Capítulo 5

El Algoritmo ENNC

El algoritmo ENNC (*Evolutionary Design of Nearest Neighbour Classifiers*) surge por la necesidad de supervisar la discretización del espacio de estados con la función de valor que se está calculando en el proceso de aprendizaje por refuerzo. Es un aprendizaje que desea obtener un conjunto de prototipos (o regiones), pero que está supervisado por la función de valor. Por tanto, es un método de aprendizaje supervisado, que puede considerarse como un aproximador de las funciones de valor. No obstante, este clasificador busca el cumplimiento de ciertas propiedades más, que se consideran en el siguiente apartado. En la sección 5.2 se hace una introducción a los métodos de clasificación del prototipo más cercano, mientras que la sección 5.3 describe el algoritmo ENNC de forma detallada. La sección 5.4 muestra algunos experimentos realizados para verificar la validez del algoritmo, y por último, en la sección 5.5 se detallan las principales conclusiones sobre este algoritmo. La integración de este clasificador en un modelo de aprendizaje por refuerzo se detalla posteriormente en el capítulo 6.

5.1. Introducción

La motivación de este algoritmo es encontrar un método que permita obtener una discretización del espacio de estados que minimice la introducción de indeterminismo producida por la pérdida de la propiedad de Markov al realizar la discretización. En principio, este algoritmo se plantea para dominios deterministas, para los que el número de valores distintos de la función de valor puede ser finito, tal y como se mostrará en el capítulo 6. Por tanto, esos valores pueden entenderse como un conjunto finito de clases nominales. No obstante, se mostrará que el método es también válido para dominios inherentemente estocásticos.

Otra de las características fundamentales que se plantean es que el número de parámetros que requiere para obtener buenas soluciones sea lo más reducido posible. En lo que a la tarea de aprendizaje por refuerzo se refiere, esto queda reflejado principalmente en que el número de prototipos que debe utilizar, lo que se traduce en el tamaño de la discretización del espacio de estados, sea calculado automáticamente. De igual modo, el conjunto de prototipos inicial, o discretización inicial, debería ser obtenido automáticamente, para que el algoritmo no se vea influido por esa inicialización.

5.2. Clasificación del Vecino más Cercano

Tal y como se introdujo en el apartado 2.1, los clasificadores basados en la regla del vecino más cercano, o clasificadores del prototipo más cercano, son un tipo de clasificadores que asignan, para cada nuevo individuo no etiquetado, v, la etiqueta del prototipo más cercano, r_i, perteneciente a un conjunto $C = \{r_1, \ldots, r_N\}$ de prototipos previamente etiquetados [Duda and Hart, 1973]. El diseño de este tipo de clasificadores es complicado, y tiene como principales problemas el definir el número de prototipos a utilizar, es decir, N, así como la situación inicial de este conjunto de prototipos.

Existe una gran discusión sobre cuál es la técnica más adecuada para resolver este problema [Kuncheva and Bezdek, 1998]. Algunas técnicas de agrupación (*clustering*) [Patanè and Russo, 2002, Bermejo and Cabestany, 2000, Pal *et al.*, 1993] están basadas en dos pasos fundamentales. Por un lado está el encontrar un conjunto adecuado de grupos o *clusters* para definir un conjunto reducido de prototipos. El segundo paso sería etiquetar estos prototipos basándose en ejemplos conocidos, previamente etiquetados, y en la regla del vecino más cercano.

Existen muchas otras aproximaciones que se basan en la utilización de diferentes arquitecturas de redes neuronales para realizar la clasificación, como el algoritmo LVQ [Kohonen, 1984], o las redes de base radial [Fritzke, 1994]. Para encontrar un número aproximado de prototipos a utilizar, se suelen seguir, a su vez, dos aproximaciones fundamentales. Por un lado, se pueden utilizar heurísticas que introducen o eliminan prototipos (o neuronas) según se está diseñando el clasificador, basándose en heurísticas tales como el error de cuantificación medio [Patanè and Russo, 2001], o el éxito en la clasificación [Pérez and Vidal, 1993]. Por otro lado, otras aproximaciones intentan definir primero el número óptimo de prototipos, para luego aprender el clasificador con este valor. Los algoritmos genéticos [Holland, 1975] son una aproximación ampliamente utilizada para definir el conjunto inicial de prototipos, en conjunto con otra técnica que realiza la optimiza-

ción local [Merelo *et al.*, 1998]. En [Zhao and Higuchi, 1996], se presenta una aproximación evolutiva basada en la regla R^4 (*recognition, remembrance, reduction and review*) para evolucionar perceptrones multicapa basados en el vecino más cercano.

La aproximación denominada *Evolutionary Nearest Neighbour Classifier* (ENNC), permite diseñar el clasificador sin introducir condiciones iniciales. La principal diferencia con otros métodos es que ésta es una técnica completa en sí misma. Anteriores trabajos se basan en el uso de una técnica existente para diseño de clasificadores, a la que se le añaden los elementos necesarios para resolver los problemas de inicialización. Es decir, se pueden separar perfectamente las partes orientadas a localizar correctamente los prototipos, de las partes orientadas a obtener el número adecuado de ellos. Sin embargo, en este trabajo, tanto la optimización de la localización de los prototipos, como la obtención del número adecuado de estos, están totalmente integradas en un conjunto de operadores, sin que puedan utilizarse unos sin los otros.

5.3. Descripción del algoritmo ENNC

En esta sección se pretende realizar una descripción detallada del algoritmo ENNC. Para ello, a continuación se enumeran los principales conceptos a tener en cuenta, pasando posteriormente a describir la representación y el proceso de aprendizaje.

5.3.1. Conceptos

A continuación se describen los principales conceptos manejados en *ENNC*. Muchos de ellos fueron introducidos en la sección 4.1, ya que un clasificador del prototipo más cercano puede considerarse como un cuantificador de Voronoi cuyos prototipos han sido etiquetados con una clase.

Prototipo, r_i. Define cada prototipo del clasificador. El prototipo está compuesto por su situación en el espacio y la clase a la que pertenece.

Clasificador, C. Un conjunto de N prototipos, $C = \{r_1, \ldots, r_N\}$.

Tamaño del Clasificador, N. Es el número de prototipos del clasificador.

Región, r_i. El entorno es dividido en un conjunto de N regiones de Voronoi, definidas por la regla del vecino más cercano y las posiciones de los prototipos, tal y como fue definido en 4.1. En este sentido, hay una relación directa entre los prototipos y las regiones, por lo que en adelante, se hablará de regiones y prototipos indistintamente.

Patrón, v_r. Es cada uno de los ejemplos que serán utilizados para entrenar al sistema. Todos ellos componen un conjunto $V = \{v_1, \ldots, v_M\}$, y, al igual que los prototipos, están compuestos por su localización y su clase.

Número de patrones, M. El tamaño del conjunto V.

Clase, s_j. Tanto los prototipos, como los patrones, pertenecen a una clase del conjunto $S = \{s_1, \ldots, s_L\}$. El objetivo de todo prototipo r_i, de la clase, s_j, es contener en su región tantos patrones de la clase s_j como sea posible, así como no contener a ningún patrón de cualquier otra clase $s_k \neq s_j$.

Calidad de un prototipo, $calidad_{r_i}$. Es una medida de lo bueno que es el prototipo, tomando en cuenta el número de patrones que hay en su región, y si esos patrones pertenecen a su misma clase o no. Este concepto será descrito formalmente más adelante.

5.3.2. Representación de ENNC

El sistema puede ser representado por una matriz bidimensional, donde cada fila está asociada a una región $r_i \in C$, y cada columna está asociada a una clase $s_j \in S$. Cada posición (i, j) de la matriz es una estructura que contiene características o información acerca del conjunto de patrones de entrenamiento que pertenecen a la región r_i y a la clase s_j. Estos conjuntos son denominados Conjuntos Región-Clase, y se denotan por V_{ij}. Además, se almacena información acerca de los patrones que pertenecen a una determinada clase (conjunto VS_j), y de los que pertenecen a una determinada región, (conjunto VR_i). Por último, también se mantiene información sobre el conjunto total de patrones, V. Esto significa que el sistema almacena características acerca de las regiones existentes, las clases, los patrones que están situados en cada región, los patrones que pertenecen a cada clase, y los patrones de cada clase que están localizados en cada región. No obstante, hay que destacar que la estructura no almacena todos los conjuntos región, clase y región-clase, sino sólo algunas características sobre ellos, como son su tamaño, centroide, etc. y que serán definidos posteriormente. Toda esta información es la base del funcionamiento del algoritmo. Para todos estos conjuntos, también se define una función de pertenencia a ellos, tal y como se mostrará posteriormente. La Figura 5.1 muestra la estructura utilizada para mantener toda esta información. Cada uno de los nodos almacena las características acerca de los conjuntos que representa.

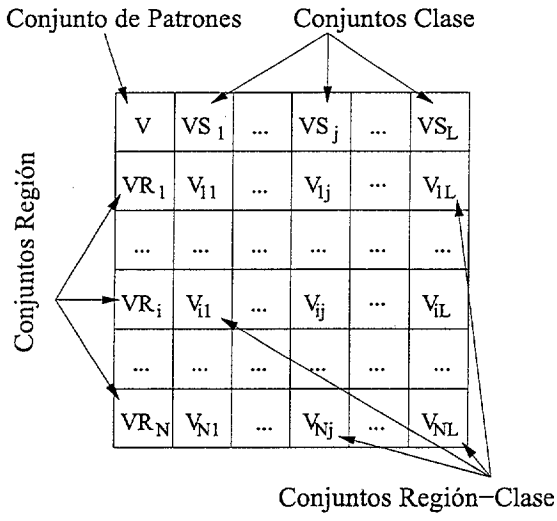


Figura 5.1: Representación de ENNC.

Conjuntos Clase

El conjunto VS_j se define como el conjunto de patrones que pertenecen a la clase s_j. La función de pertenencia a este conjunto viene definida por la igualdad entre la clase de los patrones y la clase asociada al conjunto. Si $clase_v$ es la clase asociada al patrón v, la función de pertenencia al conjunto VS_j viene dada por la ecuación 5.1.

$$\forall v \in V, \forall s_j \in S, v \in VS_j \text{ sii } clase_v = s_j \tag{5.1}$$

Sobre este conjunto se pueden calcular ciertas características que serán útiles para el algoritmo, tales como:

- $\|VS_j\|$, o tamaño del conjunto VS_j
- $regiones_{s_j}$, es el número de regiones o prototipos cuya clase es s_j.
- $expectativa_{s_j}$ es el número de patrones que se espera que cualquier prototipo r_i de la clase s_j clasifique correctamente. Este valor es calculado de forma relativa, utilizando la relación entre el número de patrones y el número de regiones de cada clase s_j. Este valor da una aproximación acerca de cuántos patrones debe clasificar cada prototipo con el fin de que todos ellos clasifiquen un número aproximado, y es calculado siguiendo la ecuación 5.2.

$$expectativa_{s_j} = \frac{\|VS_j\|}{regiones_{s_j}} \tag{5.2}$$

Toda esta información es calculada incrementalmente sin almacenar el conjunto entero, tal y como se describirá posteriormente.

Conjuntos Región-Clase

El conjunto V_{ij} se define como el conjunto de patrones que están localizados en la región r_i y que pertenecen a la clase s_j. Cada conjunto V_{ij} es la intersección entre el conjunto región VR_i (que se especificará posteriormente) y el conjunto clase, VS_j, y la pertenencia a este conjunto viene dada por la ecuación 5.3.

$$\forall v \in V, \forall r_i \in R, v \in V_{ij} \text{ sii } d(v, r_i) \leq d(v, r_{i'}), \forall r_{i'} \in R \text{ y } clase_{s_j} = clase_v \tag{5.3}$$

La medida de distancia utilizada es el error cuadrático medio, que se define en la ecuación 5.4.

$$d(x, y) = \sum_{i=0}^{i<K} (x[i] - y[i])^2 \tag{5.4}$$

donde K es la dimensión de los datos.

A partir de este conjunto, se pueden obtener varios datos importantes:

- $centroide_{V_{ij}}$: Es el centroide de los elementos contenidos en el conjunto V_{ij}, siguiendo la definición de centroide introducida en la sección 4.1.
- $\|V_{ij}\|$ es el tamaño del conjunto V_{ij}

De la misma forma que para los conjuntos VS, toda esta información puede ser calculada de forma incremental, sin almacenar el conjunto realmente.

Conjuntos Región

El conjunto región, VR_i, se define como el conjunto de patrones que están localizados en la región r_i. La función de pertenencia a este conjunto sigue la regla del vecino más cercano, y se define por la ecuación 5.5.

$$\forall v \in V, \forall r_i \in R, v \in VR_i \text{ sii } d(v, r_i) \leq d(v, r_{i'}), \forall r_{i'} \in R \tag{5.5}$$

Hay que destacar que en el caso de que varias regiones satisfagan esta condición, sólo una de ellas será elegida. La función de distancia utilizada es también el error cuadrático medio, definido en la ecuación 5.4.

De este conjunto, también se puede mantener cierta información que puede ser muy útil, como:

- $localización_{r_i}$, es la localización del prototipo r_i.
- $clase_{r_i}$ es la clase del prototipo r_i.
- $\|VR_i\|$ es el número de elementos del conjunto VR_i
- $éxito_{r_i}$ es el éxito de clasificación del prototipo. Se calcula tal y como se muestra en la ecuación 5.6, siendo $s_i = clase_{r_i}$.

$$éxito_{r_i} = \frac{\|V_{is_i}\|}{\|VR_i\|} \tag{5.6}$$

  donde $\|V_{is_i}\|$ es el número de prototipos situados en la región r_i y que pertenecen a la misma clase que el prototipo r_i (ver sección 5.3.2).

- $aportación_{r_i}$ es una relación entre el número de patrones que clasifica correctamente este prototipo y el número de patrones que está previsto que clasifique. Si $s_i = clase_{r_i}$, este último valor es $expectativas_i$ (definido en la ecuación 5.2) dividido por 2. La motivación de dividir la expectativa por 2 es dar al prototipo una medida menos rígida de su aportación, dado que la distribución de los patrones de cada clase puede no ser uniforme. Por tanto, este valor puede ser mayor que 1. Para calcular el valor de aportación, se utiliza la ecuación 5.7.

$$aportación_{r_i} = \frac{\|V_{is_i}\|}{\frac{expectativa_{s_i}}{2}} \tag{5.7}$$

- $calidad_{r_i}$ es la calidad del prototipo r_i, calculada a partir de la ecuación 5.8. Este valor es una relación entre el éxito de clasificación del prototipo, y su contribución a la clasificación del conjunto de patrones total. La principal idea es que la calidad de un prototipo es alta sólo si clasifica correctamente, y si lo hace para un conjunto de patrones suficientemente alto. Este valor se trunca al rango [0, 1].

$$calidad_{r_i} = max(1, éxito_{r_i} * aportación_{r_i}) \tag{5.8}$$

- $vecinos(r_i)$ es el conjunto de regiones que limitan con la región r_i.

5.3.3. El Algoritmo

La fase de aprendizaje es un proceso iterativo donde los prototipos pueden ejecutar diversos operadores, una vez que las características definidas en la sección 5.3.2 han sido calculadas. Estas operaciones son heurísticas que permiten a los prototipos cambiar su localización, introducir nuevos prototipos, etc. El algoritmo se resume en el diagrama de flujo mostrado en la Figura 5.2. Primero, hay una inicialización en la que se crea un clasificador compuesto únicamente de un prototipo. Después, el clasificador evoluciona mediante la ejecución, en un bucle, de todos los operadores diseñados. Estos operadores, que serán descritos posteriormente, utilizan la información adquirida al principio de cada bucle para ejecutarse.

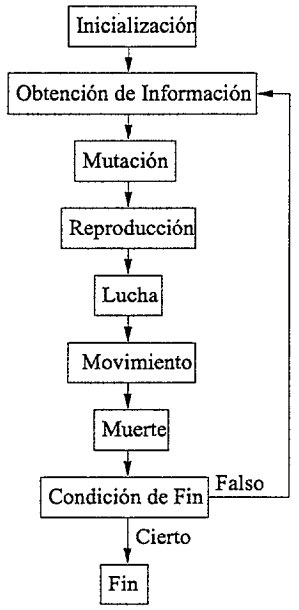


Figura 5.2: Diagrama de flujo del algoritmo ENNC.

Hay que destacar que la mayoría de los operadores o heurísticas definidas a continuación han sido tomados de la literatura. Por tanto, una de las características más relevantes de este algoritmo no son las heurísticas en sí, sino la forma en que son ejecutadas. Esta forma se plantea desde un punto de vista evolutivo, dando a los prototipos la capacidad de decidir cuándo ejecutarlos, e introduciendo un alto componente estocástico en estas decisiones, así como en la ejecución de dichos operadores.

Inicialización

Una característica muy relevante de este método es que se eliminan completamente las decisiones sobre la inicialización. Estas decisiones iniciales suelen resumirse en tres: el número de prototipos a utilizar, el conjunto inicial de prototipos, así como un parámetro de enfriamiento. El algoritmo ENNC permite aprender sin estos parámetros, ya que:

- El número inicial de prototipos es siempre 1. El método es capaz de generar nuevos prototipos, estabilizándose en el número más adecuado en términos de la medida de calidad definida anteriormente.
- La situación inicial del prototipo inicial es irrelevante, ya que define una única región que ocupa todo el entorno.
- No hay parámetro de enfriamiento, ya que el método ajusta automáticamente la intensidad de cambio en los prototipos, teniendo en cuenta sus calidades en cada iteración.

Obtención de Información

Al comienzo de cada iteración, el algoritmo debe calcular la información requerida para ejecutar los operadores. Esta información fue presentada en la sección anterior, y se refiere a los conjuntos región, clase, y región-clase.

La fase de obtención de información puede entenderse como una fase donde los patrones de entrenamiento son "insertados" en los conjuntos clase, región-clase y región, utilizando las funciones de pertenencia a estos conjuntos, definidos en las ecuaciones 5.1, 5.3 y 5.5, respectivamente. Al final de esta fase, todos los patrones habrán sido "introducidos" en los conjuntos, y la información acerca de ellos habrá sido calculada. Cabe destacar que los elementos de los conjuntos no son almacenados realmente, sino que sólo se calculan los atributos deseados sobre estos conjuntos. La forma de calcular estos atributos de forma incremental se detalla a continuación.

El centroide de un conjunto de vectores, P, puede ser calculado mediante la ecuación 5.9, sin más que obtener la suma de cada una de las componentes de los vectores, dividido por el número de vectores del conjunto, tal y como se definió en la ecuación 4.4 para la Iteración de Lloyd.

$$centroide(P) = \frac{1}{\|P\|} \sum_{x_j \in P} x_j \tag{5.9}$$

donde P es el conjunto de vectores o puntos cuyo centroide quiere ser calculado, y $\|P\|$ es su tamaño.

Sin embargo, el centroide puede ser calculado de forma incremental cada vez que un nuevo ejemplo, x, es introducido en un conjunto. Cuando esto ocurre, el tamaño del conjunto es incrementado en 1, y el centroide del conjunto puede ser recalculado siguiendo la ecuación 5.10. A esta operación se le denomina *inserción simple.*

$$centroide(P) = \frac{centroide(P) * \|P\| + x}{\|P\| + 1} \tag{5.10}$$

Cabe destacar que esta ecuación supone que el tamaño del conjunto no ha sido aún incrementado.

La ventaja de esta aproximación es que permite recalcular los centroides de un conjunto de ejemplos sin necesidad de que las instancias concretas que componen dicho conjunto sean almacenadas. Otra ventaja es que estos valores pueden ser recalculados de forma muy eficiente, por lo que cuando se producen muchas inserciones o extracciones de los conjuntos, los valores pueden ser recalculados continuamente sin apenas introducir ningún coste. La ecuación 5.11 muestra la operación de *extracción simple*, que permite recalcular el centroide ante una extracción de un elemento x del conjunto P.

$$centroide(P) = \frac{centroide(P) * \|P\| - x}{\|P\| - 1} \tag{5.11}$$

Cabe destacar, de nuevo, que esta ecuación supone que el tamaño del conjunto P no ha sido decrementado aún. Estas operaciones de inserción y extracción pueden ser ejecutadas no sólo cuando se desea insertar o extraer un elemento del conjunto, sino cuando se desean insertar o extraer muchos a la vez, todos ellos representados por otro centroide. Las ecuaciones 5.12 (*inserción múltiple*) y 5.13 (*extracción múltiple*) muestran estas operaciones, dados un conjunto P, sobre el que se realizan las operaciones, y un conjunto Q que va a ser insertado, y otro $W \subseteq P$ que va a ser extraído.

$$centroide(P) = \frac{centroide(P) * \|P\| + centroide(Q) * \|Q\|}{\|P\| + \|Q\|} \tag{5.12}$$

$$centroide(P) = \frac{centroide(P) * \|P\| - centroide(W) * \|W\|}{\|P\| - \|W\|} \tag{5.13}$$

Cabe recordar que estas ecuaciones se están definiendo para valores escalares. En el caso de vectores, las operaciones son equivalentes, pero es necesario realizarlas componente a componente. La utilidad de estas operaciones se mostrará cuando se definan los operadores.

