-Soccer Keepaway*
   Tipo de participación: Póster
   Congreso: 12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA'07+TTIA)
   Publicación: Actas CAEPIA'07+TTIA
   Lugar de celebración: Salamanca, España
   Fecha: 2007

Bibliografía

[Aamodt and Plaza, 1994] Agnar Aamodt and Enric Plaza. Case-Based Reasoning; Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, 1994.

[Abbeel and Ng, 2005] Pieter Abbeel and Andrew Y. Ng. Exploration and Apprenticeship Learning in Reinforcement Learning. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 1–8, New York, NY, USA, 2005. ACM.

[Abbeel *et al.*, 2006] Pieter Abbeel, Varun Ganapathi, and Andrew Ng. Learning vehicular dynamics, with application to modeling helicopters. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1–8, Cambridge, MA, 2006. MIT Press.

[Abbeel *et al.*, 2007] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. An application of reinforcement learning to aerobatic helicopter flight. In *In Advances in Neural Information Processing Systems 19*, page 2007. MIT Press, 2007.

[Abbeel *et al.*, 2008] Pieter Abbeel, Adam Coates, Timothy Hunter, and Andrew Y. Ng. Autonomous Autorotation of an RC Helicopter. In *ISER*, pages 385–394, 2008.

[Abbeel *et al.*, 2010] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *I. J. Robotic Res.*, 29(13):1608–1639, 2010.

[Abbott, 2008] Robert G. Abbott. Robocup 2007: Robot soccer world cup xi. chapter Behavioral Cloning for Simulator Validation, pages 329–336. Springer-Verlag, Berlin, Heidelberg, 2008.

[Aha and Kibler, 1991] David W. Aha and Dennis Kibler. Instance-based learning algorithms. In *Machine Learning*, pages 37–66, 1991.

[Aha, 1997] David W. Aha, editor. *Lazy learning*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[Albus, 1975] J. S. Albus. A new approach to manipulator control: the cerebellar model articulation controller (cmac. *Journal of Dynamic Systems, Measurement, and Control*, 97:220–227, 1975.

[Andersen *et al.*, 2009] Kresten Toftgaard Andersen, Yifeng Zeng, Dennis Dahl Christensen, and Dung Tran. Experiments with online reinforcement learning in real-time strategy games. *Applied Artificial Intelligence*, 23(9):855–871, 2009.

[Anderson *et al.*, 2000] Charles W. Anderson, Bruce A. Draper, and David A. Peterson. Behavioral cloning of student pilots with modular neural networks. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 25–32. Morgan Kaufmann, 2000.

[Argall *et al.*, 2009] Brenna Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A Survey of Robot Learning from Demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, May 2009.

[Asari, 2005] Vijayan K. Asari. Adaptive technique for image compression based on vector quantization using a self-organizing neural network. *J. Electronic Imaging*, 14(2):023009, 2005.

[Atkeson and Schaal, 1997] Christopher G. Atkeson and Stefan Schaal. Robot learning from demonstration. In Douglas H. Fisher, editor, *ICML*, pages 12–20. Morgan Kaufmann, 1997.

[B. Bethke and J. How and A. Ozdaglar, 2008] B. Bethke and J. How and A. Ozdaglar. Approximate Dynamic Programming Using Support Vector Regression. In *IEEE Conference on Decision and Control (CDC)*, Cancun, Mexico, 2008.

[Bäck and Schwefel, 1993] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, 1:1–23, March 1993.

[Bäck, 1996] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press, 1996.

[Bagnell and Schneider, 2004] J. Andrew Bagnell and Jeff G. Schneider. Covariant policy search. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 1019–1024. Morgan Kaufmann, 2004.

[Bain and Sammut, 1995] M. Bain and C. Sammut. *Machine Intelligence Agents.* Oxford University Press, 1995.

[Baird and Klopf, 1993] L. C. Baird and A. H. Klopf. Reinforcement learning with highdimensional, continuous actions. Technical report, Wright-Patterson Air Force Base Ohio: Wright Laboratory, 1993.

[Baird, 1995] Leemon Baird. Reinforcement learning with function approximation. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.

[Barreto *et al.*, 2010] André M. S. Barreto, Douglas A. Augusto, and Helio J. C. Barbosa. On the characteristics of sequential decision problems and their impact on evolutionary computation and reinforcement learning. In *Proceedings of the 9th international conference on Artificial evolution*, EA'09, pages 194–205, Berlin, Heidelberg, 2010. Springer-Verlag.

[Barto *et al.*, 1983] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):835–846, September/October 1983.

[Bartsch-Spörl *et al.*, 1999] Brigitte Bartsch-Spörl, Mario Lenz, and André Hübner. Case-based reasoning: Survey and future directions. In Frank Puppe, editor, *XPS*, volume 1570 of *Lecture Notes in Computer Science*, pages 67–89. Springer, 1999.