La última característica que puede ser calculada de forma incremental es la de los vecinos de un prototipo, $vecinos(r_i)$. Para ello, se redefine el concepto de vecinos de forma que se dice que dos regiones r_i y r_j son vecinas, es decir, tienen una frontera común, si existe al menos un ejemplo del conjunto de patrones para el que el prototipo más cercano es r_i, y el segundo más cercano es r_j, o viceversa. Formalmente, esto queda expresado con la ecuación 5.14.

$$\begin{aligned} &\text{si } \exists v \in V, d(v, r_i), d(v, r_j) < d(v, r_t), \forall r_t \in R, r_i \neq r_j \rightarrow \\ &\quad r_i \in vecinos(r_j) \text{ y } r_j \in vecinos(r_i) \end{aligned} \tag{5.14}$$

A partir de esta ecuación se comprueba que, dado el conjunto de patrones V, todos los vecindarios pueden ser calculados de forma incremental. Cabe destacar, en cualquier caso, que la implicación de la ecuación anterior sólo se cumple en una dirección, por lo que no se asegura que los conjuntos de vecindario sean calculados totalmente, y ello depende del conjunto V.

Operador de Mutación

El objetivo de este operador es etiquetar cada prototipo a la clase más común en cada región. Una vez que los atributos sobre los conjuntos región son calculados, cada prototipo conoce el número de patrones de cada clase que están localizados en su región, por lo que el prototipo puede cambiar, si es necesario, y puede reconvertirse a la clase más abundante en su región. La Figura 5.3 muestra un ejemplo de ejecución de este operador. En el ejemplo, un prototipo de clase 2 cambia a clase 1, dado que los patrones de la clase 1 son los más populares en su región.

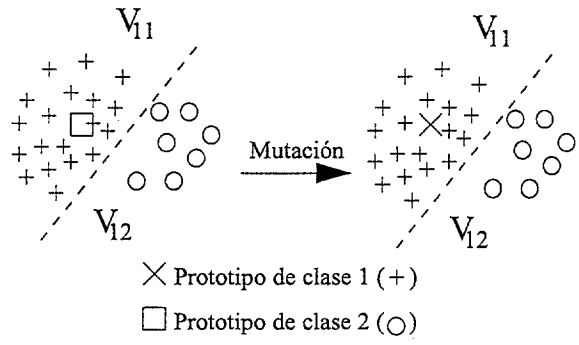


Figura 5.3: Ejemplo de ejecución del operador de mutación.

Esta forma de obtener la clase del prototipo es utilizada ampliamente cuando se utiliza aprendizaje no supervisado en problemas de clasificación [Bermejo and Cabestany, 2000, Pal *et al.*, 1993]. En estos trabajos, el algoritmo genera un conjunto de grupos o *clusters* que tienen en cuenta sólo la distribución de los datos. En una segunda fase, los *clusters* son etiquetados a la clase más abundante en el *cluster*. Sin embargo, en ENNC, el etiquetado es incluido en cada fase del proceso iterativo, y no sólo como una fase aislada y "a posteriori". Cabe recordar que la calidad de un prototipo depende de la relación entre el número de patrones que hay en su región, y el número de patrones que hay en su región y que pertenecen a su misma clase, por lo que la forma más sencilla de mejorar este valor es convirtiéndose a la clase más abundante. La formalización de este operador se muestra en la ecuación 5.15.

$$\forall r_i \in C, clase_{r_i} = \arg \max_{j \in S} \|V_{ij}\| \tag{5.15}$$

donde $\|V_{ij}\|$ es el tamaño del conjunto V_{ij}.

Operador de Reproducción

El objetivo de este operador es introducir nuevos prototipos en el clasificador. La inserción de nuevos prototipos es una decisión que es tomada por cada prototipo, en el sentido de que cada prototipo tiene la oportunidad de añadir nuevos prototipos con el fin de incrementar su propia calidad. La razón de proporcionar a los prototipos de esta capacidad es conseguir que cada uno de ellos sólo tenga en su región patrones que pertenezcan a su misma clase. Si se recuerda la representación planteada en la Figura 5.1, se puede ver que esto corresponde con una situación en la que sólo hay un conjunto V_{ij} no vacío, para cada r_i, es decir, sólo hay un V_{ij} en cada fila. La Figura 5.4 muestra una región r_1 que sólo tiene dos conjuntos V_{11} y V_{12} no vacíos, siendo 1 la clase del prototipo. Una forma sencilla de alcanzar la situación deseada es introduciendo otro prototipo r_2 de clase 2 que contenga el conjunto V_{12}, que por tanto, sería renombrado como V_{22}. Resumiendo, las regiones con patrones que pertenecen a diferentes clases, pueden crear nuevas regiones que contienen los prototipos de clase diferente a la de la región original.

La única decisión que queda por resolver es en qué situaciones los prototipos introducen nuevos prototipos. Para ello, cada prototipo r_i de la clase s_j, ejecuta una ruleta. Cada porción de la ruleta representa a un conjunto $V_{ij'}$, para todo $s_{j'} \in S$. El tamaño de cada porción es proporcional al número de elementos del conjunto $V_{ij'}$ al que representa. Los resultados posibles en esta ruleta son de dos tipos. Por un lado, si el conjunto ganador $V_{ij'}$ es el conjunto V_{ij}, es decir, si $j = j'$, no se ejecuta ninguna reproducción. Por otro

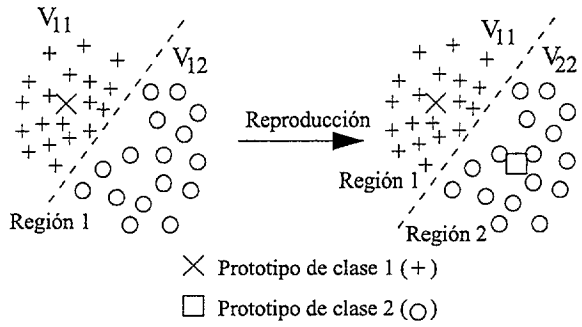


Figura 5.4: Ejemplo de ejecución del operador de reproducción.

lado, si el conjunto ganador $V_{ij'}$ no es el conjunto V_{ij}, es decir, si $j \neq j'$, se realiza la reproducción, y se crea una nueva región $r_{i'}$ para que contenga los patrones de $V_{ij'}$, que es renombrado a $V_{i'j'}$.

De esta forma es fácil ver que mientras se cumpla el objetivo de mantener un único conjunto V_{ij} no vacío por cada región r_i, los prototipos nunca se reproducirán. Sin embargo, si el número de conjuntos región-clase crece, así como sus tamaños, la probabilidad de reproducción será cada vez mayor.

En caso de que el prototipo r_i ejecute el operador de reproducción, se actualizan sus atributos ejecutando el operador de extracción múltiple, mostrado en la ecuación 5.13.

Operador de Lucha

El operador de lucha proporciona al prototipo la capacidad de obtener patrones de otras regiones. Formalmente, este operador permite a un prototipo, r_i, modificar sus conjuntos V_{ij} a partir del conjunto $V_{i'j}$ de otro prototipo $r_{i'}$, para $i \neq i'$. Este operador tiene varios pasos:

1. Elegir al prototipo rival $r_{i'}$ contra el cual luchar. Los prototipos se eligen a partir del conjunto $vecinos(r_i)$ definido en el apartado 5.3.2. Para decidir contra qué prototipo luchar del conjunto de prototipos vecinos, se utiliza también una ruleta para asignar a cada región $r_j \in vecinos(r_i)$ una porción proporcional a la diferencia entre su calidad y la calidad del prototipo r_i.

2. Decidir si luchar o no. La probabilidad de luchar entre dos prototipos r_i y $r_{i'}$ ($P_{lucha}(r_i, r_{i'})$) es igual a la distancia entre sus calidades, tal y como muestra la ecuación 5.16.

$$P_{lucha}(r_i, r_{i'}) = |calidad_{r_i} - calidad_{r_{i'}}| \tag{5.16}$$

3. Si el prototipo r_i decide luchar contra el prototipo $r_{i'}$, existen dos posibilidades. Dados $clase_{r_i} = s_i$, y $clase_{r_{i'}} = s_{i'}$:

   - Si $s_i \neq s_{i'}$ (cooperación). Los prototipos no pertenecen a la misma clase. En este caso, el prototipo $r_{i'}$ dará al prototipo r_i los patrones de la clase s_i. Esto se realiza mediante la inserción de los patrones del conjunto $V_{i's_i}$ en el conjunto V_{is_i} y haciendo $V_{i's_i}$ vacío. La Figura 5.5 muestra una ejecución de este operador, donde el prototipo 1, que posee instancias de las clases 1 y 2, cede al prototipo 2 las instancias de la clase 2, introduciendo las instancias antes contenidas en el conjunto V_{12} en el conjunto V_{22}.
   - Si $s_i = s_{i'}$ (competición). Ambos prototipos pertenecen a la misma clase. En este caso, los patrones pueden ser transferidos del conjunto $V_{i's_i}$ al conjunto V_{is_i}, o viceversa, dependiendo de quién gane la lucha. La Figura 5.6 muestra una ejecución de este operador, donde el prototipo 1 roba algunos patrones del prototipo 2. Quién es el ganador se decide nuevamente utilizando una ruleta con sólo dos partes, cada una de ellas perteneciente a cada prototipo, y con tamaños proporcionales a sus calidades. Además, la cantidad de patrones que se transfieren depende de una probabilidad proporcional a las calidades de ambos prototipos.

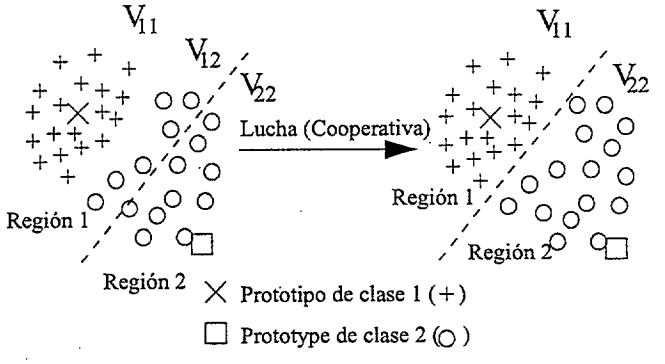


Figura 5.5: Ejemplo de ejecución del operador de lucha con cooperación.

Estos traspasos de instancias de unos conjuntos a otros se realizan mediante las operaciones de inserciones y extracciones múltiples definidas anteriormente. Por último, en las Figuras 5.5 y 5.6, se observa que el traspaso de patrones de un conjunto a otro sólo se hará efectivo con la ejecución del operador de movimiento, ya que será en ese momento cuando los prototipos

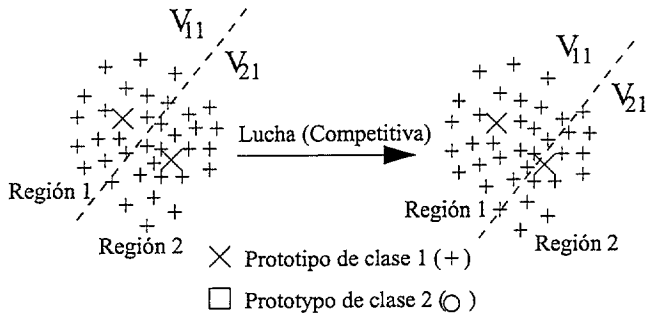


Figura 5.6: Ejemplo de ejecución del operador de lucha con competición.

recalculen su posición en función de los patrones que les han sido asignados, y esta modificación en sus posiciones será la que defina el límite entre sus regiones, tal y como se detalla a continuación.

Operador de Movimiento

El movimiento de los prototipos implica la recolocación de cada uno de ellos en el que se considera el mejor sitio para ellos. Por tanto, cada prototipo r_i, decide moverse al centroide del conjunto V_{ij}, siendo $clase_{r_i} = s_j$, es decir, toma como centroide el centroide del conjunto V_{ij} de su clase, tal y como muestra la ecuación 5.17.

$$localización_{r_i} = centroide_{V_{is_i}} \tag{5.17}$$

donde $s_i = clase_{r_i}$. La Figura 5.7 muestra la ejecución del operador de movimiento desde la situación alcanzada tras la ejecución del operador de lucha de la Figura 5.5. La figura muestra cómo el prototipo 2 cambia su situación al centroide del conjunto V_{22}.

Este operador está basado en el segundo paso de la iteración de Lloyd, planteada en la Figura 4.2, y permite realizar la optimización local del clasificador, incrementando su rendimiento.

Operador de Muerte

La probabilidad de que un prototipo sea eliminado es 1 menos el doble de su calidad, tal y como se define en la ecuación 5.18. Por tanto, los prototipos exitosos sobrevivirán con una probabilidad de 1, mientras que los prototipos poco útiles, con una calidad menor de 0,5, podrían morir. En la bibliografía se pueden encontrar un gran número de heurísticas sobre cómo reducir el

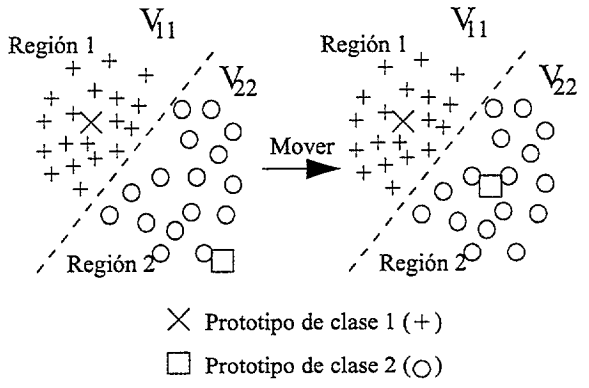


Figura 5.7: Ejemplo de ejecución del operador de movimiento.

número de prototipos en un clasificador [Fritzke, 1994, Patanè and Russo, 2001, Cagnoni and Valli, 1994].

$$P_{morir}(r_i) = \begin{cases} 0, \text{ cuando } calidad_{r_i} > 0{,}5 \\ 1 - 2 * calidad_{r_i}, \text{ cuando } calidad_{r_i} \leq 0{,}5 \end{cases} \tag{5.18}$$

Condición de Fin

La condición de fin es uno de los elementos más difíciles de definir en esta aproximación. Se supone que el algoritmo debe converger a una solución óptima, pero, ¿qué es una solución óptima? En este área, se suele decir que una solución es buena cuando obtiene un alto porcentaje en cuanto al éxito de clasificación, con un número reducido de prototipos. Sin embargo, decidir cuál de los dos parámetros tiene más peso, no es fácil. Por un lado, se puede plantear que incrementando el número de prototipos, también se puede aumentar el éxito de clasificación. Sin embargo, se pueden producir problemas de sobre-ajuste. Por otro lado, si se reduce el número de prototipos, se podría reducir también el éxito de clasificación en situaciones donde no existía el sobre-ajuste antes planteado.

La aproximación que se plantea en este trabajo es dejar que la población evolucione, almacenar el conjunto de clasificadores que se han ido generando en el proceso evolutivo, y dejar que el usuario elija el más apropiado. En cualquier caso, se pueden plantear muchas aproximaciones distintas para decidir cuándo parar: convergencia a un número de prototipos, convergencia a un porcentaje de éxito en la clasificación, una versión ponderada de ambos, etc. No obstante, los experimentos que se detallan posteriormente muestran

que la convergencia se produce tanto a un pequeño rango del valor del éxito en clasificación, como en número de prototipos obtenidos. A continuación se detallan algunos de los criterios de parada que se han implementado.

1. Número de iteraciones. El usuario puede definir un número máximo de iteraciones.

2. Éxito de clasificación. El usuario puede definir un éxito de clasificación deseado, por lo que el algoritmo, cuando llega a ese nivel, se detiene.

3. Éxito de clasificación y número de iteraciones. El usuario define su éxito de clasificación deseado, pero también define un número máximo de iteraciones, por lo que si el algoritmo no consigue llegar al éxito de clasificación indicado por el usuario en el número de iteraciones fijado, se para.

4. Convergencia a un número de prototipos. Este es un mecanismo automático que verifica la convergencia del algoritmo a un número de prototipos. Este criterio se basa en detener el algoritmo sólo cuando la frecuencia relativa de aparición de un tamaño del clasificador es mayor que cierto nivel, una vez que el algoritmo ha ejecutado un número mínimo de iteraciones.

5. Convergencia en el éxito de clasificación. Es un método automático de verificar la convergencia a un éxito de clasificación. El algoritmo se detiene cuando detecta que no es capaz de incrementar el éxito de clasificación en un alto número de iteraciones predefinido.

Una vez que la condición de fin ha sido fijada, el clasificador ganador puede ser obtenido de diversas formas. La más sencilla es elegir aquél con un porcentaje de éxito mayor sobre el conjunto de entrenamiento. Sin embargo, para evitar problemas de sobre-ajuste, se podría elegir aquél con un éxito de clasificación bueno, pero un número de prototipos más reducido.

5.4. Experimentos

En esta sección se muestran diversos experimentos realizados utilizando el algoritmo ENNC. El primero es realizado sobre un conjunto de datos de ejemplo que siguen 5 distribuciones gaussianas, que pretende mostrar de forma sencilla cómo funciona el algoritmo. El resto de experimentos comparan el algoritmo ENNC con otras aproximaciones previas extraídas de la bibliografía. Entre estas otras aproximaciones cabe destacar el uso de árboles de

decisión, concretamente C4.5 [Quinlan, 1993], reglas de decisión [Frank and Witten, 1998], Naive Bayes [Duda and Hart, 1973, John and Langley, 1995], e IBK [Aha and Kibler, 1991], para valores de $k = 1$ y $k = 3$, junto con otras aproximaciones que se irán introduciendo en cada momento. Así, en la sección 5.4.2 se muestra el segundo experimento sobre un dominio con datos distribuidos en forma de dos espirales entrelazadas, cada una de ellas asociada a una clase distinta. En la sección 5.4.3 se experimenta sobre un dominio de dos clases distribuidas de forma uniforme; *Straight Line Class Boundaries* se muestra en la sección 5.4.4; el *Iris Data Set* en la sección 5.4.5 y el dominio del *LED display* en la sección 5.4.6.

Para todos estos experimentos, la condición de fin establecida para ENNC es un número máximo de iteraciones suficiente para alcanzar buenas soluciones en cada caso.

5.4.1. Datos con Distribución Gaussiana

En este experimento se dispone de ejemplos de dos clases distintas que siguen las distribuciones mostradas en la Figura 5.8. El conjunto de datos consta de 250 ejemplos, de los cuales 200 fueron utilizados para entrenamiento, y 50 para test.

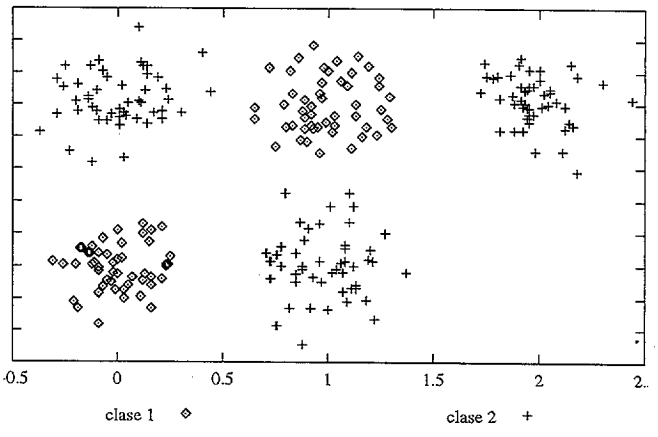


Figura 5.8: Datos que siguen distribuciones gaussianas.

El algoritmo ENNC comienza con una población de un único prototipo que se supone será capaz de evolucionar para encontrar el conjunto adecuado de prototipos. Esta evolución tiene una alta componente estocástica, por lo que se pueden obtener distintas soluciones en distintas ejecuciones. Para estudiar las posibles diferencias entre unas ejecuciones y otras, se repitió el

proceso de aprendizaje hasta 20 veces. El resultado fue que en las 20 ejecuciones se obtuvo un éxito de clasificación de un 100 %, tanto para los conjuntos de entrenamiento como para los de test. El número de prototipos generados en cada ejecución fue de 5 en 19 ejecuciones, obteniendo 6 sólo en una de ellas.

La Figura 5.9 muestra la evolución de una de estas ejecuciones. El eje x muestra las iteraciones del algoritmo, mientras que el eje y muestra el éxito de clasificación del clasificador en esa iteración, tanto para el conjunto de entrenamiento como para el de test, así como el número de prototipos utilizados. Se observa que tanto el conjunto de entrenamiento, como el de test, evolucionan de forma muy similar.

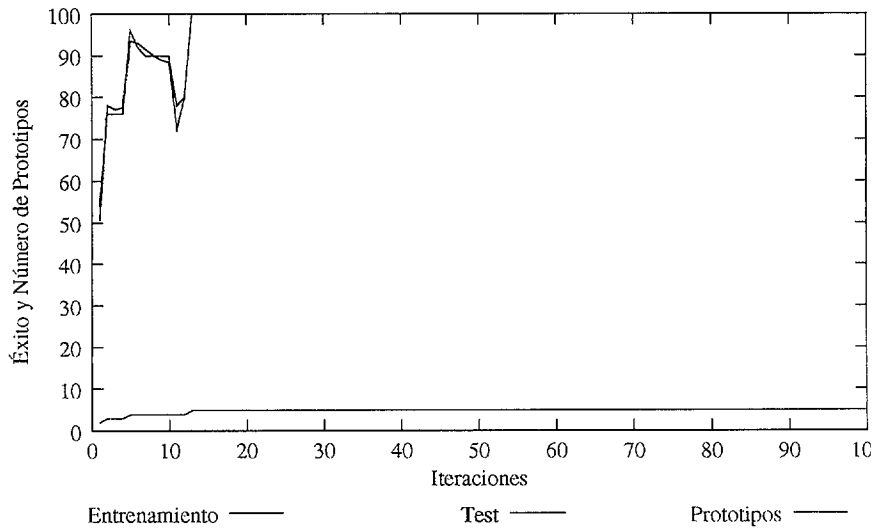
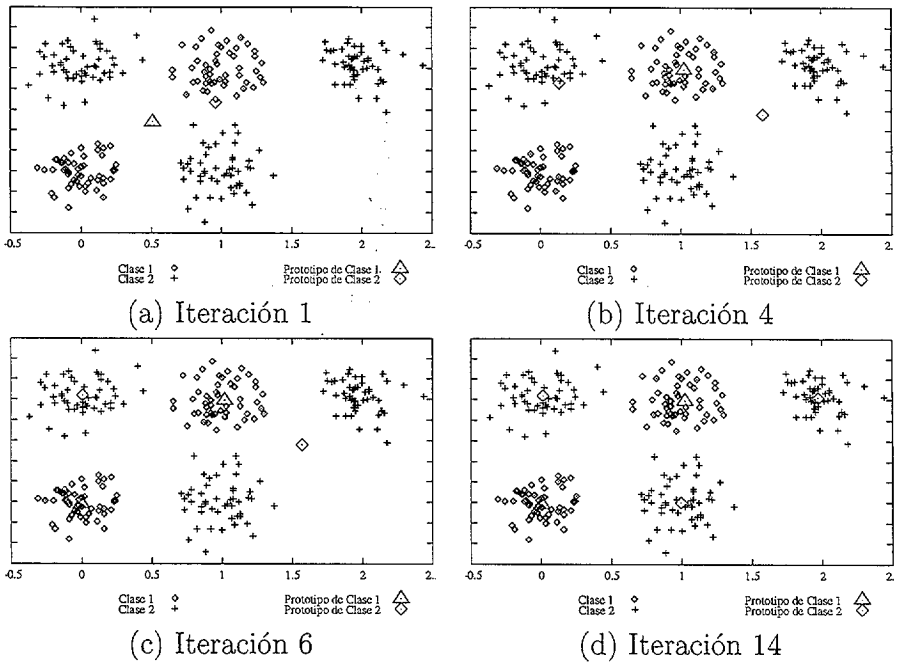


Figura 5.9: Evolución de una ejecución del algoritmo ENNC sobre datos con distribuciones gaussianas.

La Figura 5.10 permite entender esta evolución mostrando el estado del clasificador al final de cuatro de las iteraciones del algoritmo. La Figura 5.10(a) muestra los prototipos resultantes de la primera iteración, en la que un nuevo prototipo ha sido añadido a la población inicial. Las Figuras (b), (c) y (d) muestran los resultados de las iteraciones 4, 6, y 14 respectivamente, cada una de ellas con un prototipo más que en la figura anterior.



(a) Iteración 1 (b) Iteración 4

(c) Iteración 6 (d) Iteración 14

Figura 5.10: Evolución del clasificador.

5.4.2. Datos en Espiral

Este experimento consiste en datos que siguen dos espirales, tal y como muestra la Figura 5.11. Los ejemplos que están en una misma espiral pertenecen a la misma clase, y hay 500 ejemplos para cada espiral. En este caso, 900 ejemplos del total fueron utilizados para entrenamiento, y 100 para test.

Al igual que en el experimento anterior, se han realizado 20 ejecuciones del algoritmo ENNC, cada una de ellas de una longitud de 300 iteraciones. Los resultados de estos experimentos se resumen en la Tabla 5.1. Dicha tabla ofrece, para cada una de las ejecuciones, los datos referentes al clasificador que obtiene mayor porcentaje de clasificación sobre el conjunto de entrenamiento. Este clasificador es elegido como salida del aprendizaje, y es utilizado para realizar el test. La tabla muestra que para los datos de entrenamiento, se obtienen porcentajes de éxito de alrededor del 99 %, mientras que se obtiene un éxito de clasificación de entre un 94-98 % para el conjunto de test, lo que supone una diferencia máxima de 4 errores para dicho conjunto, entre unas y otras ejecuciones. Por tanto, aunque no todos los resultados son iguales, se mantienen en un rango muy pequeño. De igual forma, el número de prototipos de los clasificadores ganadores es muy similar, manteniéndose cercano a 80.

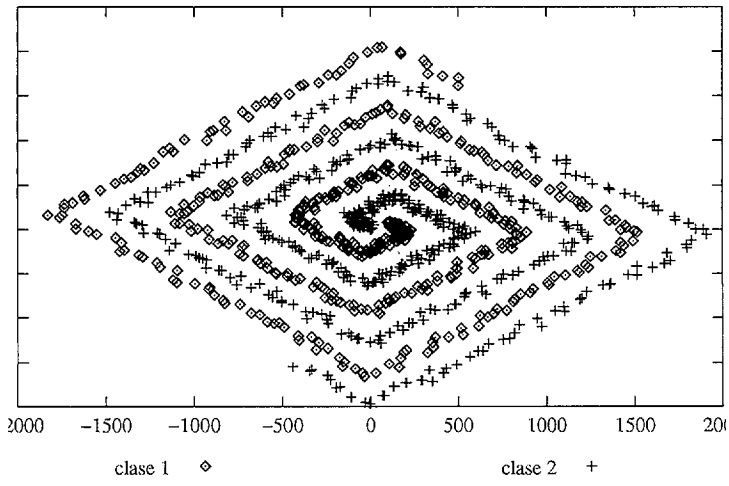


Figura 5.11: Datos en espiral.

Iteración	Prototipos	Entrenamiento (%)	Test (%)
289	82	99,33	97,0
158	75	99,0	96,0
174	81	99,11	95,0
267	81	99,33	96,0
282	81	99,44	97,0
260	78	99,11	97,0
170	79	99,56	96,0
182	82	99,33	97,0
120	79	98,67	95,0
281	73	98,9	96,0
300	84	99,44	97,0
123	84	99,56	96,0
81	79	99,33	94,0
260	82	99,44	96,0
131	77	98,67	95,0
81	81	99,33	98,0
97	79	99,33	96,0
299	78	99,56	96,0
291	84	99,78	97,0
179	75	99,0	97,0

Tabla 5.1: Resultados de diferentes ejecuciones del algoritmo ENNC sobre datos en espiral.

La tabla 5.2 muestra la comparativa de estos resultados con otros clasificadores, mostrando el mejor y el peor resultado de ENNC. En ella se observa que los resultados para C4.5, reglas de decisión (PART) y Naive Bayes son bastante pobres, y quedan muy lejos de las soluciones proporcionadas con ENNC, tanto para su mejor, como para su peor solución. No obstante, tam-

bién se observa que es un dominio donde IBK obtiene soluciones de un 100 % de éxito en clasificación, tal y como era de esperar en un dominio con las instancias de cada clase tan separadas entre sí, como mostraba la Figura 5.11.

C4.5	PART	Naive Bayes	IBK ($k = 1$)	IBK ($k = 3$)	ENNC (mejor)	ENNC (peor)
62	56	50	100	100	98	94

Tabla 5.2: Comparativas sobre datos en espiral.

La evolución de una ejecución del algoritmo ENNC se muestra en la Figura 5.12, con el número de prototipos y el éxito de clasificación obtenido sobre los conjuntos de entrenamiento y test. En dicha ejecución se muestra que en sólo 25 iteraciones del algoritmo, el número de prototipos se incrementa hasta 64, proporcionando ya un éxito de clasificación de un 91,34 % sobre el conjunto de entrenamiento, y de un 85 % sobre el conjunto de test. Sin embargo, el número de prototipos se incrementa aún más hasta el rango 70-80, donde continúa la búsqueda, consiguiendo el mejor resultado en la iteración 203 para el conjunto de entrenamiento del 98,56 %, y del 96 % para el conjunto de test, con 76 prototipos. El éxito de clasificación más alto para el conjunto de entrenamiento es del 99,34 % en la iteración 176, donde el éxito sobre el conjunto de test es del 94 %. El clasificador obtenido en esta ejecución se muestra en la Figura 5.13.

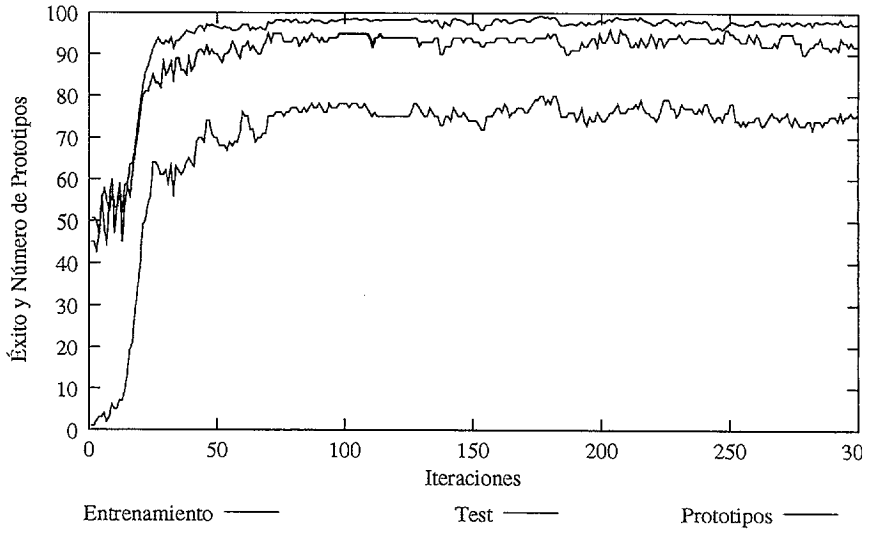


Figura 5.12: Evolución del algoritmo ENNC sobre una ejecución en el conjunto de datos en espiral.

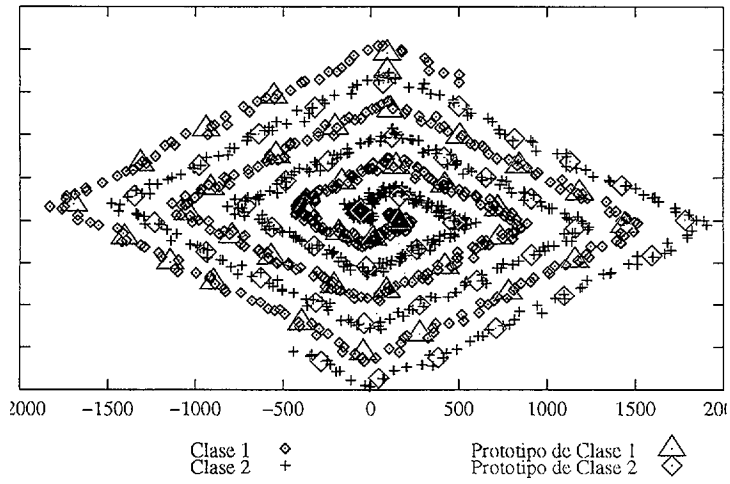


Figura 5.13: Clasificador de 76 prototipos obtenido por ENNC sobre el conjunto de datos en espiral.

5.4.3. Datos Distribuidos Uniformemente

En este experimento, el algoritmo ENNC ha sido probado en un dominio con datos pertenecientes a dos clases, y que siguen la distribución mostrada en la Figura 5.14, tal y como se define en [Burrascano, 1991]. Al igual que en ese trabajo, los datos contienen 2000 instancias de cada clase, utilizando 500 de ellas para el entrenamiento, y las 1.500 restantes para el test.

Siguiendo los planteamientos definidos en los anteriores experimentos, se ha vuelto a ejecutar el algoritmo *ENNC* 20 veces, con un número máximo de 300 iteraciones. La Tabla 5.3 resume los resultados obtenidos en estas 20 ejecuciones, mostrando la misma información que en los casos anteriores, es decir, la iteración donde se obtuvo la solución, el porcentaje de éxito en la clasificación para los conjuntos de entrenamiento y de test, y el número de prototipos del clasificador obtenido.

En la tabla se muestra que en 16 de las 20 ejecuciones, el algoritmo devuelve la solución de sólo 2 prototipos, con un porcentaje de éxito del 98,6 % sobre los conjuntos de entrenamiento y test, un valor que se alcanza en la primera iteración del algoritmo. Este resultado tan sorprendente es debido a que la forma en que el algoritmo ENNC es inicializado, así como los operadores que utiliza, son muy adecuados para encontrar esta solución. En la inicialización se parte de un único prototipo, que dado que hay el mismo número de instancias de cada clase, obtiene un éxito de clasificación del 50 %. Este prototipo tratará de reproducirse para aumentar su porcentaje

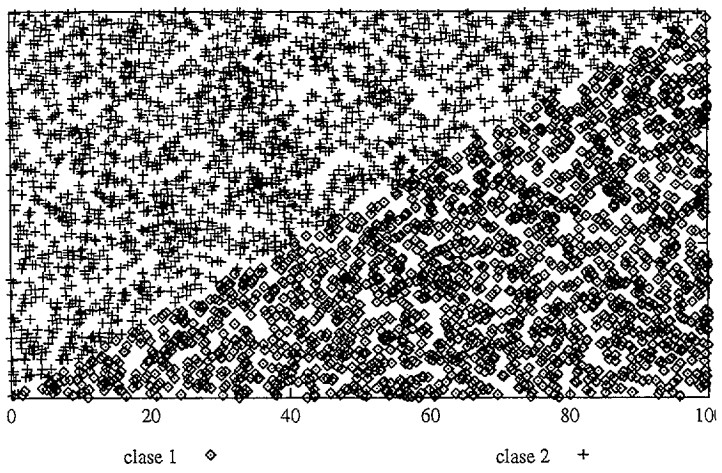


Figura 5.14: Datos distribuidos uniformemente.

Iteración	Prototipos	Entrenamiento (%)	Test (%)
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
266	19	98,8	98,3
1	2	98,6	98,6
279	30	98,8	98,7
1	2	98,6	98,6
1	2	98,6	98,6
1	2	98,6	98,6
179	14	99,1	98,73
37	2	98,7	98,93

Tabla 5.3: Resultados de diferentes ejecuciones del algoritmo ENNC sobre datos distribuidos uniformemente.

de clasificación, cediendo al nuevo prototipo las instancias de entrenamiento que él no desea. Por último, los dos prototipos ejecutarán el operador de movimiento, colocándose cada uno de ellos en el centroide de los datos de su correspondiente clase. Esta situación sería óptima para un conjunto de datos de entrenamiento infinito perfectamente distribuido. No obstante, como el número de datos es limitado, los porcentajes de éxito no llegan al 100 %.

Dado que al algoritmo se le permite buscar nuevas soluciones, hasta cumplir las 300 iteraciones máximas, en algunas ocasiones mejora el valor anterior, obteniendo el 98.73 para clasificadores de más prototipos (12), e incluso una solución de 2 prototipos que alcanza un 98,93 % de éxito para el conjunto de test. Cabe destacar nuevamente, que aunque el algoritmo produce distintas soluciones, todas ellas son muy buenas desde el punto de vista del éxito de clasificación y del número de prototipos obtenido.

La Figura 5.15 muestra la evolución de una ejecución del algoritmo. Como en el resto de los experimentos, el eje x muestra las iteraciones del algoritmo, mientras que el eje y muestra el porcentaje de éxito de clasificación, así como el número de prototipos obtenido.

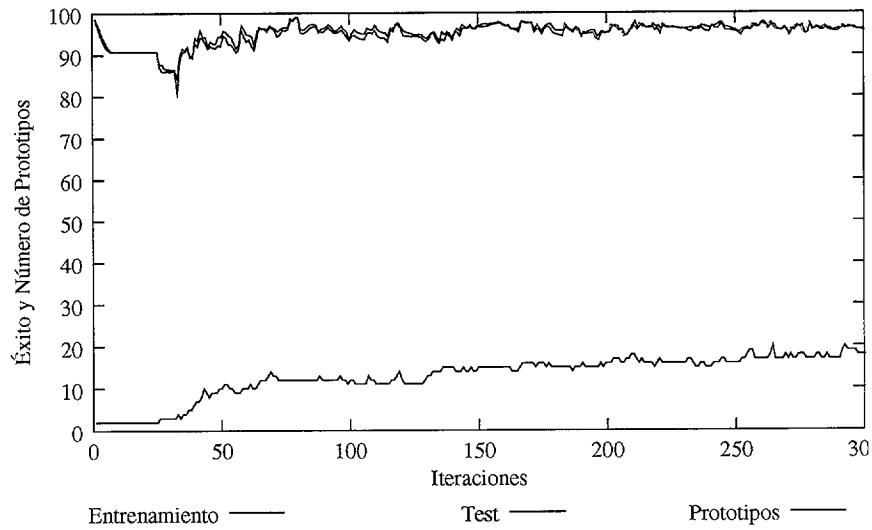


Figura 5.15: Evolución del algoritmo ENNC sobre datos distribuidos uniformemente.

La figura muestra que el algoritmo obtiene un clasificador de sólo 2 atributos en la primera iteración, y que este valor va creciendo en la búsqueda de clasificadores con mejores porcentajes de éxito en la clasificación. Así, se obtiene un clasificador de 12 prototipos en la iteración 79 que obtiene un porcentaje de éxito del 98.967 %.

La Tabla 5.4 muestra una comparativa de estos resultados con otras aproximaciones anteriores, como el algoritmo LVQ [Kohonen, 1984] y dos implementaciones de redes neuronales probabilísticas (PNN) [Specht, 1990] que permiten reducir la estructura de la red de neuronas. LVQ-PNN [Burrascano, 1991] lo consigue utilizando LVQ para encontrar un conjunto reducido de

Algoritmo	Número de prototipos/Neuronas	Éxito
LVQ	10	96,13
LVQ	100	97,77
LVQ-PNN	10	96,99
LVQ-PNN	100	98,17
PNN (Mao)	8	98,85
ENNC (mejor)	2	98,93
ENNC (peor)	2	98,6

Tabla 5.4: Resultados comparativos en el dominio de datos distribuidos uniformemente.

neuronas en su segunda capa, en vez de usar tantas neuronas como patrones de entrenamiento, típico de las redes probabilísticas. En [Burrascano, 1991] se dan valores para dos ejecuciones distintas, una utilizando 10 neuronas y la otra utilizando 100, dando el mejor resultado sobre el conjunto de test del 98,17 % para el segundo caso.

La segunda aproximación dada en [Mao *et al.*, 2000] permite diseñar la red probabilística determinando el parámetro de aprendizaje de la red mediante algoritmos genéticos [Holland, 1975], y definiendo la estructura de la red mediante un algoritmo que selecciona las neuronas importantes, por lo que el número de neuronas a utilizar se calcula de forma automática por el algoritmo. En la Tabla 5.4 se muestra que el resultado obtenido por ENNC con dos prototipos (98.6 %) es muy similar a la mejor solución obtenida por esta versión de PNN, con un 98,85 %. La Figura 5.16 muestra dos de las soluciones generadas con el algoritmo ENNC, una de ellas con 2 prototipos, y la otra con 12.

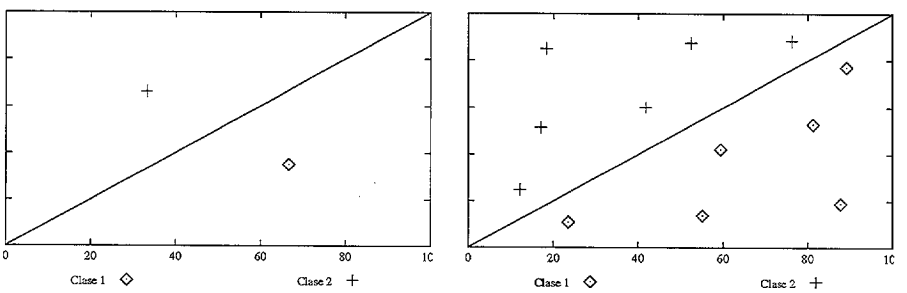


Figura 5.16: Clasificadores obtenidos con ENNC sobre datos distribuidos uniformemente.

Por último, la tabla 5.5 muestra comparativas son algunos otros algoritmos, verificando que los resultados de ENNC están siempre entre los mejores, en este caso sólo igualado también por *IBK*, tanto para $k = 1$ como para

$k = 3$.

C4.5	PART	Naive Bayes	IBK ($k = 1$)	IBK ($k = 3$)	ENNC (mejor)	ENNC (peor)
96,97	96,6	96,77	98,7	98,47	98,93	98,6

Tabla 5.5: Comparativas sobre datos en espiral.