[Bellman, 1958] Richard Bellman. Dynamic programming and stochastic control processes. *Information and Control*, 1(3):228–239, 1958.

[Bertsekas and Tsitsiklis, 1996] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[Bishop, 1996] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1 edition, January 1996.

[Bishop, 2006] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 1st ed. 2006. corr. 2nd printing edition, October 2006.

[Borrajo *et al.*, 2009] Fernando Borrajo, Yolanda Bueno, Isidro de Pablo, Begoña Santos, Fernando Fernández, Javier García, and Ismael Sagredo. Simba: A simulator for business education and research. *Decision Support Systems*, June 2009.

[Borrajo *et al.*, 2010] Fernando Borrajo, Yolanda Bueno, Isidro de Pablo, Begoña Santos, Fernando Fernández, Javier García, and Ismael Sagredo. Business Simulation in Business Education. *Distance Education. Nova Publishers*, 2010.

[Borrajo *et al.*, 2011] Fernando Borrajo, Isidro de Pablo, Javier García, and Fernando Fernández. Business Simulators for Business Education and Research: SIMBA Experience. *Business, Technological and Social Dimensions of Computer Games: Multidisciplinary Developments. IGI Global*, 2011.

[Boutilier *et al.*, 1999] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[Bowling and Veloso, 2002] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[Bryson and Ho, 1969] Arthur E. Bryson and Yu-Chi Ho. *Applied optimal control; optimization, estimation, and control*. Blaisdell Pub. Co. Waltham, Mass.,, 1969.

[Buşoniu *et al.*, 2008] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, March 2008.

[Carysforth, 2006] C. Carysforth. *BTEC First Business*. BTEC first. Heinemann, 2006.

[Chernova and Veloso, 2007] Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *in Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2007.

[Chernova and Veloso, 2008] Sonia Chernova and Manuela Veloso. Multi-thresholded approach to demonstration selection for interactive robot learning. In *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, HRI '08, pages 225–232, New York, NY, USA, 2008. ACM.

[Cichosz, 1995] Pawel Cichosz. Truncating temporal differences: On the efficient implementation of td(lambda) for reinforcement learning. *J. Artif. Intell. Res. (JAIR)*, 2:287–318, 1995.

[Cichosz, 1996] Pawel Cichosz. Truncated temporal differences with function approximation: Successful examples using cmac. In *In Proceedings of the Thirteenth European Symposium on Cybernetics and Systems Research (EMCSR-96*, 1996.

[Coons *et al.*, 2008] Katherine K. Coons, Behnam Robatmili, Matthew E. Taylor, Bertrand A. Maher, Kathryn McKinley, and Doug Burger. Feature selection and policy optimization for distributed instruction placement using reinforcement learning. In *Proceedings of the Seventh International Joint Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 32–42, October 2008.

[Coraluppi and Marcus, 1999] Stefano P. Coraluppi and Steven I. Marcus. Risk-Sensitive and Minimax Control of Discrete-Time, Finite-State Markov Decision Processes. *AUTOMATICA*, 35:301–309, 1999.

[Cournot, 1927] A. Cournot. *Mathematical Principles of the Theory of Wealth*. The Macmillan company, New York, 1927.

[Cover and Hart, 1967] T. Cover and P. Hart. Nearest neighbor pattern classification. 13:21– 27, 1967.

[Dayan, 1992] Peter Dayan. Technical note q-learning. *Machine Learning*, 292(3):279–292, 1992.

[Defourny *et al.*, 2008] Boris Defourny, Damien Ernst, and Louis Wehenkel. Risk-aware decision making and dynamic programming, 2008.

[Driessens and Džeroski, 2004] Kurt Driessens and Sašo Džeroski. Integrating guidance into relational reinforcement learning. *Mach. Learn.*, 57(3):271–304, December 2004.

[Driessens and Ramon, 2003] K. Driessens and J. Ramon. Relational instance based regression for relational rl. In *International Conference of Machine Learning (ICML)*, pages 123–130, 2003.

[Duda and Hart, 1973] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons Inc, 1973.

[Eiben and Smith, 2008] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer, October 2008.

[Engel *et al.*, 2005] Y. Engel, Y. Yekutieli, R. Sagiv-Zohar, R. Aharonov, B. Hochner, and T. Flash. A dynamic model of the octopus arm. i. biomechanics of the octopus reaching movement. *Journal of Neurophysiology*, 94:1443–1458, 2005.

[Engel *et al.*, 2006] Yaakov Engel, Peter Szabo, and Dmitry Volkinshtein. Learning to control an octopus arm with gaussian process temporal difference methods. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 347–354. MIT Press, Cambridge, MA, 2006.