5.4.4. Dominio *Straight Line Class Boundaries*

Este dominio de clasificación fue definido en [Hart, 1968] y utilizado en [Geva and Sitte, 1991] para mostrar el rendimiento del clasificador *DSM*. El dominio consiste en dos clases distintas, distribuidas tal y como muestra la Figura 5.17, con 6400 ejemplos para entrenamiento, y otros 6400 para test. El algoritmo DSM pertenece a la familia de los algoritmos LVQ y funciona del siguiente modo: primero selecciona un pequeño conjunto de entrenamiento y lo utiliza como semilla del algoritmo de aprendizaje. El algoritmo adapta gradualmente su posición para aproximar correctamente el conjunto de entrenamiento completo.

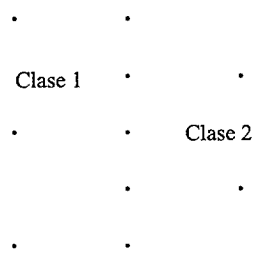


Figura 5.17: Dominio *Straight Line Class Boundaries* y una solución óptima.

La Figura 5.17 también muestra una solución al problema con únicamente 10 prototipos. Al igual que en los otros experimentos, se ha ejecutado el algoritmo ENNC 20 veces, y los resultados se muestran en la Tabla 5.6. Se comprueba que el algoritmo genera soluciones en el rango de 27-33 prototipos, con un éxito de clasificación de aproximadamente el 98 %. El éxito medio obtenido en las 20 ejecuciones es del 98.14 %, y este valor es el que se utiliza para compararlo con las soluciones expuestas en [Geva and Sitte, 1991] y resumidas en la Tabla 5.7.

Dicha tabla muestra que el algoritmo ENNC mejora los resultados obtenidos por LVQ1, y que está muy cercano a los producidos por el algoritmo

Iteración	Prototipos	Entrenamiento (%)	Test (%)
183	32	98,07	98,04
170	30	98,17	98,21
194	32	98,22	98,21
234	28	98,18	98,23
300	33	98,37	98,37
262	33	98,08	98,03
248	38	98,35	98,31
119	28	98,21	98,16
228	29	98,46	98,49
243	33	98,28	98,23
87	27	98,11	98,06
184	30	98,10	98,11
202	29	98,23	98,24
175	26	97,82	97,76
220	29	98,41	98,36
235	31	98,12	98,14
245	31	98,08	98,08
264	32	98,18	98,19
159	33	97,71	97,74
292	30	97,92	97,92

Tabla 5.6: Resultados de distintas ejecuciones del algoritmo ENNC en el dominio *Straight Line Class Boundaries Domain.*

Prototipos	DSM	LVQ1	*Backpropagation*
6	92,86	81,00	90,58
8	96,18	80,45	98,47
9	98,14	85,36	98,73
10	99,57	87,66	98,34
20	99,55	95,66	98,47
24	99,59	96,94	98,62
50	99,51	97,49	98,44
250	99,21	98,16	98,45

Tabla 5.7: Resultados comparativos sobre el dominio *Straight Line Class Boundaries.*

de retro-propagación (*backpropagation*), mientras que no es capaz de alcanzar los mejores resultados de DSM. No obstante, hay que destacar que el algoritmo ENNC define automáticamente el número de prototipos a utilizar, mientras que es un parámetro de entrada en DSM.

La Figura 5.18 muestra la evolución de una de las ejecuciones del algoritmo. En las iteraciones iniciales, el número de prototipos utilizado es inferior a 10, manteniendo el éxito de clasificación por debajo del 90 %. Una vez que el número de prototipos supera el valor de 10, el éxito de clasificación comienza a obtener resultados de alrededor del 98 %. El número de prototipos se incrementa sucesivamente para mejorar ese valor. Por último, las curvas para el conjunto de entrenamiento y para el conjunto de test se mantienen

prácticamente idénticas durante el aprendizaje, mostrando que no se producen problemas de sobre-ajuste.

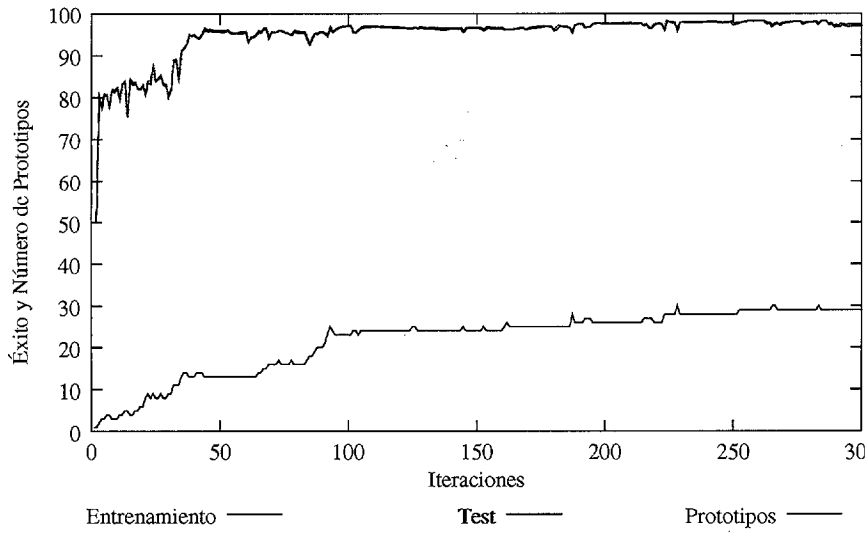


Figura 5.18: Ejecución del algoritmo ENNC sobre el dominio *Straight Line Class Boundaries Domain.*

La Figura 5.19 muestra los prototipos del clasificador obtenido en la primera ejecución del algoritmo ENNC, en la que se generaban 30 prototipos, obteniendo un porcentaje de clasificación correcta del 98,67 %. La Tabla 5.8 muestra la comparativa de ENNC con algunos otros algoritmos, tal y como se hizo en los experimentos anteriores. En este experimento se muestra que tanto los árboles de decisión, como las reglas, obtienen muy buenos resultados, prácticamente del 100 % de éxito. Esto es debido a que es un dominio muy bueno para este tipo de clasificadores, ya que las dos clases son separables con escasas decisiones. El peor resultado lo tiene *Naive Bayes*, mientras que IBK obtiene también resultados superiores a ENNC, tanto para $k = 1$ como para $k = 3$.

C4.5	PART	Naive Bayes	IBK ($k = 1$)	IBK ($k = 3$)	ENNC (mejor)	ENNC (peor)
99,99	99,99	81,04	99,7	99,7	98,46	97,71

Tabla 5.8: Comparativas sobre dominio *Straight Line Class Boundaries.*

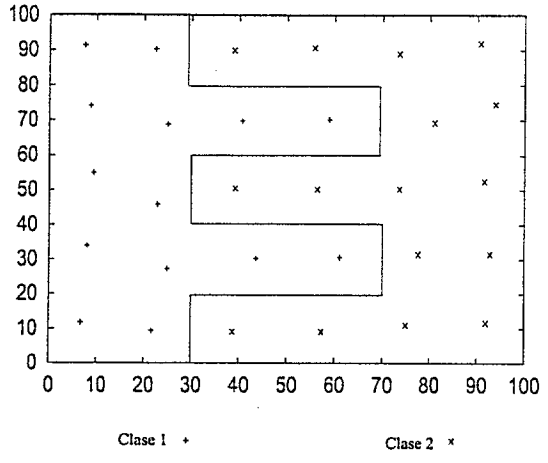


Figura 5.19: Clasificador de 30 prototipos obtenido por ENNC en el dominio *Straight Line Class Boundaries.*

5.4.5. *Iris Data Set*

El dominio *Iris Data Set* del *UCI Machine Learning Repository* [1] [Blake and Merz, 1998] es utilizado en este experimento. El conjunto consiste de 150 ejemplos de tres clases, donde cada una de las clases contiene 50 ejemplos. La dimensión del espacio de características es cuatro. En este caso, y por razones de comparación con otros métodos, el total del conjunto fue utilizado tanto para entrenamiento como para test.

El algoritmo ENNC ha sido ejecutado 20 veces, como en los experimentos anteriores, y los resultados se muestran en la Tabla 5.9. Dicha tabla muestra las mejores soluciones generadas, en las que sólo se producen 2 y 3 errores de clasificación (obteniendo un 98.667 % y un 98.000 % de acierto respectivamente). Estas soluciones se obtienen para clasificadores de 5 prototipos (que generan 3 errores) y de un rango de 6-12 prototipos (que generan 2 errores).

La Figura 5.20 muestra la evolución de una de las ejecuciones. Como en el resto de los experimentos, la figura representa las iteraciones del algoritmo en el eje x, y el porcentaje de éxito en el eje y. No obstante, sólo se muestra el éxito de clasificación para el total del conjunto. La figura muestra que el resultado de 5 prototipos y un 98 % de éxito es encontrado en sólo 15 iteraciones, tras mantenerse varias iteraciones con 3 prototipos, con alrededor de un 90 % de éxito. Tras alcanzar el 98 % de éxito, el algoritmo continúa la evolución buscando nuevos tamaños del clasificador, encontrando el mejor

[1] http://www.ics.uci.edu/~mlearn/MLRepository.html

Iteración	Éxito	Prototipos
86	98,667	7
39	98,000	6
138	98,667	7
107	98,667	7
152	98,667	11
236	98,667	8
24	98,667	7
24	98,000	5
43	98,667	12
19	98,000	5
25	98,000	5
252	98,667	10
133	98,667	8
134	98,667	10
145	98,667	9
10	98,000	5
12	98,000	5
143	98,667	7
255	98,667	11
95	98,000	6

Tabla 5.9: Resultados de diferentes ejecuciones de ENNC sobre el dominio *Iris Data Set.*

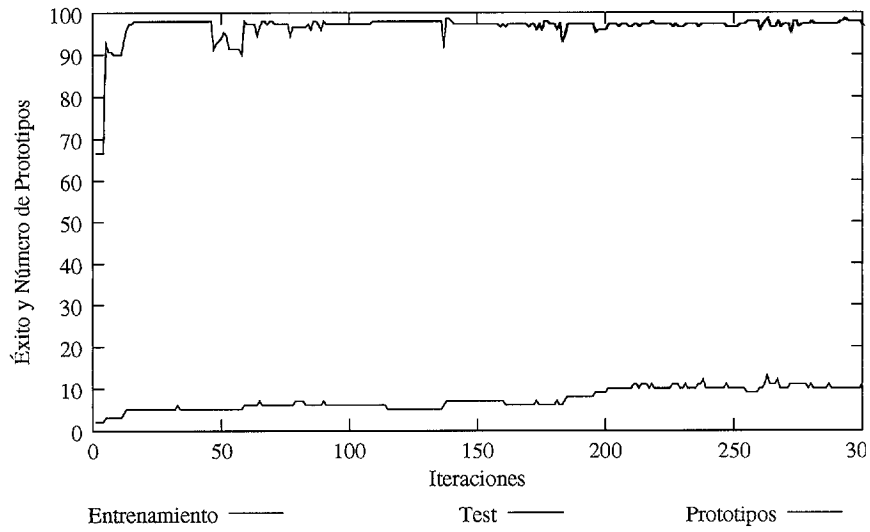


Figura 5.20: Evolución del algoritmo ENNC en el dominio *Iris Data Set.*

resultado del 98,67 % para 7 prototipos en la iteración 138.

Estos resultados son comparados con los presentados en [Kuncheva and Bezdek, 1998, Mao *et al.*, 2000], y se resumen en la Tabla 5.10, donde se muestran el número de prototipos utilizados y los errores de clasificación

cometidos. Los algoritmos comparados son LVQ [Kohonen, 1984], GLVQ-F, DR [Bezdek *et al.*, 1998], MFCM-3 [Kuncheva and Bezdek, 1998], así como la versión de PNN [Mao *et al.*, 2000] planteada en la sección 5.4.3. Se observa que el algoritmo ENNC mejora los resultados de MFCM-3, LVQ Y GLVQ-F, pero no alcanza el mejor resultado de la versión de PNN de Mao, que comete sólo un error de clasificación con sólo 3 prototipos. Para ENNC, se muestran los dos mejores resultados obtenidos, para 5 y 7 prototipos, en distintas ejecuciones.

Algoritmo	MFCM-3	LVQ	LVQ	GLVQ-F	DR	DR	I. PNN	ENNC	ENNC
Prototipos	7	7	3	8	5	3	3	5	7
Errores	11	3	17	3	3	10	1	3	2

Tabla 5.10: Resultados comparativos en el dominio *Iris Data Set.*

Por último, los resultados son comparados también con otros algoritmos, tal y como se muestra en la Tabla 5.11. Dicha tabla muestra cómo C4.5 produce también sólo 3 errores de clasificación, mientras que PART produce 4 y NAIVE BAYES 6. Obviamente, IBK con $k = 1$ produce 0 errores, ya que se usa como conjunto de test el mismo que de entrenamiento, pero se observa que si el valor de k se incrementa a 3, el número de errores es incrementado hasta 5.

C4.5	PART	Naive Bayes	IBK ($k = 1$)	IBK ($k = 3$)	ENNC (mejor)	ENNC (peor)
3	4	6	0	5	2	3

Tabla 5.11: Comparativas sobre dominio *Iris Data Set.*

5.4.6. Dominio del Visor de LED

El objetivo de este último experimento es mostrar el rendimiento del clasificador ENNC en un dominio con distintos niveles de ruido. El dominio elegido es el problema de clasificación de dígitos en un LED [Brieman *et al.*, 1984], obtenido también de UCI [Blake and Merz, 1998]. Este dominio consiste en reconocer cuál de los 10 dígitos está siendo mostrado en un LED. Para cada segmento que compone el LED, su valor está a 1 si el segmento está encendido, y a 0 si el segmento está apagado. Además, cada segmento del LED tiene un porcentaje de fallo, por lo que el valor de un segmento puede ser 1, incluso cuando el segmento está apagado, y viceversa. Variando este porcentaje de fallo, se introduce más o menos ruido en el dominio.

Desde el punto de vista de clasificación del prototipo más cercano, este problema tiene una solución óptima teórica de 10 prototipos, uno por cada LED (o clase). Por ejemplo, la Figura 5.21 muestra la correspondencia entre cada uno de los segmentos y la posición que ocupan en su representación como cadena de 7 bits. Si se supone que el LED está mostrando el número 3, el prototipo correspondiente para este número sería el {1, 0, 1, 1, 0, 1, 1}.

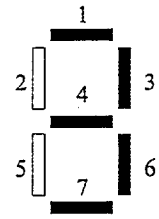


Figura 5.21: LED representando al número 3.

Se han generado dos conjuntos de 1.000 ejemplos, uno para entrenamiento y otro para test. Además, el primer conjunto se ha dividido de nuevo en dos subconjuntos de 500 instancias. El primero puede ser considerado como el conjunto de entrenamiento y que será utilizado por el algoritmo ENNC para evolucionar. El segundo subconjunto se considera como de validación, y será utilizado para seleccionar el clasificador final que genera como salida el algoritmo en cada ejecución.

La Figura 5.22 muestra la evolución de un conjunto de prototipos durante 300 iteraciones del algoritmo ENNC sobre los 500 ejemplos de entrenamiento y de validación. En este caso, los datos tienen un ruido añadido de un 10 %. La figura muestra que en sólo 13 iteraciones, el algoritmo es capaz de alcanzar un conjunto de 10 prototipos, con un éxito de clasificación de un 67 % sobre el conjunto de entrenamiento, el mismo que para el conjunto de validación. La mejor solución alcanzada se produce en la iteración 282, con un éxito de clasificación de un 74,4 % para el conjunto de entrenamiento, y de un 76 % sobre el conjunto de validación. Este clasificador es seleccionado para pasar el test, obteniendo un éxito de clasificación de un 72,9 % sobre los 1.000 ejemplos de test. Para ampliar las comparativas, también se ha ejecutado el test sobre el clasificador óptimo de 10 prototipos planteado anteriormente. Este clasificador obtiene un éxito del 74,5 % sobre el conjunto de test, sólo un 1,6 % mejor que la solución propuesta por el algoritmo *ENNC*. Sirva como referencia que el porcentaje de clasificación teórico Bayesiano para este ejemplo es del 74 %.

Se ha realizado una experimentación más extensa en este dominio. El experimento descrito anteriormente para un porcentaje de ruido de un 10 %,

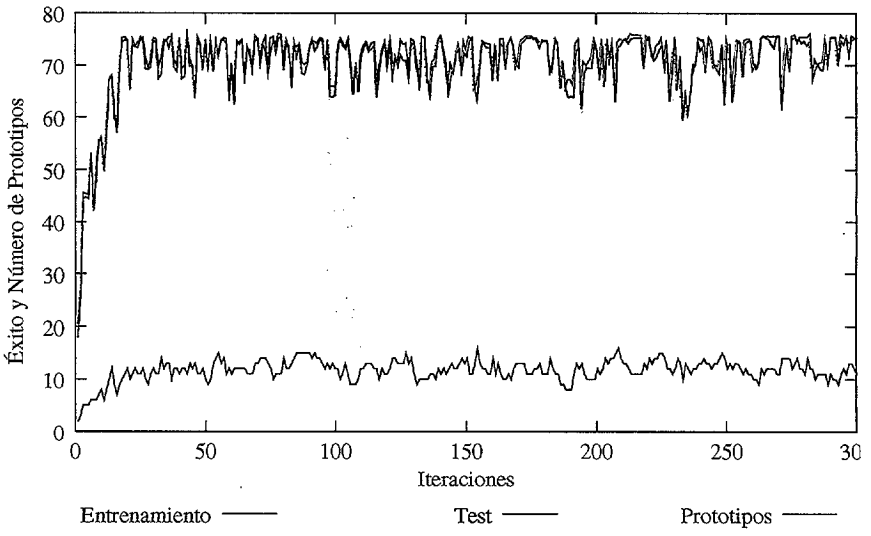


Figura 5.22: Evolución del algoritmo ENNC en el dominio del visor de LED.

ha sido ejecutado también para porcentajes de ruido que varían desde el 0 % hasta el 20 %. Para cada valor de ruido, el algoritmo ENNC se ha ejecutado 20 veces, utilizando 500 ejemplos para entrenar, otros 500 para seleccionar el clasificador y otros 1.000 para realizar el test. El test ha sido ejecutado sobre la aproximación óptima (generada a mano) de un prototipo por número, descrita anteriormente. Además, la experimentación también ha sido realizada sobre otros clasificadores, utilizando la misma cantidad de datos para el entrenamiento y para el test (1.000 para cada uno). La Figura 5.23 resume todos estos resultados.

Cuando el ruido es 0, ENNC obtiene la solución óptima de un conjunto de 10 prototipos, con éxitos para el conjunto de entrenamiento y de test del 100 % en las 20 ejecuciones realizadas. Según se incrementa el porcentaje de ruido, el éxito de clasificación decrece para todos los conjuntos de datos, tanto para el clasificador ENNC como para la solución óptima. Sin embargo, se observa que ambos valores se mantienen muy similares hasta el porcentaje de ruido del 11 %. A partir de ese punto, el clasificador ENNC produce soluciones sobre-ajustadas al conjunto de entrenamiento, pero los resultados sobre el test se mantienen similares a los del clasificador óptimo, mostrando que no se produce sobre-ajuste en el conjunto de test, y que el éxito de clasificación se mantiene cercano al óptimo. En cuanto al resto de clasificadores, se observa que todos ellos mantienen curvas muy similares a las de ENNC, excepto para KNN con $k = 1$, donde el porcentaje de éxito

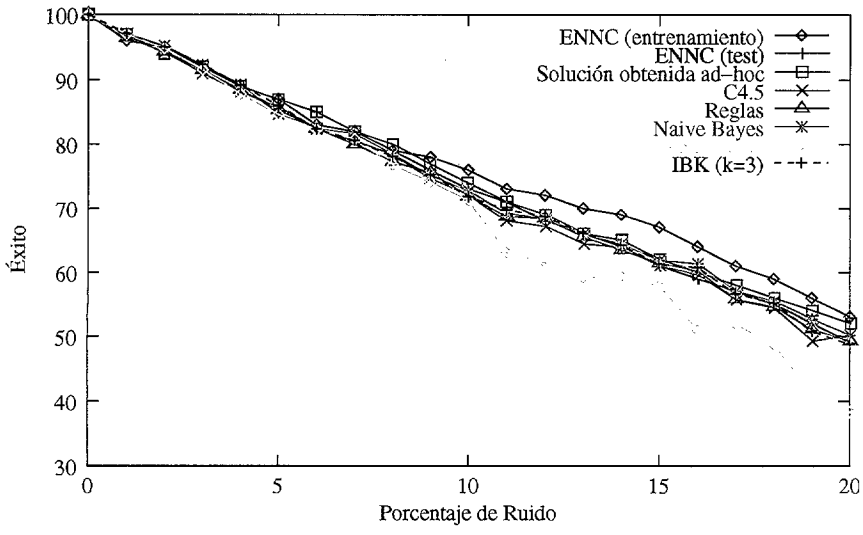


Figura 5.23: Experimentos ejecutados sobre el dominio del visor de LED.

es bastante inferior a partir de niveles de ruido del 11 %.