[Fernández and Borrajo, 1999] Fernando Fernández and Daniel Borrajo. Vqql. applying vector quantization to reinforcement learning. In Manuela M. Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup*, volume 1856 of *Lecture Notes in Computer Science*, pages 292–303. Springer, 1999.

[Fernández and Borrajo, 2008a] Fernando Fernández and Daniel Borrajo. Two steps reinforcement learning. *Int. J. Intell. Syst.*, 23(2):213–245, February 2008.

[Fernández and Borrajo, 2008b] Fernando Fernández and Daniel Borrajo. Two steps reinforcement learning. *Int. J. Intell. Syst.*, 23(2):213–245, 2008.

[Fernandez and Isasi, 2008] F. Fernandez and P. Isasi. Local feature weighting in nearest prototype classification. *Neural Networks, IEEE Transactions on*, 19(1):40–53, 2008.

[Fernández *et al.*, 2005] Fernando Fernández, Daniel Borrajo, and Lynne E. Parker. A reinforcement learning algorithm in cooperative multi-robot domains. *J. Intell. Robotics Syst.*, 43(2-4):161–174, August 2005.

[Fernández and Borrajo, 2002] Fernando Fernández and Daniel Borrajo. On determinism handling while learning reduced state space representations. In Frank van Harmelen, editor, *ECAI*, pages 380–384. IOS Press, 2002.

[Fletcher, 1987] R. Fletcher. *Practical methods of optimization; (2nd ed.)*. Wiley-Interscience, New York, NY, USA, 1987.

[Floyd and Esfandiari, 2010] Michael W. Floyd and Babak Esfandiari. Toward a domain-independent case-based reasoning approach for imitation: Three case studies in gaming. In *Workshop on Case-Based Reasoning for Computer Games at the 18th International Conference on Case-Based Reasoning (ICCBR)*, pages 55–64, 2010.

[Floyd *et al.*, 2008] Michael W. Floyd, Babak Esfandiari, and Kevin Lam. A Case-Based Reasoning Approach to Imitating Robocup Players. In *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference*, pages 251–256, 2008.

[Forbes and Andre, 2002] Jeffrey Forbes and David Andre. Representations for learning control policies. In *The University of New South*, pages 7–14, 2002.

[Gabel and Riedmiller, 2005] Thomas Gabel and Martin Riedmiller. Cbr for state value function approximation in reinforcement learning. In *In Proceedings of the 6th International Conference on CaseBased Reasoning (ICCBR 2005*, pages 206–221. Springer, 2005.

[García *et al.*, 2010] Javier García, Fernando Borrajo, and Fernando Fernández. Learning virtual agents for decision-making in business simulators. In Olivier Boissier, Amal El Fallah-Seghrouchni, Salima Hassas, and Nicolas Maudet, editors, *MALLOW*, volume 627 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[García *et al.*, 2011] Javier García, Iván López-Bueno, Fernando Fernández, and Daniel Borrajo. *In Progress of Education*, volume 21, chapter A Comparative Study of Discretization Approaches for State Space Generalization in the Keepaway Soccer Task, pages 161–190. Nova Publishers, 2011. Accepted.

[García and Fernández, 2011] Javier García and Fernando Fernández. Policy improvement through safe reinforcement learning in high-risk tasks. In *Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2011)*, Paris, France, April 2011.

[García *et al.*, 2007] Francisco Javier García, Manuela Veloso, and Fernando Fernández. Reinforcement Learning in the Robocup-Soccer Keepaway. In *Proceedings of the 12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA'07+TTIA)*, 2007.

[García *et al.*, 2012] Javier García, Fernando Borrajo, and Fernando Fernández. Reinforcement learning for decision-making in a business simulator. *International Journal of Information Technology & Decision Making*, 2012.

[Gaskett *et al.*, 1999] Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Q-learning in continuous state and action spaces. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence: Advanced Topics in Artificial Intelligence*, AI '99, pages 417–428, London, UK, 1999. Springer-Verlag.

[Geibel and Wysotzki, 2005] Peter Geibel and Fritz Wysotzki. Risk-sensitive Reinforcement Learning Applied to Control under Constraints. *Journal of Artificial Intelligence Research (JAIR)*, 24:81–108, 2005.

[Geibel, 2001] Peter Geibel. Reinforcement Learning with Bounded Risk. In *Proceedings of the 18th International Conference on Machine Learning*, pages 162–169. Morgan Kaufmann, 2001.

[Gersho and Gray, 1991] Allen Gersho and Robert M. Gray. *Vector quantization and signal compression.* Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[Goto *et al.*, 2002] Ryo Goto, Toshihiro Matsui, and Hiroshi Matsuo. State generalization with support vector machines in reinforcement learning. In *Proceedings of The 4th Asia-Pacific Conference on Simulated Evaluation and Learning*, volume 1, pages 51–55, 2002.

[Grant *et al.*, 2008] Andrew Grant, David Johnstone, and Oh Kang Kwon. Optimal betting strategies for simultaneous games. *Decision Analysis*, 5(1):10–18, 2008.

[Guyon and Elisseeff, 2003] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, March 2003.

[Hall, 1999] M. Hall. *Correlation-based Feature Selection for Machine Learning.* PhD thesis, University of Waikato, 1999.

[Hans, Alexander *et al.*, 2008] Hans, Alexander, Schneegass, Daniel, Anton M. Schäfer, and Udluft, Steffen. Safe Exploration for Reinforcement Learning. In *European Symposium on Artificial Neural Network*, pages 143–148, April 2008.

[Hassibi *et al.*, 1993] Babak Hassibi, David G. Stork, and Stork Crc. Ricoh. Com. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, pages 164–171. Morgan Kaufmann, 1993.

[Haykin, 1988] S.S. Haykin. *Digital communications.* Wiley, 1988.

[Haykin, 1994] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Macmillan, New York, 1994.

[Hecht-Nielsen, 1987] Robert Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. In *Proceedings of IEEE First Annual International Conference on Neural Networks*, volume 3, pages III–11–III–14, 1987.

[Heger, 1994] Matthias Heger. Consideration of Risk in Reinforcement Learning. In *11th International Conference on Machine Learning*, pages 105–111, 1994.

[Hernández-Díaz *et al.*, 2008] Alfredo García Hernández-Díaz, Carlos A. Coello Coello, Fatima Perez, Rafael Caballero, Julián Molina Luque, and Luis V. Santana-Quintero. Seeding the initial population of a multi-objective evolutionary algorithm using gradient-based information. In *IEEE Congress on Evolutionary Computation*, pages 1617–1624. IEEE, 2008.

[Hester *et al.*, 2011] Todd Hester, Michael Quinlan, and Peter Stone. A real-time model-based reinforcement learning architecture for robot control. Technical Report arXiv e-Prints 1105.1749, arXiv, May 2011.

[Holland, 1962] John H. Holland. Outline for a logical theory of adaptive systems. *J. ACM*, 9:297–314, July 1962.

[Holland, 1992] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[Howard, 1960] R.A. Howard. *Dynamic programming and Markov processes.* Technology Press of Massachusetts Institute of Technology, 1960.

[Hu and Wellman, 2003] Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *JOURNAL OF MACHINE LEARNING RESEARCH*, 4:1039–1069, 2003.

[Hu *et al.*, 2001] Huosheng Hu, Kostas Kostiadis, Matthew Hunter, and Nikolaos Kalyviotis. Essex wizards 2001 team description. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup*, volume 2377 of *Lecture Notes in Computer Science*, pages 511–514. Springer, 2001.

[Hu *et al.*, 2006] Guanghua Hu, Yuqin Qiu, and Liming Xiang. Kernel-based reinforcement learning. In De-Shuang Huang, Kang Li, and George W. Irwin, editors, *ICIC (1)*, volume 4113 of *Lecture Notes in Computer Science*, pages 757–766. Springer, 2006.

[Jaakkola *et al.*, 1995] Tommi Jaakkola, Satinder P. Singh, and Michael I. Jordan. Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 345–352. MIT Press, 1995.

[Jacobs *et al.*, 1991] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Comput.*, 3:79–87, March 1991.

[Jiang, 2004] Albert Xin Jiang. Mulitagent reinforcement learning in stochastic games with continuous action spaces, 2004.

[Justin Boyan, 1995] Andrew Moore Justin Boyan. Generalization in reinforcement learning: Safely approximating the value function. In T.K. Lee G. Tesauro, D.S. Touretzky, editor, *Neural Information Processing Systems 7*, pages 369–376, Cambridge, MA, 1995. The MIT Press.

[Kaelbling *et al.*, 1996] Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[Kakade, 2003] Sham Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, University College London, 2003.

[Kekre *et al.*, 2012] H. B. Kekre, Tanuja K. Sarode, and Jagruti K. Save. New clustering algorithm for vector quantization using walsh sequence. *International Journal of Computer Applications*, 39(1):4–9, February 2012. Published by Foundation of Computer Science, New York, USA.

[Konen and Bartz-Beielstein, 2009] Wolfgang Konen and Thomas Bartz-Beielstein. Reinforcement learning for games: failures and successes. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 2641–2648, New York, NY, USA, 2009. ACM.

[Koppejan and Whiteson, 2009] Rogier Koppejan and Shimon Whiteson. Neuroevolutionary reinforcement learning for generalized helicopter control. In *GECCO 2009: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 145–152, July 2009.

[Koppejan and Whiteson, 2011] Rogier Koppejan and Shimon Whiteson. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evolutionary Intelligence*, 4:219–241, 2011.