La capacidad de generalización del clasificador puede ser comprendida fácilmente si se observa la evolución en el número de prototipos obtenido, y mostrado en la Figura 5.24. Cuando el ruido es del 11 %, el número medio de prototipos obtenido en las 20 ejecuciones realizadas para cada nivel de ruido es todavía inferior a 11 (10,67). El valor de 12 es alcanzado para niveles de ruido del 16 %, y un valor de 13 es alcanzado para un ruido del 18 %. Además, se observa que la varianza del número de prototipos es menor que 1 hasta que se alcanzan niveles de ruido del 12 %, mostrando que, en las diferentes ejecuciones, el número de prototipos se mantiene similar, así como el éxito de clasificación.

5.5. Conclusiones

El algoritmo ENNC es una aproximación evolutiva que permite encontrar un conjunto de prototipos capaz de clasificar correctamente conjuntos de datos, siguiendo una aproximación basada en el prototipo más cercano (*1-nearest prototype*). Las principales ventajas de este método son, por un lado, su capacidad de alcanzar altos porcentajes de éxito de clasificación en los dominios en los que ha sido probado; y por otro lado, su capacidad de alcanzar estos buenos resultados sin que el usuario tenga que definir condiciones iniciales para el aprendizaje.

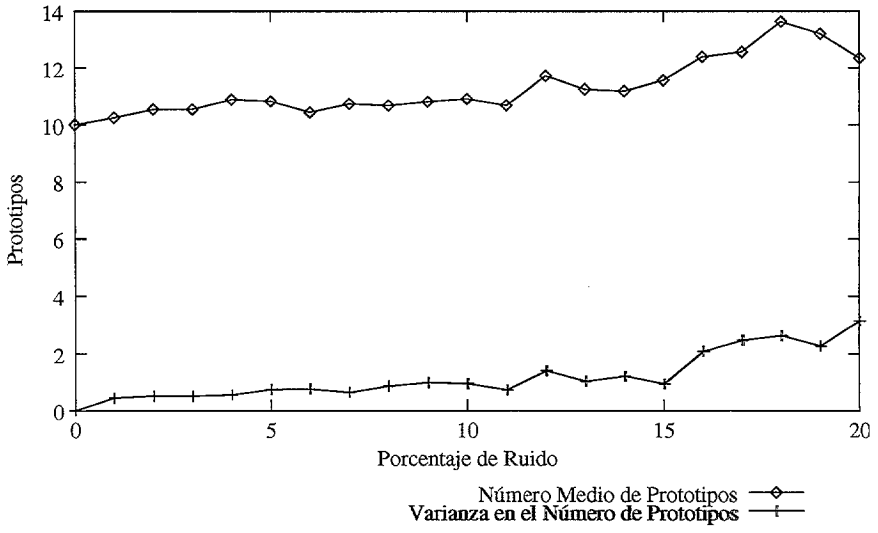


Figura 5.24: Evolución del número de prototipos obtenidos por ENNC según el porcentaje de ruído en el dominio del visor de LED.

Para alcanzar estos objetivos se ha diseñado un método desarrollado a partir de elementos de otros trabajos, como heurísticas para introducir nuevos prototipos, eliminar otros, fases de reetiquetado, etc. Otra diferencia con otros trabajos es que todas estas heurísticas están completamente integradas. La mayoría de los trabajos que se encuentran en la literatura y que, por ejemplo, permiten definir de forma automática el número de prototipos a utilizar o su posición inicial, lo hacen utilizando por un lado un método que les permite definir estos valores, y por otro lado, la técnica que realiza el aprendizaje de las posiciones definitivas de los prototipos iniciales. Sin embargo, el algoritmo ENNC no diferencia estas dos partes, sino que ambas están integradas.

Se han mostrado diversos experimentos y comparativas con otros métodos, mostrando que la aproximación planteada obtiene muy buenos resultados en la mayoría de los dominios tratados, sin que haya que definir prácticamente condiciones iniciales: el usuario sólo debe definir los conjuntos de entrenamiento, de test, y la condición de fin. Encontrar una técnica que obtenga la mejor solución en todos los dominios no es fácil, pero el algoritmo ENNC ha mostrado que obtiene resultados muy cercanos a los mejores en la mayoría de ellos, convirtiéndose en un candidato fácil de aplicar y utilizar en dominios desconocidos.

La condición de fin utilizada en los experimentos viene dado por un núme-

ro máximo de iteraciones. Se han mostrado otras condiciones de fin que pueden ser útiles. No obstante, los experimentos han mostrado que aunque el algoritmo no siempre converge a la misma solución, lo hace a un conjunto de soluciones muy similares. Cabe destacar que el objetivo del algoritmo no es encontrar una solución óptima, que en la mayoría de los casos puede no existir, sino una solución útil para el problema que se ha planteado, es decir, una solución con un alto éxito de clasificación y con un número de prototipos reducido. El algoritmo no garantiza la optimalidad de los valores encontrados, pero se ha mostrado empíricamente que suelen estar muy cerca.

El último experimento en el dominio del visor de LED ha mostrado que el algoritmo funciona también bien en dominios ruidosos, donde las aproximaciones de vecino más cercano no suelen recomendarse. Se ha mostrado que el algoritmo encuentra soluciones muy satisfactorias, y que no aparecen problemas de sobre-ajuste.

No obstante, este trabajo plantea algunos trabajos futuros en esta línea de investigación. Uno de ellos está orientado al concepto de similitud de los datos, es decir, a la medida de distancia utilizada por la regla del vecino más cercano. Una primera aproximación pasa por el uso de medidas de distancia que sean capaces de ponderar de forma adecuada los distintos atributos de los datos, permitiendo una normalización automática de dichos valores.

En lo que se refiere a la aplicación del algoritmo ENNC al problema de aprendizaje por refuerzo, se puede decir que se han cumplido los requisitos planteados al principio. Esos requisitos eran, principalmente, que se obtuvieran buenos resultados de clasificación sin que el número de prototipos fuese predefinido, y sin que hubiera que introducir una discretización inicial. Además, se ha mostrado que el clasificador es capaz de absorber una buena cantidad de ruido en los dominios, lo que, como se mostrará en el siguiente capítulo, permitirá la aplicación del modelo desarrollado también en dominios estocásticos.

Capítulo 6

El Algoritmo ENNC-QL

En el capítulo 4 se ha descrito el algoritmo VQQL, un método de aprendizaje por refuerzo basado en una discretización no supervisada del espacio de estados. Esa discretización no supervisada proporcionaba una forma sencilla de reducir drásticamente el tamaño del espacio de estados, permitiendo un mayor aprovechamiento de la experiencia obtenida en la interacción del agente con el entorno. Esto permitía, a su vez, utilizar el modelo en espacios de más dimensiones, ya que el incremento de la dimensión del problema, si bien requería espacios de estados más grandes, se mantenía en niveles aceptables. No obstante, en la sección 4.5 se introducía el problema de la pérdida de la propiedad de Markov, y cómo el modelo VQQL podría verse afectado por la introducción de indeterminismo derivada.

En este capítulo, se desarrolla el algoritmo ENNC-QL, que se plantea como un método de aprendizaje por refuerzo basado en la discretización del espacio de estados que evita la pérdida de la propiedad de Markov, y, por tanto, la pérdida de determinismo. Este método, no obstante, está muy relacionado con otros métodos basados en aproximación supervisada de la función de valor-acción, por lo que se podría decir que es un modelo mixto.

Se considera que el algoritmo ENNC-QL consta de dos fases fundamentales. En la primera de ellas, se utiliza el algoritmo ENNC como un aproximador de la función de valor-acción, dado que proporciona un conjunto de prototipos clasificados con el valor de dicha función. No obstante, dado que se aproxima la función de valor-acción $Q(s, a)$, se requieren tantos aproximadores como acciones, de forma que cada uno de ellos aproxima $Q(s, a_i)$, o lo que es lo mismo, $\hat{Q}_{a_i}(s)$, tal y como se introdujo en la sección 2.4.1.

En la segunda fase, se parte de los conjuntos de prototipos obtenidos por ENNC en la fase anterior, y se consideran como prototipos no etiquetados que discretizan el espacio de estados, y a partir de los cuales se puede aprender una tabla de valor-acción, de modo similar a como se hace con VQQL.

No obstante, hay dos diferencias fundamentales. La primera de ellas es que ahora existen tantas representaciones del espacio de estados como acciones, dado que en la primera fase teníamos también tantos aproximadores como acciones. La segunda, es que las clases obtenidas en la primera fase se usan para inicializar la tabla Q de la segunda fase.

La discretización resultado de la primera fase puede ser indeterminista, independientemente de cómo era el espacio de estados original. Si el dominio inicial era estocástico, lo va a seguir siendo. Si era determinista, dado que en el proceso de aprendizaje es realizado a partir de ejemplos, pueden haberse introducido errores, por lo que se considera que se ha introducido algo de indeterminismo con la nueva representación.

En la siguiente sección se plantea el marco en el que se pretende que el algoritmo pueda ser utilizado, definiendo las situaciones en las que su aplicación puede ser más provechosa. Posteriormente se realiza la descripción detallada del algoritmo.

6.1. Marco

Se ha introducido anteriormente que el algoritmo ENNC-QL tiene como objetivo minimizar los problemas introducidos por la pérdida de la propiedad de Markov. Como ya se planteó en la sección 4.5, la pérdida de dicha propiedad tiene como efecto inmediato la introducción de indeterminismo en el problema, y como consecuencia, que las funciones de valor no pueden ser computadas correctamente, introduciendo errores en la política de acción calculada a partir de ellas.

El efecto más indeseado de todo esto es que la política de acción puede introducir ciclos que llevan al agente, una y otra vez, a la misma región del espacio de estados. Este problema puede ser resuelto fácilmente en algunos casos, sobre todo cuando estos ciclos son pequeños, o de incluso una única acción, como por ejemplo, cuando un robot se queda bloqueado ante un obstáculo. En estos casos, se podría incluir algún sistema de detección de ciclos que hiciera que el robot saliera del ciclo en algún momento. Sin embargo, en otras ocasiones, estos ciclos pueden tener un tamaño muy considerable, como se mostrará en el siguiente capítulo para el problema de *Car on the Hill*, y por tanto, pueden ser difícilmente detectables. En general, estos ciclos normalmente se producen porque dentro de una región existen dos zonas para las que la política de acción debería ser distinta, y para las que, sin embargo, la política es la misma, dado que se encuentran en la misma región. Cuanto más crítico para resolver el problema sea distinguir esas dos zonas, más necesario es que el método a aplicar sea capaz de distinguirlas.

Por tanto el modelo que se presenta en este capítulo será más ventajoso en este tipo de problemas.

A efectos del modelo que se plantea, se puede suponer que el espacio de estados original es determinista. Esta suposición se realiza en función de dos factores. Si el dominio en verdad es determinista, la suposición es válida. Sin embargo, si el dominio no es determinista, sino estocástico, puede plantearse que la aleatoriedad del sistema es una componente de ruido sobre un comportamiento determinista del agente. Es decir, a los efectos de aprender una política de acción, se puede decir que un dominio estocástico es un dominio determinista con ruido. Lo difícil, en este caso, es saber diferenciar qué componente del comportamiento es la determinista, y cuál la ruidosa. Este problema ha sido tratado extensamente en clasificación supervisada con datos ruidosos, donde hay que discernir cuáles de los datos responden a la realidad, y cuáles al ruido. Esta es una de las capacidades del algoritmo ENNC mostrado en el capítulo anterior, que ha sido aplicado satisfactoriamente sobre dominios de clasificación ruidosos, como el del visor LED mostrado en la sección 5.4.6. El algoritmo ENNC-QL se basa en buena medida en esta propiedad, sobre todo en su aplicación en dominios estocásticos.

El considerar el dominio determinista, puede ofrecer otra ventaja, y es que si se supone el tiempo discreto y el conjunto de señales de refuerzo que se reciben del entorno finito, dado que el valor de γ es constante, el conjunto de los posibles valores que puede tomar la función de valor es también finito. Esas dos suposiciones, que en principio pueden considerarse como muy fuertes, no lo son tanto si se analizan la mayoría de los problemas de aprendizaje por refuerzo. En dichos problemas, normalmente sólo existen dos o tres señales de refuerzo distintas, que suelen corresponder con un refuerzo positivo, otro refuerzo nulo, y en algunas ocasiones, un refuerzo negativo (típicamente el conjunto $\{r_{min}, 0, r_{max}\}$). Además, el tiempo suele ser discreto, ya que se utiliza como unidad el coste de las acciones, que generalmente toma o puede tomar valor 1.

Por tanto, si suponemos que el problema que estamos tratando cumple estas dos restricciones, o que podemos redefinir el problema para que las cumpla, tenemos un problema de aprendizaje por refuerzo donde las funciones de valor pueden tomar sólo un número finito de clases, representado por la serie $\gamma^i r_{max}$, para $i = 0, \ldots, max\_camino$, donde max_camino representa el camino más largo para llegar a la meta y donde, por simplificación, suponemos sólo refuerzos positivos y nulos. Esto permitirá aplicar un método de aprendizaje supervisado con un conjunto de clases finitas, lo que permite discriminar de forma natural entre unas regiones y otras del espacio de estados original, simplemente verificando la clase a la que pertenecen los estados que contiene.

Por último, cabe destacar que al igual que para todos los métodos basados en representaciones tabulares de la función de valor acción, el conjunto de acciones que se considera es finito, y por cuestiones de capacidad de generalización en el espacio de estados, pequeño. En problemas donde esta propiedad no se cumpla, se puede buscar la utilización de macro-acciones que recojan todas las posibilidades de comportamiento del agente de una forma simplificada, tal y como se introdujo en la sección 2.4.5.

6.2. Descripción del Algoritmo

El problema de aprendizaje con el algoritmo ENNC-QL se define como sigue. Dado un dominio con un conjunto de espacios continuo, donde un agente puede ejecutar un conjunto discreto de L acciones, $A = \{a_1, \ldots, a_L\}$, el objetivo es obtener una aproximación de la función de valor-acción, $Q(s, a)$.

Como se ha introducido anteriormente, el algoritmo consta de dos fases fundamentales. En la primera de ellas, se plantea el problema de obtener de forma supervisada una aproximación de la función de valor-acción. En este caso, y siguiendo las ideas introducidas en la sección 2.4.1, se necesita un aproximador de función para cada acción, de forma que represente la utilidad de ejecutar la acción para cada estado. Es decir, se necesitan L aproximadores de la función de valor-acción, $\hat{Q}_{a_i}(s)$, para $i = 1, \ldots, L$. Por tanto, esta aproximación no es útil cuando el número de acciones es muy alto, o el espacio de acciones es continuo, tal y como se introdujo en el apartado anterior.

Como aproximador de funciones supervisado se utiliza el algoritmo ENNC. Si se consideran cubiertas todas las restricciones planteadas en la sección anterior, la aplicación del algoritmo como aproximador de la función de valor-acción es directa. El utilizar el algoritmo ENNC tiene varias ventajas. En primer lugar, están las ventajas derivadas directamente del algoritmo ENNC introducidas en el capítulo anterior, es decir, el escaso número de parámetros definidos por el usuario, y su alto éxito de clasificación.

Por otro lado, el utilizar una aproximación del prototipo más cercano aporta una ventaja más, que consiste en que estos algoritmos generan como salida un conjunto de prototipos que, liberados de la etiqueta de clase, pueden ser utilizados como cuantificadores de Voronoi, al igual que se utilizó con el modelo VQQL, y que se denotan por $S_{a_i}(s)$, para $i = 1, \ldots, L$. Por tanto, Una de las principales aportaciones de utilizar una aproximación de vecino más cercano consiste en que dado que se utilizan L aproximadores de función $\hat{Q}_{a_i}(s)$, también se calculan L representaciones del espacio de estados distintas, $S_{a_i}(s)$. Esta aproximación es novedosa desde el punto de vista de

los métodos basados en discretizaciones, que suelen utilizar una única discretización del espacio de estados para todas las acciones. No obstante, puede ser interesante desde el punto de vista de que cada acción puede necesitar distintos recursos para poder ser representada correctamente. Por tanto, cada $S_{a_i}(s)$ es una función que toma como entrada un estado de la representación original, y genera como salida un estado de la nueva representación. El obtener esta discretización del espacio de estados puede ser muy ventajosa, como se mostrará posteriormente.

El algoritmo completo consta de varios pasos, tal y como se muestra en la Figura 6.1. El algoritmo comienza con una inicialización de las L discretizaciones, $S_{a_i}(s)$, a un mismo y único estado, y de los aproximadores, $\hat{Q}_{a_i}(s)$, al valor de refuerzo nulo, típicamente 0.

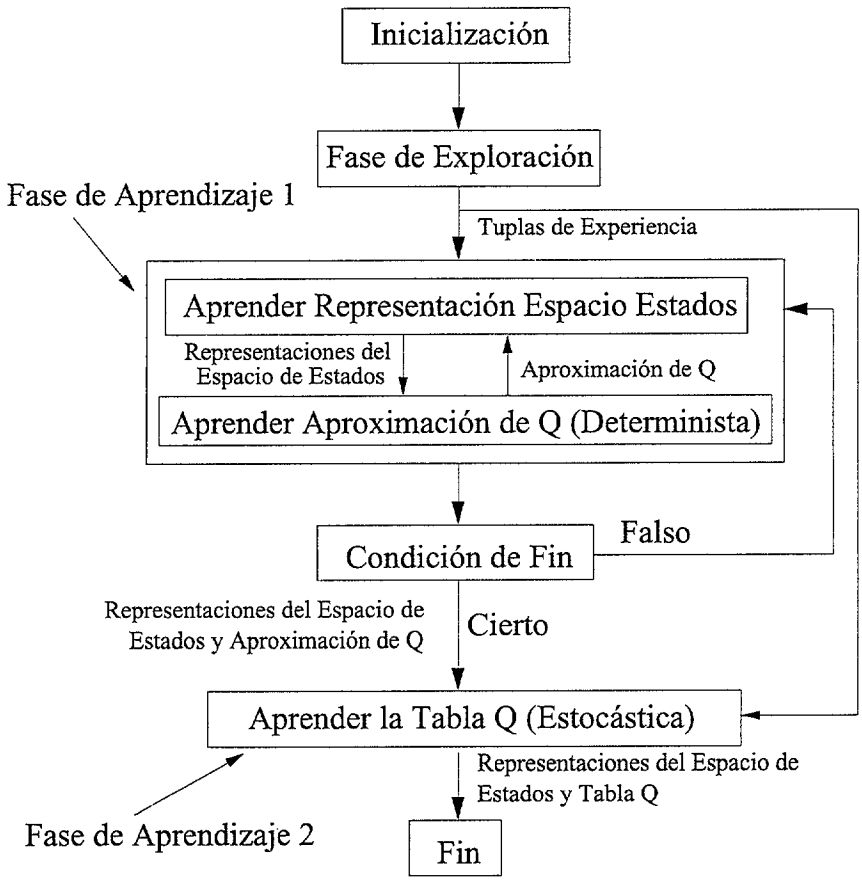


Figura 6.1: Descripción a Alto Nivel del Algoritmo ENNC-QL.

Tras esta inicialización, se produce una fase de exploración del dominio.

En esta exploración se plantea el principal sesgo del algoritmo, dado que si la relación entre las veces que se llega a meta, y por tanto, se recibe un refuerzo positivo, y las veces que no se llega a la meta, y por tanto, se recibe un refuerzo nulo, está muy desequilibrado hacia el segundo, el clasificador, en la primera iteración, puede quedarse en un mínimo local consistente en clasificar todo el entorno como con un refuerzo nulo. Por tanto, tras esa fase se obtendría de nuevo un aproximador que daría un valor nulo para todo el dominio, por lo que no se aprendería nada con respecto a la situación inicial.

No obstante, el cómo realizar esta exploración no se ha considerado como un punto esencial en esta tesis doctoral, mas que en la parte de experimentación, si bien abre una interesante línea de investigación futura. En la bibliografía se pueden encontrar muchos métodos adecuados para realizar esta exploración de la forma más completa posible [Smart, 2002]. Esta fase genera un conjunto de T tuplas de experiencia, del tipo $< s, a_i, s', r >$, donde s es cualquier estado, a_i la acción que se ejecuta desde ese estado, s' es el estado alcanzado al ejecutar la acción a_i desde el estado s, y r es el refuerzo inmediato recibido desde el entorno.

Tras la exploración se comienza la primera fase del aprendizaje. Dicha fase es un proceso iterativo donde los aproximadores $\hat{Q}_{a_i}(s)$ se aprenden a la vez que los $S_{a_i}(s)$, ya que cada $S_{a_i}(s)$ está constituido por los prototipos de $\hat{Q}_{a_i}(s)$, pero sin las etiquetas de clase. Por tanto, al final de esta fase del algoritmo se dispone de las L representaciones del espacio de estados, así como la aproximaciones $\hat{Q}_{a_i}(s)$ para cada acción. Este aprendizaje se realiza tal y como muestra la descripción a bajo nivel del algoritmo, mostrada en la Figura 6.2.