[Kroon and Whiteson, 2009] Mark Kroon and Shimon Whiteson. Automatic feature selection for model-based reinforcement learning in factored MDPs. In *ICMLA 2009: Proceedings of the Eighth International Conference on Machine Learning and Applications*, pages 324–330, December 2009.

[Kumar and Varaiya, 1986] P. R. Kumar and Pravin Varaiya. *Stochastic systems: estimation, identification and adaptive control.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1986.

[Latzke *et al.*, 2006] Tobias Latzke, Sven Behnke, and Maren Bennewitz. Imitative reinforcement learning for soccer playing robots. In *Proceedings of The 10th RoboCup International Symposium, Bremen*, 2006.

[Lauer and Riedmiller, 2000] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, 2000.

[Lazaric *et al.*, 2007] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS*. MIT Press, 2007.

[Leask and Parker, 2007] Graham Leask and David Parker. Strategic groups, competitive groups and performance within the uk pharmaceutical industry: improving our understanding of competitive process. *Strategic Management Journal*, 28(7):723–745, 2007.

[Lee and Lee, 2008] Joon-Yong Lee and Ju-Jang Lee. *Multiple Designs of Fuzzy Controllers for Car Parking Using Evolutionary Algorithm*, pages 1–6. Number May. 2008.

[Lin, 1991] Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the ninth National conference on Artificial intelligence - Volume 2*, AAAI'91, pages 781–786. AAAI Press, 1991.

[Linde *et al.*, 1980] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *Communications, IEEE Transactions on*, 28(1):84–95, 1980.

[Littman, 2001] Michael L. Littman. Value-function reinforcement learning in markov games. *Journal of Cognitive Systems Research*, 2:55–66, 2001.

[Liu *et al.*, 2003] Yaxin Liu, Richard Goodwin, and Sven Koenig. Risk-averse auction agents. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, pages 353–360, New York, NY, USA, 2003. ACM.

[Lloyd, 1957] S.P. Lloyd. Least squares quantization in pcm. Technical report, Bell Laboratories Technical Note, 1957.

[López-Bueno *et al.*, 2009] Iván López-Bueno, Javier García, and Fernando Fernández. Two steps reinforcement learning in continuous reinforcement learning task. In *10th International Work-Conference on Artificial Neural Networks (IWANN 2009). Lecture Notes in Computer Science 5517*, pages 577–584, 2009.

[Lovejoy, 1991] W S Lovejoy. A survey of algorithmic methods for partially observed markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

[Luenberger, 1998] David G. Luenberger. Investment science. *Oxford University Press*, 1998.

[Macleod *et al.*, 1987] J. E. S. Macleod, A. Luk, and D. M. Titterington. A Re-Examination of the Distance-Weighted k-Nearest neighbor classification rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17(4):689–696, 1987.

[Mannor, 2004] Shie Mannor. Reinforcement learning for average reward zero-sum games. In John Shawe-Taylor and Yoram Singer, editors, *COLT*, volume 3120 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 2004.

[Mansley *et al.*, 2011] Christopher R. Mansley, Ari Weinstein, and Michael L. Littman. Sample-based planning for continuous action markov decision processes. In Fahiem Bacchus, Carmel Domshlak, Stefan Edelkamp, and Malte Helmert, editors, *ICAPS*. AAAI, 2011.

[Martín and de Lope Asiaín, 2009] Jose Martín and Javier de Lope Asiaín. Learning autonomous helicopter flight with evolutionary reinforcement learning. In *EUROCAST*, pages 75–82, 2009.

[Martinez-Gil *et al.*, 2011] Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. Multi-agent reinforcement learning for simulating pedestrian navigation. In Peter Vrancx, Matthew Knudson, and Marek Grzes, editors, *ALA*, volume 7113 of *Lecture Notes in Computer Science*, pages 54–69. Springer, 2011.

[Mataric, 1997] Maja J. Mataric. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4:73–83, 1997.

[Mihatsch and Neuneier, 2002] Oliver Mihatsch and Ralph Neuneier. Risk-Sensitive reinforcement learning. *Machine Learning*, 49(2-3):267–290, 2002.

[Miles and Snow, 1978] Raymond E. Miles and Charles C. Snow. *Organizational strategy, structure, and process*. McGraw-Hill, New York, 1978.

[Miller, 2003] J.D. Miller. *Game Theory at Work: How to Use Game Theory to Outthink and Outmaneuver Your Competition*. McGraw-Hill, 2003.

[Millán *et al.*, 2002] J. Millán, D. Posenato, and E. Dedieu. Continuous-action q-learning. *Machine Learning*, 49(2):247–265, 2002.

[Millán, 1997] J. Millán. Incremental acquisition of local networks for the control of autonomous robots. In *Proceedings of the 7th International Conference on Artificial Neural Networks*, ICANN '97, pages 739–744, London, UK, UK, 1997. Springer-Verlag.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[Moldovan and Abbeel, 2012] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. *CoRR*, abs/1205.4810, 2012.

[Molineaux *et al.*, 2008] Matthew Molineaux, David W. Aha, and Philip Moore. Learning continuous action models in a real-time strategy environment. In David Wilson and H. Chad Lane, editors, *FLAIRS Conference*, pages 257–262. AAAI Press, 2008.

[Monahan, 1982] George E Monahan. A survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[Moore and Atkeson, 1993] Andrew Moore and Chris Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[Moore and Atkeson, 1995] Andrew Moore and Chris Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21, 1995.

[Moriarty and Miikkulainen, 1997] David E. Moriarty and Risto Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4):373–399, 1997.

[Moriarty *et al.*, 1999] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999.

[Morimoto and Atkeson, 2007] Jun Morimoto and Christopher G. Atkeson. Learning biped locomotion: Application of poincare-map-based reinforcement learning. In *Robotics and Automation Magazine*, volume 14, pages 41–51, 2007.

[Morimoto *et al.*, 2004] Jun Morimoto, Gordon Cheng, Christopher G. Atkeson, and Garth Zeglin. A simple reinforcement learning algorithm for biped walking. In *International Conference on Robotics and Automation*, volume 3, pages 3030–3035, 2004.

[Morimura *et al.*, 2010a] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka. Parametric return density estimation for reinforcement learning. In P. Grünwald and P. Spirtes, editors, *The 26th Conference on Uncertainty in Artificial Intelligence (UAI2010)*, pages 368–375, Catalina Island, California, USA, Jul. 8–11 2010.

[Morimura *et al.*, 2010b] Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. Nonparametric return distribution approximation for reinforcement learning. In Johannes Fürnkranz and Thorsten Joachims, editors, *ICML*, pages 799–806. Omnipress, 2010.

[Munos and Moore, 2002] Remi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49, Numbers 2/3:291–323, November/December 2002.

[Munos, 2000] Rémi Munos. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Mach. Learn.*, 40:265–299, September 2000.

[Narendra and Thathachar, 1974] K S Narendra and M A L Thathachar. Learning automata - a survey. *Ieee Transactions On Systems Man And Cybernetics*, SMC-4(4):323–334, 1974.

[Narendra and Thathachar, 1989] Kumpati S. Narendra and Mandayam A. L. Thathachar. *Learning automata: an introduction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[Navarro *et al.*, 2011] Nicolás Navarro, Cornelius Weber, and Stefan Wermter. Real-world reinforcement learning for autonomous humanoid robot charging in a home environment. In *Proceedings of the 12th Annual conference on Towards autonomous robotic systems*, TAROS'11, pages 231–240, Berlin, Heidelberg, 2011. Springer-Verlag.

[Ng *et al.*, 2004] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Inverted autonomous helicopter flight via reinforcement learning. In *In International Symposium on Experimental Robotics*. MIT Press, 2004.

[Nolfi and Floreano, 2004] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Intelligent Robotics and Autonomous Agents. MIT Press, 2004.

[Nouri and Littman, 2008] Ali Nouri and Michael L. Littman. Multi-resolution exploration in continuous spaces. In *NIPS*, pages 1209–1216, 2008.

[Parr and Russell, 1995] Ronald Parr and Stuart Russell. Approximating optimal policies for partially observable stochastic domains. In *In Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1088–1094. Morgan Kaufmann, 1995.

[Pazis and Lagoudakis, 2009] Jason Pazis and Michail G. Lagoudakis. Binary action search for learning Continuous-Action control policies. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 793–800, New York, NY, USA, 2009. ACM.

[Peng and Williams, 1993] J. Peng and R. J. Williams. Efficient learning and planning with the dyna framework. *Adaptive Behavior*, 1:437–454, 1993.

[Perkins and Barto, 2002] Theodore Perkins Perkins and Andrew G. Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3:803–832, 2002.

[Peshkin, 2002] Leonid Peshkin. *Reinforcement Learning by Policy Search*. PhD thesis, Brown University, 2002.

[Peters *et al.*, 2010] Jan Peters, Russ Tedrake, Nicholas Roy, and Jun Morimoto. Robot learning. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 865–869. Springer, 2010.

[Poli and Cagnoni, 1997] Riccardo Poli and Stefano Cagnoni. Genetic programming with user-driven selection: Experiments on the evolution of algorithms for image enhancement. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 269–277. Morgan Kaufmann, 1997.

[Porter, 2008] Michael E. Porter. *On Competition, Updated and Expanded Edition*. Harvard Business School Press, Boston, MA, updated and expanded edition, September 2008.

[Powell, 1998] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, 1998.

[Powers and Shoham, 2005] Rob Powers and Yoav Shoham. New criteria and a new algorithm for learning in Multi-Agent systems. In Lawrence K. Saul and Yair W. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1089–1096, Cambridge, MA, 2005. MIT Press.