A partir del conjunto inicial de tuplas, y utilizando los aproximadores $\hat{Q}_{a_i}$, $i = 1, \ldots, L$ generados en la iteración anterior, se puede aplicar la función de actualización de *Q-Learning* para dominios deterministas para obtener L conjuntos de entrenamiento, T_i^0, $i = 1, \ldots, L$, con entradas del tipo $< s, c_{s,a_i} >$, donde c_{s,a_i} es el valor resultante de aplicar la función de actualización de *Q-Learning* para cada tupla obtenida en la exploración. En la primera iteración, y por la inicialización realizada, $\hat{Q}_{a_i}(s) = 0$, $i = 1, \ldots, L$, para todo s, por lo que los posibles valores para c_{s,a_i} se limitan al valor nulo y al de máximo refuerzo, r_{max}, que se habrá obtenido en aquellas experiencias que tengan como estado final la meta. No obstante, en la siguiente iteración ya habrá estados s para los cuales algún $\hat{Q}_{a_i}(s)$ sea r_{max}, y por tanto se generarán ejemplos del tipo $< s, \gamma^1 r_{max} >$. Y así sucesivamente, hasta el momento en que se tendrán ya ejemplos de tipo $< s, \gamma^t r_{max} >$, para $t = 1, \ldots, k$ (donde k es la longitud, en número de acciones, del camino más largo para llegar a la meta). Otra propiedad importante es que, en cada iteración, las aproximaciones de Q y/o las discretizaciones no cambian demasiado con respecto a la iteración

Algoritmo ENNC-QL

- **Inicialización**
  - t
  - L s_0 s_0 s a_i $S^0_{a_i}(s)$ s_0 $\tilde{Q}^0_{a_i}(s)$
- **Fase de exploración** T $< s, a_i, s', r >$
- **Fase de aprendizaje 1** k
  - a_i $\tilde{Q}^{t+1}_{a_i}(s)$ $S^{t+1}_{a_i}(s)$
    - T^t_i $< s, a_i, s', r >$ T
      - c_{s,a_i} s a_i

        $$c_{s,a_i} \leftarrow r + \gamma \max_{a_j \in A} \tilde{Q}^t_{a_j}(s')$$

      - $< s, c_{s,a_i} >$ T^t_i
    - T^t_i $\tilde{Q}^t_{a_i}(s)$ $\tilde{Q}^{t+1}_{a_i}(s)$ $S^{t+1}_{a_i}(s)$ s
  - $t \leftarrow t + 1$
- **Fase de aprendizaje 2** ENNC
  - $\tilde{Q}(s, a)$ $S^t_{a_i}(s)$ $i = 1, \ldots, L$
  - i $\tilde{Q}(s, a)$ $\tilde{Q}^t_{a_i}(s)$
  - *Q-Learning* $\tilde{Q}(s, a)$ $S^t_{a_i}(s)$

Figura 6.2: Algoritmo ENNC-QL.

anterior, por lo que pueden utilizarse como semilla las aproximaciones obtenidas en la iteración anterior, con un tiempo de aprendizaje menor que el que requeriría un aprendizaje completo. En esta primera fase, por tanto, la arquitectura del algoritmo es la planteada en la Figura 6.3.

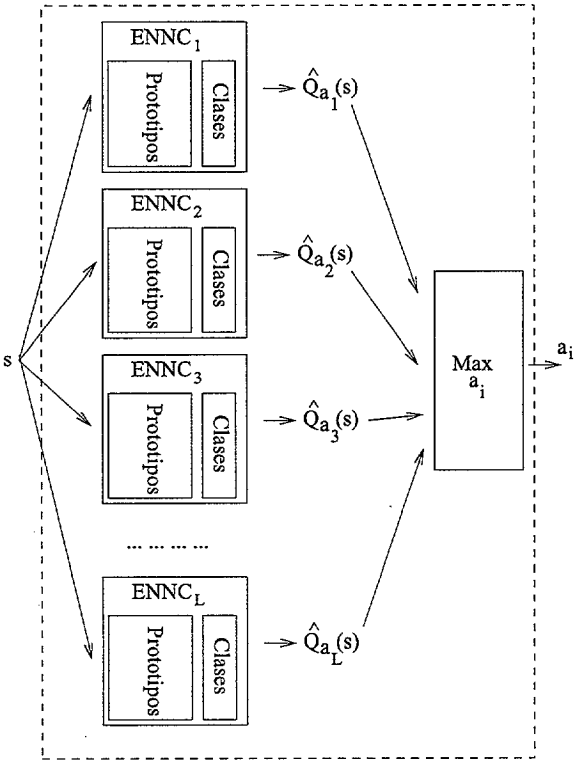


Figura 6.3: Uso de ENNC como aproximador de funciones en ENNC-QL.

Esta arquitectura es similar a la planteada en la Figura 2.11, en la que se utilizaba una red neuronal como aproximador. El modo de aprendizaje en esta primera fase es prácticamente equivalente al del algoritmo *Iterative Smooth Q-Learning*, planteado en la Figura 2.13. En este caso, el aproximador es el algoritmo ENNC, y como salida se genera no sólo la aproximación de la función Q, sino también las discretizaciones del espacio de estados para cada acción.

En este momento, por tanto, se dispone ya de una buena aproximación de la función de valor acción, representada por $\hat{Q}_{a_i}(s)$, y como se ha introducido anteriormente, también se dispone de las discretizaciones del espacio de estado, $S_{a_i}(s)$. Estas discretizaciones tienen una propiedad muy importante, si se supone que el dominio es determinista, y que el clasificador es perfecto (es

capaz de delimitar perfectamente las distintas clases) la nueva representación del espacio de estados mantiene la propiedad de Markov, y por tanto, no ha introducido indeterminismo. Esto es debido a que, en cada iteración, el algoritmo propaga el refuerzo positivo desde la meta hacia el resto del dominio. De esta forma, se puede decir que el parámetro k o número de iteraciones que se deben realizar viene dado por el camino más largo desde cualquier punto a la meta. Desde el punto de vista práctico, se puede decir que cuando ningún punto del entorno tenga un valor de Q nulo, es que se ha terminado esta fase (si se supone que la meta es alcanzable desde cualquier posición inicial del entorno).

Sin embargo, ninguno de los dos requisitos planteados anteriormente tienen por qué ser ciertos. El dominio de aplicación, unas veces será determinista, y otras veces estocástico. Y es probable que nunca se encuentre el clasificador que discrimine perfectamente todas las clases, simplemente porque se parte de un conjunto más o menos reducido de ejemplos. Por tanto, la nueva representación del espacio de estados obtenida puede incluir errores, y por tanto, también el aproximador de la función de valor-acción construido sobre esas discretizaciones.

Esto motiva la segunda fase de aprendizaje del algoritmo, en la que se plantea utilizar las discretizaciones obtenidas como tales, sin tener en cuenta los aproximadores, y aprender la tabla $Q(s,a)$ tal y como se hacía con el modelo $VQQL$, y como queda reflejado en la Figura 6.4.

La principal diferencia con aquel modelo, es que ahora cada acción tiene su propia discretización del espacio de estados original, y que la construcción de las regiones ha sido supervisada, y se considera que mantiene en gran medida la propiedad de Markov. Además, la aproximación de la función $Q(s,a)$ que se disponía sirve para inicializar la nueva tabla Q, por lo que tienen ya valores que pueden ser muy similares a los finales, dependiendo del nivel de indeterminismo de las discretizaciones obtenidas. Si estas discretizaciones dieran lugar a un dominio determinista, aplicar esta segunda fase no tendría sentido, puesto que serían exactamente los mismos que fueron obtenidos en la fase anterior. Según aumente el indeterminismo del dominio en la nueva discretización, más diferentes pueden ser las aproximaciones de la función Q obtenida en la primera fase de aprendizaje de la obtenida en la segunda.

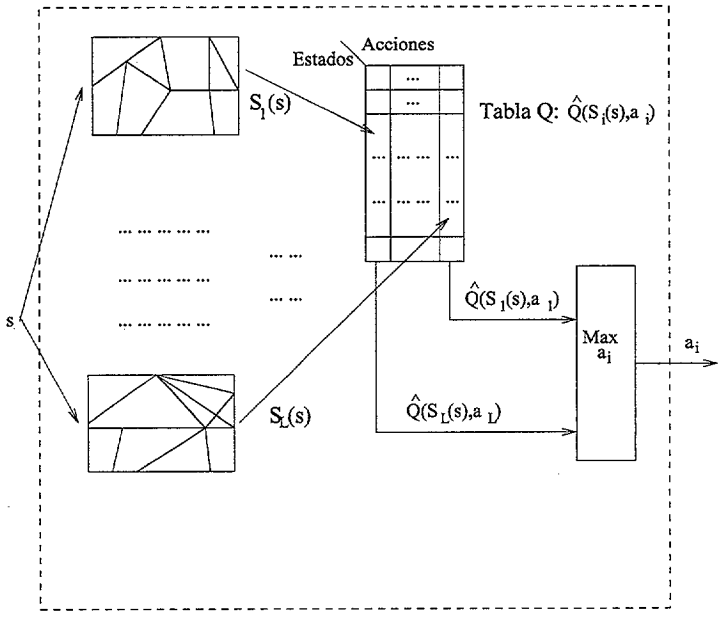


Figura 6.4: Arquitectura del algoritmo ENNC-QL en la segunda fase de aprendizaje.

Capítulo 7

Experimentos y Resultados

A lo largo de los capítulos anteriores se han descrito dos métodos de aprendizaje por refuerzo. Por un lado, en el capítulo 4 se ha descrito el modelo VQQL, basado en la discretización no supervisada del espacio de estados, lo que permite una implementación tabular de las funciones de valor. En dicho capítulo se introdujo parte de la experimentación realizada con este modelo en dos dominios, el de aprendizaje de habilidades en la *RoboCup*, y el de *CMOMMT*.

Por otro lado, en el capítulo 6, se ha planteado otro modelo basado en dos fases, una basada en el uso de un clasificador supervisado para aproximar las funciones de valor y generar una discretización del espacio de estados, que son utilizados en la segunda fase para obtener, nuevamente, una representación tabular de las funciones de valor. En este capítulo se pretende verificar la utilidad de este algoritmo, presentando las ventajas e inconvenientes que tiene, y mostrando comparativas con otros métodos.

La experimentación ha sido dividida por dominios de aplicación. En la sección 7.1 se muestra la experimentación realizada en el dominio *Car on the Hill*, descrito en la sección 2.7.2. Sobre este dominio se van a realizar diversos experimentos, tanto del modelo VQQL, como de ENNC-QL, que serán contrastados con los resultados obtenidos con otros métodos. En concreto se van a realizar comparativas con el algoritmo *Iterative Smooth Q-Learning* planteado en la sección 2.4.2. En este caso, el aproximador de la función de valor utilizado es C4.5 [Quinlan, 1993], en la versión implementada e integrada en la herramienta WEKA [Witten and Frank, 2000]. Este dominio presenta una especial dificultad para la aplicación del algoritimo ENNC-QL, ya que éste está pensado para dominios con un conjunto de valores de la función de refuerzo finito, y el dominio *Car on the Hill* presenta una función de refuerzo continua.

En la sección 7.2, la experimentación se realiza en el dominio de navega-

ción por oficinas que se definió en la sección 2.7.1. En este dominio se van a realizar experimentos para los mismos métodos de aprendizaje que en el experimento anterior. El principal objetivo de este experimento es verificar el comportamiento de ENNC-QL en dominios indeterministas. Para ello, se plantearán distintos experimentos para el mismo dominio, pero con distintos niveles de ruido, con el fin de verificar las diferencias de comportamiento de los algoritmos partiendo del caso determinista, hasta casos con una alta componente ruidosa.

La sección 7.3 tiene como objetivo verificar el comportamiento de ENNC-QL en dominios de más dimensiones. Para ello, se aplica el algoritmo sobre el dominio *CMOMMT* definido en la sección 2.7.3. Para realizar comparativas sobre este dominio se parte de la experimentación previa descrita en la sección 4.4.2 con el algoritmo *Pessimistic Lazy Q-Learning* y con VQQL. Por tanto, se plantea el mismo escenario de aprendizaje que en aquella ocasión, lo que permite verificar el comportamiento de ENNC-QL ante espacios de estados de hasta 6 dimensiones. Además, este dominio posee ciertas características que lo hacen poco apropiado para ser afrontado por ENNC-QL. Entre estas características cabe destacar que, tal y como fue definido en la sección 4.4.2, no tiene zona de meta, es decir, una situación donde un refuerzo positivo pueda ser aplicado como fin a una secuencia de acciones, y a partir del cuál los refuerzos deban ser propagados al resto del dominio. Por tanto, los refuerzos positivos son recibidos en escasas ocasiones, por lo que salir del mínimo local de asignar a todo el dominio un refuerzo nulo puede ser complicado para cualquier método basado en la aproximación supervisada de la función de valor, tal y como ocurre en la primera fase de ENNC-QL. Por último, en la sección 7.4 se muestran las principales conclusiones de toda esta experimentación.

7.1. *Car on the Hill*

El primer dominio donde se ha realizado una extensa experimentación, ha sido el dominio *Car on the Hill*, introducido en la sección 2.7.2. El objetivo de este dominio es hacer llegar al coche a la cima de la montaña ($x >= 1$). Además, se desea que la velocidad de llegada (v) sea lo más cercana posible a 0.

Los datos de entrenamiento de obtienen a partir de 4.000 intentos. En cada intento, el coche trata de alcanzar la cima de la montaña desde una posición aleatoria del entorno. La forma de realizar la exploración es siguiendo una política de comportamiento aleatoria. Para mejorar este proceso de exploración, y generar un conjunto de experiencias lo más representativo po-

sible, se ha optado por limitar cada uno de estos intentos a un máximo de 10 acciones.

Los test se realizan también sobre 4.000 intentos de llegar a la meta desde posiciones iniciales aleatorias. En este caso, no obstante, cada intento tiene una longitud máxima de 150 acciones. Si tras las 150 acciones el robot no ha conseguido llegar a la meta, se supone que el robot está bloqueado o en un ciclo, y por tanto se considera un intento fallido.

En este dominio se han realizado varios experimentos, comparando el comportamiento del robot cuando ha aprendido utilizando discretizaciones uniformes y utilizando discretizaciones generadas con el algoritmo GLA, siguiendo el modelo VQQL. Para estos casos se han realizado varios experimentos utilizando discretizaciones de distintos tamaños. Estos resultados se han comparado con otros dos métodos que generan una solución única cada uno de ellos. Por un lado, se ha aplicado el algoritmo *Iterative Smooth Q-Learning* con C4.5 [Quinlan, 1993] como aproximador de la función de valor-acción, y por otro lado, ENNC-QL. A continuación, se describen cada uno de estos experimentos, mostrando las comparativas oportunas.

La aplicación del algoritmo ENNC-QL en el dominio de *Car on the Hill* requiere únicamente una consideración. En ENNC-QL, se asume un único refuerzo positivo al final de un episodio, cuando se llega a la meta. Al propagarse este refuerzo hacia el resto del entorno, tomando como cierto el requisito de un dominio determinista, y acciones ejecutadas en un tiempo discreto, se genera un conjunto de posibles valores de la función de valor discreto, tal y como se planteó en la sección 6.1. Sin embargo, en el dominio *Car on the Hill*, la función de refuerzo sólo es positiva cuando se llega a la zona de meta ($x = 1$), y su valor depende del resultado de otra nueva función. Ésta función es lineal con la velocidad, siendo máxima para una velocidad 0, y mínima para una velocidad máxima de 4, tal y como se describe en la ecuación 2.17. Para resolver esta diferencia, se ha optado por discretizar también la función de refuerzo a los posibles valores que se sabe "a priori" que la función de valor puede tomar, es decir, la sucesión $\gamma^i r_{max}$, para $i = 1, \ldots, \infty$. Además, para velocidades mayores que 2, el refuerzo es considerado también nulo. Una vez realizada esta discretización, la aplicación del algoritmo ENNC-QL es directa. Estas consideraciones en la función de refuerzo son mantenidas en toda la experimentación descrita a continuación.

Además, cabe destacar que se ha elegido un valor de $\gamma = 0{,}9$. Este valor es elegido alto para que se destaquen las diferencias en la función de valor en aquellas regiones del espacio de estados que están alejadas de la meta. Si se eligiera un valor de γ pequeño, los posibles valores de la función de valor para estas regiones serían todas muy cercanas a 0, y prácticamente indiferenciables entre unas y otras, dado que la sucesión $\{\gamma^i\}$ tiende a 0 más

rápido que otra $\{\gamma'^i\}$, para $\gamma < \gamma'$.

7.1.1. Discretizaciones Uniformes y VQQL

Tanto para discretizaciones uniformes, como para VQQL, se han utilizado representaciones del espacio de estados de distintos tamaños. Estas discretizaciones tienen como fin representar el espacio de estados continuo presente en *Car on the Hill.*

En el caso de VQQL, se ha seguido la versión por lotes definida en la Figura 4.9. El primer paso consiste en obtener el conjunto de ejemplos necesario para diseñar el cuantificador utilizando el algoritmo GLA. Una vez que se dispone de los cuantificadores, el proceso seguido para aprender la política con estas discretizaciones es equivalente al que se pueden seguir para el caso uniforme.

Para aprender la política, una vez que se disponía de las discretizaciones, se utilizó la función de actualización de *Q-Learning*, utilizando el valor de $\gamma = 0,9$, tal y como se introdujo anteriormente. Para mejorar los resultados obtenidos sin tener que incrementar el tamaño de la exploración, las tuplas de experiencia obtenidas en la exploración inicial son reutilizadas hasta 20 veces, en un proceso iterativo. Además, se parte de un valor inicial de α de 0,2, que se va decrementando en cada iteración en 0,01. Estos valores fueron obtenidos experimentalmente, y como se mostrará posteriormente, producen unos resultados aceptables, y que si bien pueden no ser óptimos, sí permiten realizar las comparativas oportunas.

La Figura 7.1 muestra el porcentaje medio de intentos exitosos para discretizaciones uniformes de 64, 256, 400, 676, 1024, 2116 y 3721 estados, que representan discretizaciones uniformes 8, 16, 20, 26, 32, 46 y 61 niveles por atributo. También se muestran los resultados para VQQL con los mismos tamaños.

En la figura se muestra que con ambos métodos se obtienen resultados de alrededor del 80 % de éxito, y para espacios de estados pequeños (64 estados). El mejor resultado lo obtiene la discretización uniforme de 400 estados, con un 80,65 % de éxito, si bien estas diferencias parecen poco significativas. Cabe destacar que estos resultados se consideran óptimos, ya que en este dominio existe una amplia región del espacio desde la que es imposible llegar a la meta, ya que independientemente de la política de acción que se siga, el coche llega hasta la posición $x = -1$, es decir, se sale del dominio por el lado opuesto al de la montaña. Esta zona es de aproximadamente un 20 % del dominio. El porcentaje de éxito decrece para espacios de estados muy grandes, debido a que se introducen de nuevo espacios de estados demasiado grandes que requerirían más experiencia para poder aproximar de forma adecuada la

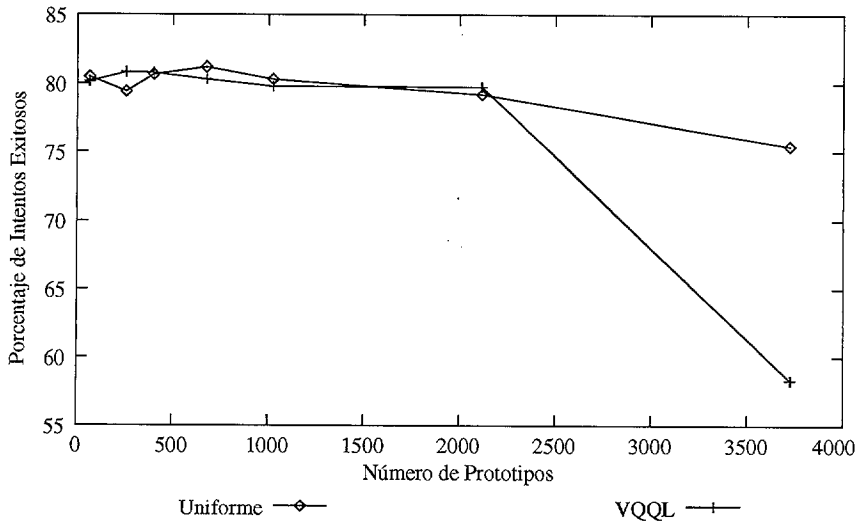


Figura 7.1: Porcentaje de intentos exitosos con discretizaciones uniformes y *VQQL*.

función Q.

Cabe destacar, por tanto, que en lo que a porcentaje de éxitos se refiere, ambos métodos obtienen muy buenos resultados incluso con espacios de estados pequeños. En lo que a la velocidad media con la que se llega a la meta, sin embargo, los resultados no son tan optimistas. La Figura 7.2 muestra la velocidad media y la varianza con la que se ha llegado a la zona de meta para el caso uniforme y VQQL.

En la figura se observa que la velocidad media de llegada a meta inferior a uno sólo es alcanzada cuando el número de prototipos es muy alto (2116), tanto para el caso uniforme como para VQQL. Esto muestra que, aunque con menos estados se obtienen políticas que permiten llegar a la meta en un porcentaje muy alto de intentos, la velocidad con la que se alcanza dicha meta es también, en media, muy alta, por lo que hay que elegir espacios de estados de mayor resolución. Esto muestra, por tanto, la dificultad que se puede encontrar en muchos dominios para decidir el tamaño adecuado del espacio de estados.