[Prokhorov and Wunsch, 1997] Danil Prokhorov and Don Wunsch. Adaptive critic designs. *IEEE Transactions on Neural Networks*, 8:997–1007, 1997.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, April 1994.

[Rasmusen, 2007] E. Rasmusen. *Games and information: an introduction to game theory*. Blackwell Pub., 2007.

[Rasmussen and Williams, 2006] Carl E. Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[Röttger and Liehr, 2009] Michael C. Röttger and Andreas W. Liehr. Control task for reinforcement learning with known optimal solution for discrete and continuous actions. *JILSA*, 1(1):28–41, 2009.

[Rudolph, 1996] Stephan Rudolph. On a genetic algorithm for the selection of optimally generalizing neural network topologies. In *Proceedings of the 2nd International Conference on Adaptative Computing in Engineering Design and Control '96, I.C. Parmee (Ed.), University of Plymouth, March 26th-28th, Plymouth, United Kingdom*, pages 79–86, 1996.

[Rumelhart *et al.*, 1988] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation.* MIT Press, Cambridge, MA, USA, 1988.

[Russell and Norvig, 2010] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.).* Pearson Education, 2010.

[Salkham *et al.*, 2008] Asad Salkham, Raymond Cunningham, Anurag Garg, and Vinny Cahill. A collaborative reinforcement learning approach to urban traffic control optimization. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT '08. IEEE/WIC/ACM International Conference on*, volume 2, pages 560–566, 2008.

[Santamaría *et al.*, 1998] Juan C. Santamaría, Richard S. Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6:163–218, 1998.

[Sathappan and Pannirselvam, 2011] S. Sathappan and Dr. S. Pannirselvam. An enhanced vector quantization method for image compression with modified fuzzy possibilistic c-means using repulsion. *International Journal of Computer Applications*, 21(5):27–34, May 2011. Published by Foundation of Computer Science.

[Sato *et al.*, 2002] M Sato, H Kimura, and S Kobayashi. Td algorithm for the variance of return and mean-variance reinforcement learning. *Transactions of the Japanese Society for Artificial Intelligence*, 16:353–362, 2002.

[Schneider, 1996] Jeff G. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *in Neural Information Processing Systems 9*, pages 1047–1053. The MIT Press, 1996.

[Schölkopf and Smola, 2002] B. Schölkopf and A.J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* Adaptive computation and machine learning. MIT Press, 2002.

[Sen and Weiß, 1999] Sandip Sen and Gerhard Weiß. Learning in multiagent systems. In Gerhard Weiß, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 259–298. MIT Press, Cambridge, MA, USA, 1999.

[Sharma *et al.*, 2007] Manu Sharma, Michael Holmes, Juan Santamaria, Arya Irani, Charles Isbell, and Ashwin Ram. Transfer learning in real-time strategy games using hybrid cbr / rl. *Learning*, 94(8):1041–1046, 2007.

[Shoham *et al.*, 2003] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-Agent reinforcement learning: a critical survey. Technical report, Stanford University, 2003.

[Siebel and Sommer, 2007] Nils T. Siebel and Gerald Sommer. Evolutionary reinforcement learning of artificial neural networks. *Int. J. Hybrid Intell. Syst.*, 4:171–183, August 2007.

[Singh and Jordan, 1994] Jaakkola T. Singh, S. P. and M. I. Jordan. Learning without state-estimation in partially observable markovian decision problems. *Neural Comput.*, 6:1185–1201, November 1994.

[Smallwood and Sondik, 1973] R D Smallwood and E J Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

[Smart and Kaelbling, 2000] William D. Smart and Leslie Pack Kaelbling. Practical reinforcement learning in continuous spaces. pages 903–910. Morgan Kaufmann, 2000.

[Smart and Kaelbling, 2002] W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings- IEEE International Conference on Robotics and Automation*, volume 4, pages 3404–3410, 2002.

[Smola and Schölkopf, 2004] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, August 2004.

[Sondik., 1971] E. J. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes*. PhD thesis, Standford, California, 1971.

[Stanley and Miikkulainen, 2002a] Kenneth O. Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, page 9, San Francisco, 2002. Morgan Kaufmann.

[Stanley and Miikkulainen, 2002b] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[Stone *et al.*, 2005] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[Strehl and Littman, 2008] Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *J. Comput. Syst. Sci.*, 74(8):1309–1331, 2008.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[Sutton, 1990] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *In Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.

[Tanaka and Sano, 1995] Kazuo Tanaka and Manabu Sano. Trajectory stabilization of a model car via fuzzy control. *Fuzzy Sets Syst.*, 70:155–170, March 1995.

[Tang *et al.*, 2010] J. Tang, A. Singh, N. Goehausen, and P. Abbeel. Parameterized maneuver learning for autonomous helicopter flight. In *International Conference on Robotics and Automation (ICRA)*, 2010.

[Tanner and White, 2009] Brian Tanner and Adam White. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research*, 10:2133–2136, September 2009.

[Taylor *et al.*, 2006] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 1321–1328, New York, NY, USA, 2006. ACM.

[Taylor *et al.*, 2007] Matthew E. Taylor, Shimon Whiteson, and Peter Stone. Temporal difference and policy search methods for reinforcement learning: An empirical comparison. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence*, pages 1675–1678, July 2007. Nectar Track.

[Taylor *et al.*, 2011] Matthew E. Taylor, Brian Kulis, and Fei Sha. Metric learning for reinforcement learning agents. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2011.

[Tesauro, 1992] Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3-4):257–277, May 1992.

[Tesauro, 1994] Gerald Tesauro. TD-gammon, a Self-Teaching backgammon program, achieves Master-Level play. *Neural Computation*, 6(2):215–219, March 1994.

[Thomaz and Breazeal, 2006] Andrea L. Thomaz and Cynthia Breazeal. Reinforcement learning with human teachers: evidence of feedback and guidance with implications for learning performance. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, pages 1000–1005. AAAI Press, 2006.

[Turk and Pentland, 1991] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591. IEEE Comput. Sco. Press, 1991.

[van Hasselt and Wiering, 2007] H. van Hasselt and M. A. Wiering. Reinforcement Learning in Continuous Action Spaces. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pages 272–279, 2007.

[Vlassis, 2003] Nikos Vlassis. A concise introduction to multiagent systems and distributed AI. Informatics Institute, University of Amsterdam, September 2003.

[Vollbrecht, 2003] H. Vollbrecht. *Hierarchical reinforcement learning in continuous state spaces.* PhD thesis, University Ulm, 2003.

[Volná, 2010] Eva Volná. Neuroevolutionary optimization. *CoRR*, abs/1004.3557, 2010.

[Watkins, 1989] C. Watkins. *Learning from Delayed Rewards.* PhD thesis, University of Cambridge,England, 1989.

[Weigend *et al.*, 1990] A.S. Weigend, D.E. Rumelhart, and B.A. Huberman. Backpropagation, weight-elimination and time series prediction. In *Proceedings of the 1990 Summer School on Connectionist Models*, 1990.

[Weiss, 1999] Gerhard Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[Werbos, 1974] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[Werbos, 1994] P.J. Werbos. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Adaptive and Learning Systems for Signal Processing, Communications, and Control. John Wiley & Sons, 1994.

[Whitehead, 1991] Steven D. Whitehead. Complexity and cooperation in q-learning. In *Proceedings Eighth International Workshop on Machine Learning*, pages 363–367, 1991.

[Whiteson and Stone, 2006] Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006.

[Whiteson *et al.*, 2009] Shimon Whiteson, Brian Tanner, Matthew E. Taylor, and Peter Stone. Generalized domains for empirical evaluations in reinforcement learning. In *ICML 2009: Proceedings of the Twenty-Sixth International Conference on Machine Learning: Workshop on Evaluation Methods for Machine Learning*, June 2009.

[Whiteson, 2010] S. Whiteson. *Adaptive Representations for Reinforcement Learning*. Studies in Computational Intelligence. Springer, 2010.

[Witten and Frank, 2005] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA, 2. edition, 2005.

[Wyatt, 1997] J. Wyatt. *Exploration and Inference in Learning from Reinforcement*. University of Edinburgh, 1997.

[Xu *et al.*, 2005] Xin Xu, Tao Xie, Dewen Hu, Xicheng Lu, Xin Xu, Tao Xie, Dewen Hu, and Xicheng Lu. Kernel least-squares temporal difference learning kernel least-squares temporal difference learning. *International Journal of Information Technology*, 11(9):54–63, 2005.

[Xu *et al.*, 2007] Xin Xu, Dewen Hu, and Xicheng Lu. Kernel-Based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, July 2007.

[Xu *et al.*, 2011] Xin Xu, Chunming Liu, and Dewen Hu. Continuous-action reinforcement learning with fast policy search and adaptive basis function selection. *Soft Comput.*, 15(6):1055–1070, June 2011.

[Yaeger and Sporns, 2008] L. S. Yaeger and O. Sporns. Evolution of neural structure and complexity in a computational ecology. *Artificial Life, 2008. ALIFE'08. IEEE Symposium on*, 2008.

[Yao, 1999] Xin Yao. Evolving artificial neural networks. *PIEEE: Proceedings of the IEEE*, 87:1423–1447, September 1999.

[Zanuy, 2000] M.F. Zanuy. *Tratamiento digital de voz e imagen y aplicación a la multimedia*. ACCESO RÁPIDO. Marcombo, 2000.