Por último, la Figura 7.3 muestra el tiempo medio de llegada a meta para ambos modelos, medido en número de acciones ejecutadas. Por cada uno de los métodos, se muestra el número medio de acciones teniendo en cuenta sólo los intentos que en los que se ha llegado a meta, y también el mismo número, pero penalizando en 100 acciones aquellos intentos en los que no se

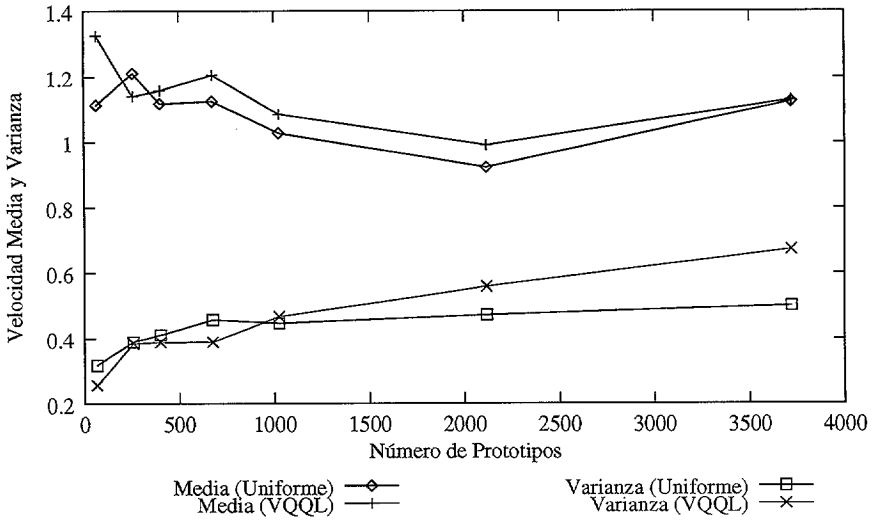


Figura 7.2: Velocidad de llegada a meta con discretizaciones uniformes y *VQQL*.

ha conseguido llegar a la meta. La figura muestra que para discretizaciones de tamaño reducido, el tiempo medio en llegar a la meta es alto, y que este valor sólo es menor para discretizaciones de 400, 676 y 1024 estados. La figura también muestra que para 2116 estados, valor para el que se consigue llegar a meta con la velocidad media más pequeña, el número de acciones medio en llegar a dicha meta es también más alto. Es decir, para 2116 estados realiza mejor la tarea, aunque tarda más tiempo.

La razón por la que que *VQQL* no obtiene mejores resultados sobre la discretización uniforme se comprende fácilmente si se observa la figura 7.4. En dicha figura se muestran 10000 de los ejemplos de estados por los que pasa el coche tras el proceso exploratorio utilizado en el modelo VQQL, y necesario para obtener las discretizaciones. En dicha figura se muestra que todo el dominio parece importante, y que apenas hay zonas donde una discretización uniforme esté en desventaja con la discretización aprendida con el algoritmo GLA.

La Figura 7.5 muestra los prototipos de las discretizaciones de tamaño 400 que se tienen, tanto para el caso uniforme, como para el caso de VQQL, esta última aprendida con el algoritmo GLA. En la figura se muestra cómo los prototipos obtenidos con GLA están más adaptados a los datos iniciales, aunque esta adaptación apenas produce importantes ventajas.

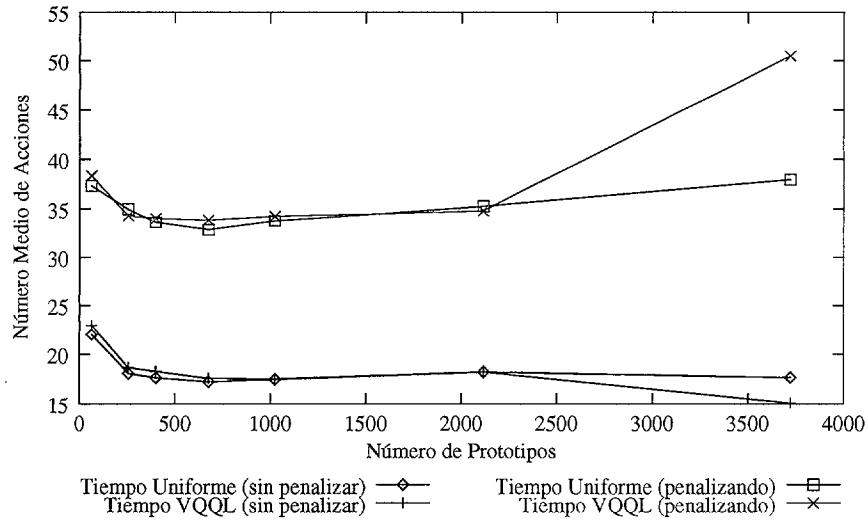


Figura 7.3: Tiempos de llegada a meta con discretizaciones uniformes y VQ-QL.

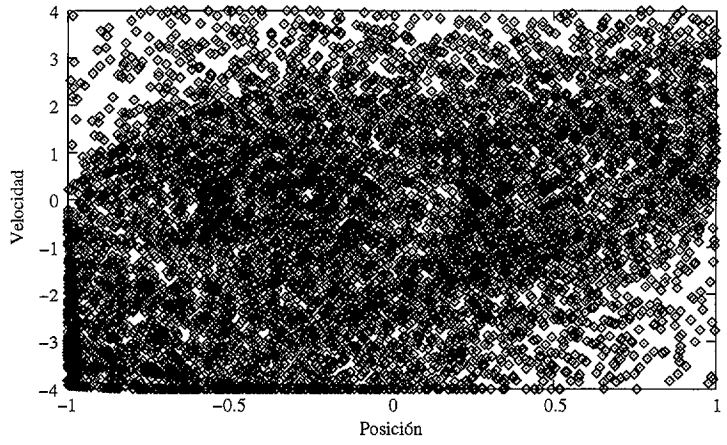
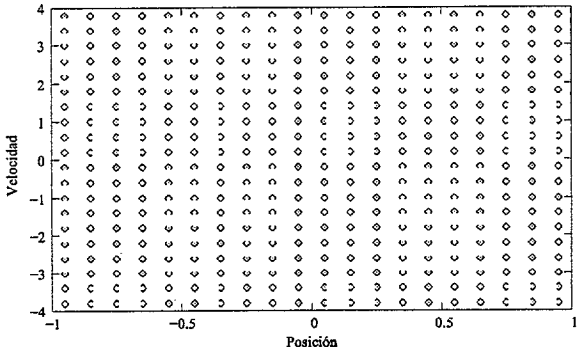
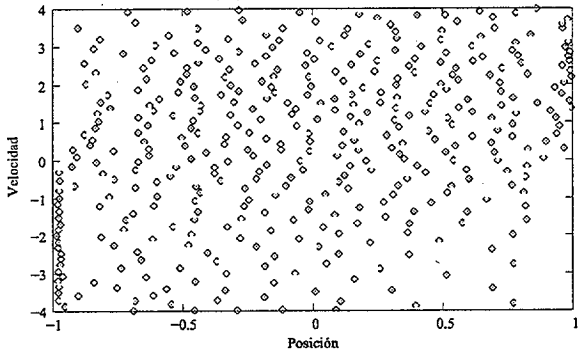


Figura 7.4: Ejemplos de estados en dominio *Car on the Hill*



(a) Uniforme



(b) GLA

Figura 7.5: Ejemplos de discretizaciones en dominio *Car on the Hill*

7.1.2. *Iterative Smooth Q-Learning* con C4.5

El dominio *Car on the Hill* ha sido también resuelto con el algoritmo *Iterative Smooth Q-Learning*, utilizando árboles de decisión, concretamente C4.5, como aproximador. Los parámetros de ejecución de C4.5 son los que la herramienta WEKA utiliza por omisión. El número de iteraciones ejecutadas es de 30, que es un valor que empíricamente demuestra ser suficiente para que los refuerzos sean propagados desde la meta hasta el resto del dominio.

El porcentaje de éxito alcanzado es del 59,5 %. Este porcentaje es sensiblemente inferior al de los casos anteriores, mostrando que C4.5 no es adecuado como aproximador en este dominio. El número de acciones medio necesario para llegar a la meta es de 13,9, valor inferior a los obtenidos con discretizaciones uniformes y VQQL. No obstante, si se penalizan los intentos en los que no se ha llegado a la meta, este valor asciende hasta un 48,77.

7.1.3. ENNC-QL

El algoritmo ENNC-QL también ha sido aplicado sobre el dominio *Car on the Hill*. Sin embargo, una diferencia fundamental de este algoritmo respecto a VQQL y al basado en discretizaciones uniformes es que el algoritmo ENNC-QL es capaz de definir automáticamente el tamaño de las discretizaciones a utilizar, por lo que no es necesario realizar la experimentación varias veces para distintas discretizaciones.

El proceso de entrenamiento se ha realizado utilizando también las tuplas de experiencia generadas por 4.000 intentos de llegar a la meta a partir de situaciones aleatorias, con una longitud máxima de 10 acciones por intento. Además, el entrenamiento se ha dividido en dos partes. Por una lado, está la fase de aprendizaje 1, consistente en utilizar el algoritmo ENNC como un aproximador de funciones supervisado, y extraer los resultados como tal, y que equivale al algoritmo *Iterative Smooth Q-Learning* con ENNC como aproximador. Por otro lado, se utiliza esa salida como una discretización del espacio de estados y se termina la ejecución del algoritmo ENNC-QL (fases 1 y 2).

En la Figura 7.6 se muestra la evolución del porcentaje de veces que el coche es capaz de llegar a la meta, en función del número de iteraciones ejecutadas en la primera fase de aprendizaje del algoritmo. Se observa que con la primera fase únicamente, el algoritmo es capaz de alcanzar porcentajes de hasta un 70 % de intentos exitosos, tras al menos 35 iteraciones. Esto muestra que utilizando ENNC como aproximador de la función Q, se obtienen buenos porcentajes de éxito, incluso mejores que los que se obtuvieron con C4.5. No obstante, si se ejecuta la segunda fase del algoritmo, se observa que se llega al 80 % de éxito en tan sólo 14 iteraciones, y que este porcentaje es mantenido a lo largo del tiempo.

Sin embargo, en este dominio no es sólo importante llegar a la zona de meta, sino hacerlo con una velocidad mínima. La Figura 7.7 muestra la evolución de la media y la varianza de este valor para los 4000 intentos ejecutados para realizar el test.

La gráfica muestra cómo la velocidad media de llegada es cercana al uno para ENNC-QL completo, y un poco superior para el caso en que sólo se ha ejecutado la primera fase. También se observa que la varianza es mucho menor en el primer caso. Por último, la Figura 7.8 muestra los tiempos medios de llegada a meta, teniendo en cuenta sólo los intentos exitosos, y teniendo en cuenta los fracasos, que se penalizan con un valor de 100. La figura muestra que se obtienen resultados similares a los alcanzados para la discretización de tamaño 2116, con VQQL y discretizaciones uniformes.

La Figura 7.9 muestra los prototipos que se generan en la iteración 30

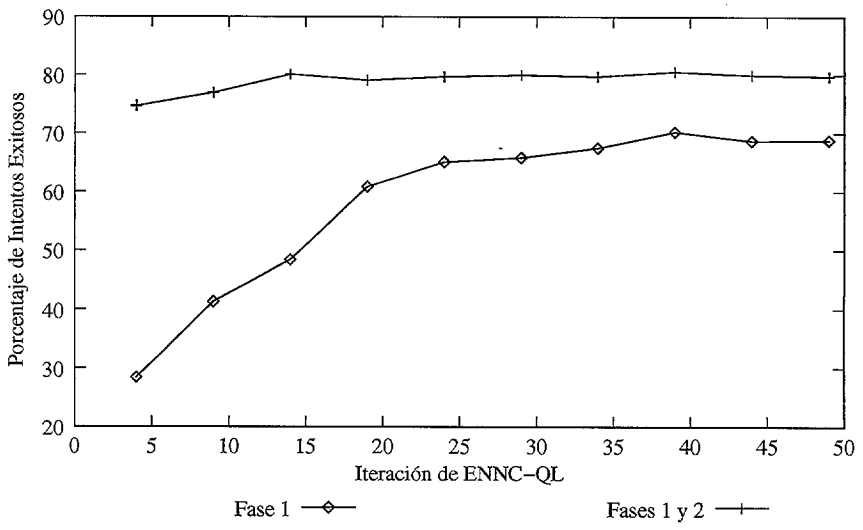


Figura 7.6: Porcentaje de intentos exitosos con ENNC-QL.

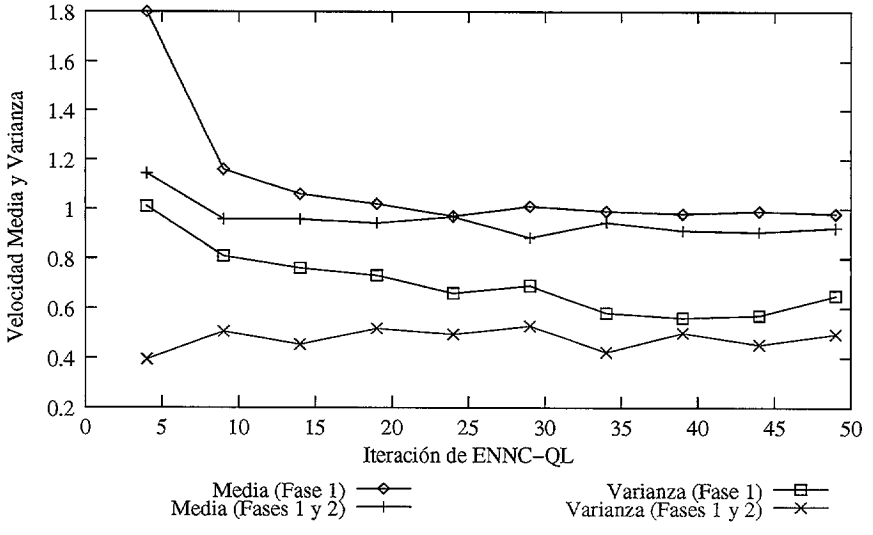


Figura 7.7: Velocidad de llegada a meta con ENNC-QL.

de la primera fase de aprendizaje del algoritmo ENNC-QL. En dicha figura se muestra que los prototipos son insertados en mayor o menor cantidad teniendo en cuenta las diferencias mayores o menores en la función de valor entre unas zonas y otras.

Para finalizar, la Tabla 7.1 especifica los resultados obtenidos con cada una de las aproximaciones, mostrando los mejores resultados obtenidos con

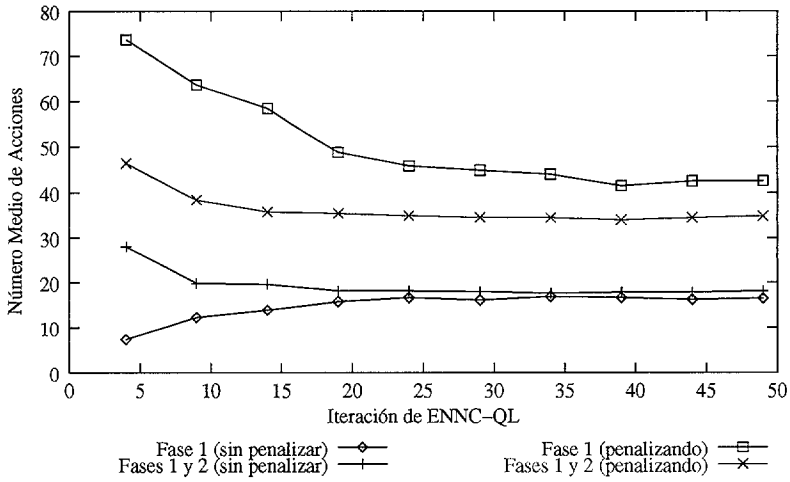


Figura 7.8: Tiempos de llegada a meta con ENNC-QL.

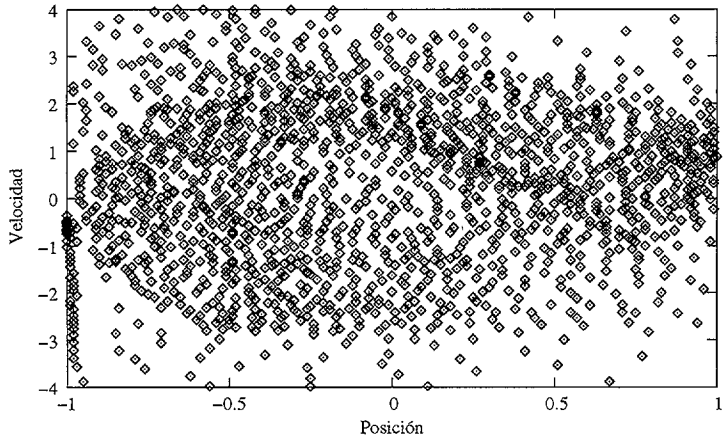


Figura 7.9: Prototipos generados por ENNC-QL en la iteración 30 en el dominio de *Car on the Hill*

discretizaciones uniformes y con VQQL en lo que se refiere a porcentaje de éxito en llegar a la cima de la montaña (definidos como *Uniforme 1* y *VQQL 1*), y velocidad media de llegada mínima (definidos como *Uniforme 2* y *VQQL 2*).

	Uniforme 1	Uniforme 2	VQQL 1	VQQL 2	C4.5	ENNC-QL
Éxito	81,175	79,175	80,82	79,75	59,5	80
Velocidad	1,12	0,92	1,14	0,99	1,08	0,88
Regiones	676	2116	256	2116	970	1572

Tabla 7.1: Comparativa de los distintos métodos en el dominio *Car on the Hill*.

7.2. Dominio de Navegación por Oficinas

El siguiente dominio en el que se ha experimentado es el dominio de navegación por un entorno de oficinas, definido en la sección 2.7.1. Tal y como se planteó entonces, este dominio permite introducir varios niveles de ruido en la percepción que el robot tiene de su posición en el entorno. Siguiendo esta aproximación, se han realizado varias comparativas teniendo en cuenta que la información sobre la localización de que se dispone tiene un error de un 0 %, de un 5 %, de un 10 % y de un 20 % respectivamente.

La experimentación realizada sobre este dominio es similar a la del dominio *Car on the Hill*, habiendo obtenido resultados para modelos de aprendizaje basados en discretizaciones uniformes de distintos tamaños, para VQ-QL, también con distintos tamaños, usando árboles de decisión (C4.5) como aproximador en *Iterative Smooth Q-Learning*, y con *ENNC-QL*.

Además, la mayoría de los parámetros de aprendizaje son también similares a los utilizados en el dominio de oficinas. Así, el parámetro de descuento se mantiene en $\gamma = 0{,}9$, el número de intentos de aprendizaje se mantiene en 4.000, con inicializaciones aleatorias en todo el entorno, e intentos de un máximo de 10 acciones. Los test también se han realizado sobre 4.000 intentos de llegar a la meta desde posiciones aleatorias en el dominio, si bien en este caso, se considera un intento fallido cuando el robot no ha sido capaz de llegar a la meta tras ejecutar 100 acciones. A continuación, se detallan los experimentos realizados con cada uno de los métodos.

7.2.1. Discretizaciones Uniformes

La Figura 7.10 muestra los prototipos utilizados para hacer una discretización uniforme de 1024 estados. Cada uno de estos prototipos define una región de Voronoi, tal y como se describió en la sección 4.1, y por tanto, se supone que para todos los estados comprendidos en cada una de esas regiones, las funciones de valor devuelven el mismo resultado.

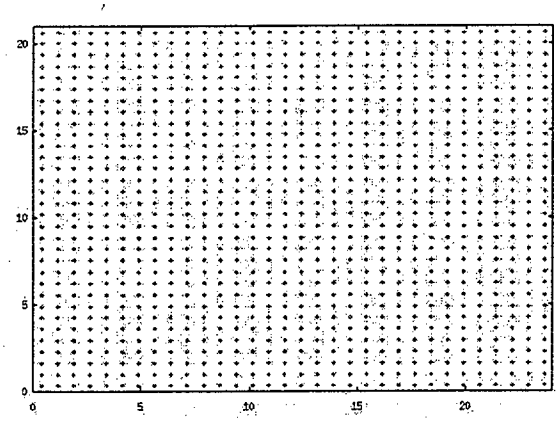


Figura 7.10: Prototipos utilizando una discretización uniforme.

Una vez que se dispone de las discretizaciones uniformes, el aprendizaje de la política se realiza siguiendo un proceso de aprendizaje equivalente al de los experimentos en *Car on the Hill*. Por tanto, se utiliza un valor de $\gamma = 0,9$, y las tuplas de experiencia se utilizan 20 veces en un proceso iterativo de reutilización de dichas tuplas, en el que se parte de un valor de α de 0,2, que se decrementa en 0,01 en cada iteración.

En la Figura 7.11 se muestran los resultados obtenidos. En el eje x se muestra el tamaño de la discretización utilizado. En este caso, se ha realizado el aprendizaje de la política de acción para discretizaciones de tamaño 400, 525, 676, 1024, 2116, 3721, 6400 y 9801 estados. Estos valores son cuadrados perfectos, es decir, que se ha utilizado el mismo número de niveles para discretizar las coordenadas x e y, excepto para el valor 525, que es 24×21, es decir, el tamaño exacto del dominio. Éste último valor para la discretización es óptima, ya que con ella no se pierde la propiedad de Markov en la discretización, y por lo tanto el dominio mantiene el determinismo. Además, se muestran los resultados para distintos valores de ruido en el dominio, tal y como se introdujo anteriormente. Por último, en el eje y se muestra el porcentaje de intentos en el que el robot sí ha sido capaz de llegar a la meta desde las distintas posiciones aleatorias dentro del dominio.

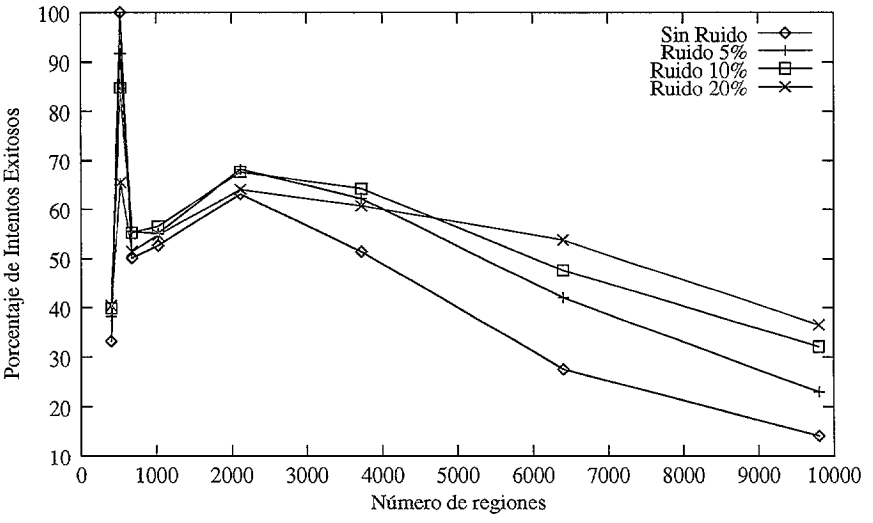


Figura 7.11: Porcentaje de intentos exitosos con discretizaciones uniformes de distintos tamaños y distintos niveles de ruido.

En esta figura se pueden destacar varias cosas. Con el dominio determinista, es decir, sin ruido, y 525 estados, se obtiene un 100 % de intentos en los que se llega a la meta. Este resultado era de esperar, ya que la discretización de 24×21 estados es óptima, como se indicó anteriormente. Por otro lado, se observa que si el dominio es estocástico, esta situación cambia en función de los niveles de ruido que tenga el dominio. Así, con un 5 % de ruido, el acierto baja hasta el 91,67 %, con un 10 % de ruido, baja hasta el 84,85 %, y con un 20 % baja hasta el 65,45 %. Esto es debido a la aleatoriedad introducida en el dominio por el ruido, que hace creer al agente estar en estados distintos al que en verdad está, y por tanto puede hacerle creer que está también en regiones distintas.

Sin tener en cuenta esta discretización óptima, se observa que para tamaños pequeños, como 400, 676 e incluso 1024 estados, los resultados son pobres, sin llegar al 60 %. Para 2116, así como para 3721 estados, se supera ya ese nivel, llegando incluso al 68.2 % con 2116 niveles y un nivel de ruido del 5 %. No obstante, para tamaños mayores, los resultados vuelven a descender, ya que el número de estados es muy grande, y se produce un mal aprovechamiento de la experiencia.

Un elemento a destacar es que cuando el dominio tiene ruido, se obtienen en general mejores resultados que cuando no lo tiene. Esto es debido a que la componente de aleatoriedad que introduce el ruido puede hacer que el agente salga de ciclos, ya que puede hacer que varíe ligeramente la situación del

agente, llegando a dos estados distintos, que además pueden estar situados en distintas regiones de la discretización, y que por tanto pueden tener asignadas acciones distintas en la política. Estas diferencias son mayores según el número de regiones es mayor, ya que cuanto más pequeñas son las regiones definidas, es más fácil que la pequeña variación en la percepción del robot producida por el ruido genere un cambio de región. En conclusión, se puede decir que el ruido hace que incremente la exploración del dominio, lo cual en ocasiones puede hacer que el robot se salga del buen camino, pero también puede sacarle de situaciones de bloqueo o ciclos.

En la Figura 7.12 se muestra el número de acciones medio que ha tardado el robot en llegar a la meta, incluyendo en esta media sólo aquellas ocasiones en las que en verdad se llegó a la meta.

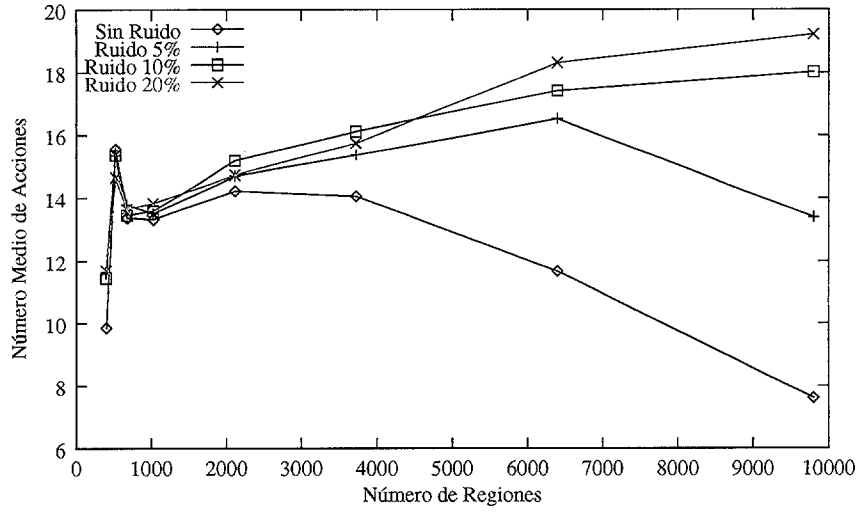


Figura 7.12: Número de acciones medio en llegar a la meta de entre los intentos exitosos con discretizaciones uniformes de distintos tamaños y distintos niveles de ruido.

El número de acciones medio para la política óptima encontrada para 525 estados en el dominio sin ruido es de 15,55 acciones. En la mayoría de los otros casos este valor es menor, debido no a que la política sea mejor, sino a que el robot es capaz de llegar menos a la meta, y normalmente, cuando lo hace, lo hace desde posiciones iniciales cercanas a ella. No obstante, para los dominios con ruido, y discretizaciones muy finas, se observa que el número de acciones es mayor. Esto corrobora la idea planteada anteriormente, en la que se explicaba que con altos niveles de ruido, se aumentaba la exploración. Es decir, se llega más a la meta, pero también se tarda más en hacerlo. Si se

penalizan aquellos intentos en los que no se llega a la meta, con un coste de 100, se tienen los resultados mostrados en la Figura 7.13. Aquí sí se observa que el menor valor se da para la discretización óptima de 525 estados, y cómo éste valor es muy alto para discretizaciones de tamaño pequeño (400 ó 676 estados) y de tamaño grande (6400 y 9801 estados). Prácticamente, se muestran curvas inversas a las de la Figura 7.11.

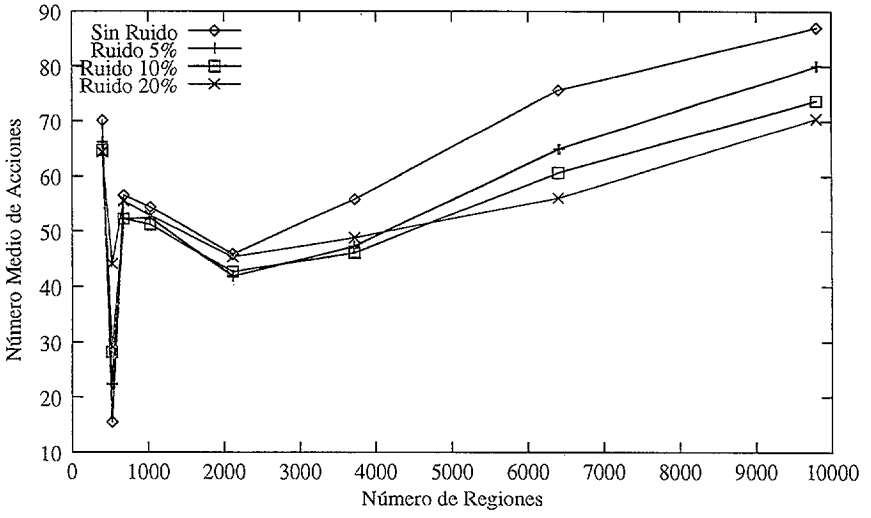


Figura 7.13: Número de acciones medio en llegar a la meta de entre todos los intentos con discretizaciones uniformes de distintos tamaños y distintos niveles de ruido.

7.2.2. VQQL

También se ha utilizado el modelo VQQL para resolver el problema de la navegación de un robot por un dominio de oficinas. Los parámetros de aprendizaje utilizados en este caso son los mismos que en el caso anterior, y además, se han realizado exactamente los mismos experimentos, en lo que se refiere a ruido en el dominio y tamaño de las discretizaciones.

El algoritmo VQQL plantea como paso inicial el aprendizaje de la discretización del espacio de estados. Para ello, se realiza una exploración sobre el entorno, que en este caso, es la misma que la desarrollada para obtener las tuplas de experiencia que serán utilizadas para aprender la política. Por tanto, la experiencia utilizada para aprender con este modelo, es exactamente la misma que la utilizada en la sección anterior con las discretizaciones uniformes.

Para obtener las discretizaciones se utiliza el Algoritmo de Lloyd Generalizado, tal y como se planteó en el capítulo 4. Del total de ejemplos se utiliza un 80 % como conjunto de entrenamiento, y otro 20 % como conjunto de test. En la Figura 7.14 se muestra cómo la distorsión media, tanto para el conjunto de entrenamiento como para el conjunto de test, va disminuyendo según el algoritmo va ejecutando distintas iteraciones, llegando a un valor de 0,044 y 0.059 respectivamente, en 14 iteraciones.

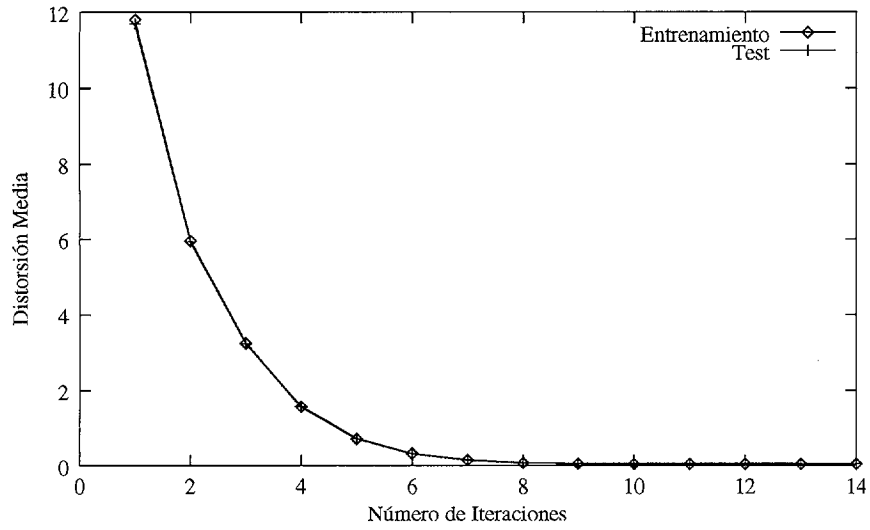


Figura 7.14: Evolución de la distorsión media al ejecutar GLA.

La Figura 7.15 muestra un ejemplo de los prototipos generados con GLA para una discretización de 1024 estados, y para el dominio sin ruido. En esta figura, se observa cómo los prototipos sólo son localizados en aquellas zonas relevantes del espacio de estados, es decir, en aquellas zonas por donde el agente va a pasar, y por tanto, en las que deberá tomar decisiones sobre qué nueva acción debe ejecutar. Además, al haber prototipos sólo en esas zonas, está asegurado que para todas las regiones, se van a actualizar los valores de la tabla Q. Dado que al modificar el ruido en el dominio, se puede considerar que se tratan dominio distintos, se ha obtenido una discretización por cada uno de ellos.

La Figura 7.16 muestra el porcentaje de intentos en que el robot ha llegado a la meta, utilizando para aprender la política de acción las discretizaciones obtenidas en el paso anterior, y siguiendo las mismas características de aprendizaje que en el caso uniforme, es decir, $\gamma = 0,9$, y un valor de $\alpha = 0{,}2$, que se va decrementando en $0,01$ en cada pasada por el conjunto de tuplas de experiencia.

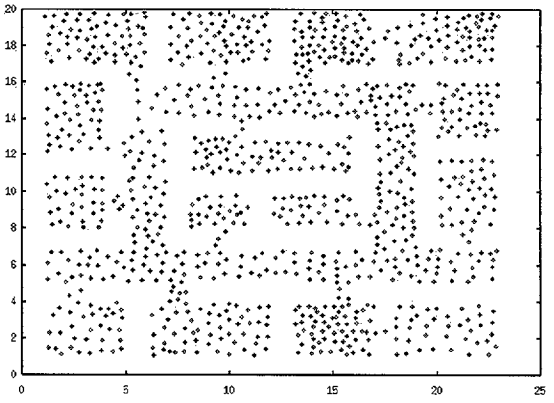


Figura 7.15: Prototipos utilizando una discretización generada con GLA.

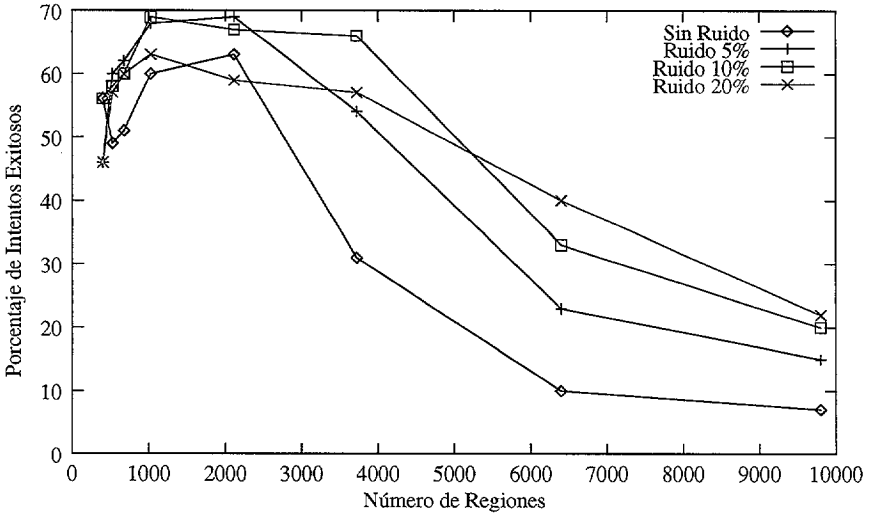


Figura 7.16: Porcentaje de intentos exitosos con VQQL y con distintos tamaños y distintos niveles de ruido.

Las curvas que se muestran en la figura son similares a las del caso uniforme, con dos diferencias principales. En primer lugar, en este caso no se obtiene la discretización óptima, y por tanto, nunca se llega al 100 % de intentos satisfactorios. En segundo lugar, en ambos experimentos, y sin contar la discretización óptima, se llega a valores cercanos al 70 %, si bien, con *VQQL* se consigue con un número de estados menor, ya que con 1024 se alcanzan esos niveles. El resto de características se mantienen similares. Por ejemplo, el hecho de que, según aumenta el ruido en el dominio, al aumentar la exploración, se conseguían mejores resultados con las discretizaciones uniformes también sigue siendo válido con las discretizaciones obtenidas con GLA. Las Figuras 7.17 y 7.18 muestran el tiempo medio en llegar a la meta. En la primera de ellas, sólo se tienen en cuenta para hacer la media aquellos intentos en que, en efecto, se ha llegado a la meta, mientras que en la segunda, los intentos fallidos se han penalizado con un tiempo de 100, y se han incluido en la media. Los resultados que se observan en estas dos figuras son las esperadas a partir de los resultados mostrados en la figura anterior, si se extrapolan las conclusiones obtenidas para el caso uniforme.

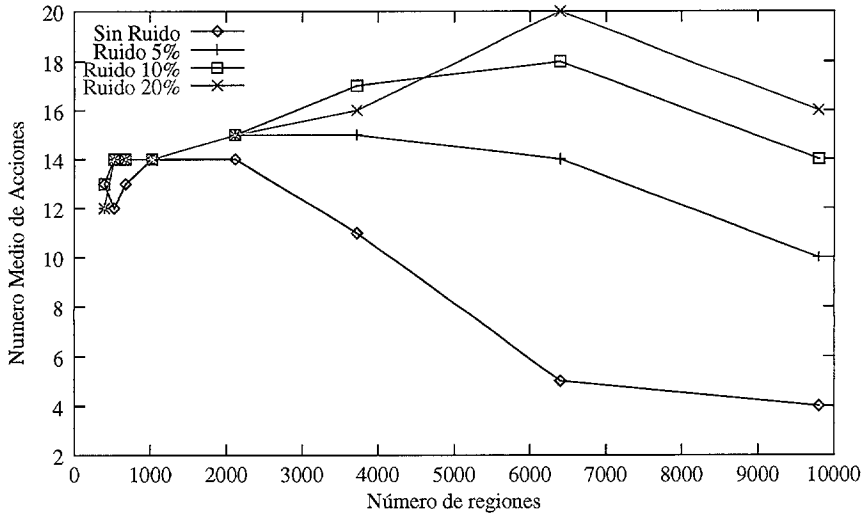


Figura 7.17: Número de acciones medio en llegar a la meta de entre los intentos exitosos con VQQL y con distintos tamaños y distintos niveles de ruido.

7.2.3. *Iterative Smooth Q-Learning* con C4.5

Los resultados de aplicar *Iterative Smooth Q-Learning* con C4.5 sobre el dominio de navegación por oficinas se muestran en la Figura 7.19, para un

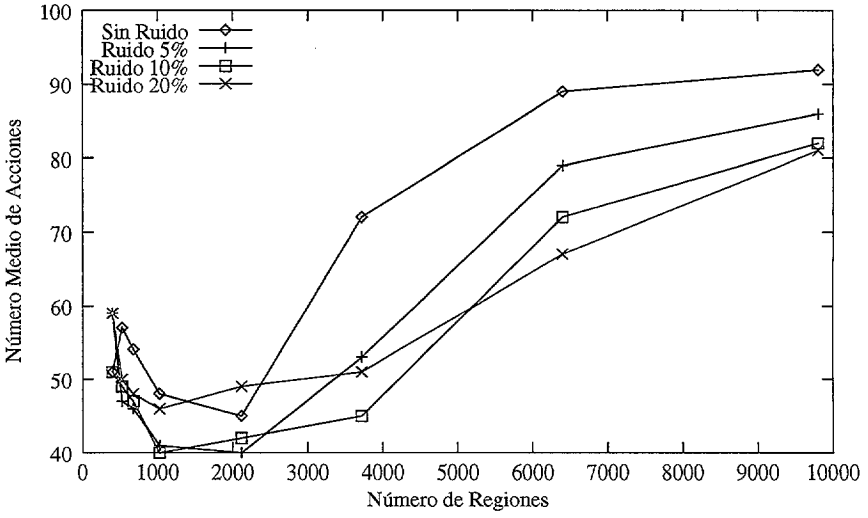


Figura 7.18: Número de acciones medio en llegar a la meta de entre todos los intentos con VQQL y con distintos tamaños y distintos niveles de ruido.

proceso de aprendizaje con los mismos parámetros que los utilizados en los experimentos anteriores. No obstante, en estos experimentos, los tests han sido realizados sobre 1000 intentos, en lugar de los 4000 utilizados en los experimentos anteriores [1]. En la figura se representa en el eje x el porcentaje de ruido, para los valores de 0, 5, 10 y 20 % habituales en los experimentos anteriores. En el eje y se muestra el porcentaje de veces que el robot es capaz de alcanzar la zona de meta.

Se observa que el algoritmo es muy sensible al ruido en el dominio. Así con el dominio determinista, se alcanza un porcentaje de éxito de un 64,8, mientras que según se va incrementando el ruido, este porcentaje desciende hasta el 22,4 % para el dominio con un ruido del 20 %. Para determinar el número de iteraciones que se deben ejecutar de *Iterative Smooth Q-Learning*, se ha seguido una heurística sencilla, de forma que se considera que se ha aprendido completamente cuando los refuerzos son propagados a todo el dominio. Si suponemos que inicialmente el valor de Q para cualquier estado y acción se inicializa a un valor nulo, una vez que se modifique este valor para todo el entorno, se puede decir que se han propagado los refuerzos desde las zonas de meta a todo el entorno y por tanto, que se puede detener el algoritmo. De esta forma, se han requerido 29, 27 27 y 29 iteraciones para los dominios con un 0, 5, 10 y 20 % de ruido respectivamente. Las diferencias entre unos

[1] Esta reducción viene motivada únicamente por acelerar la experimentación, ya que la integración con WEKA hace que el sistema sea muy lento.

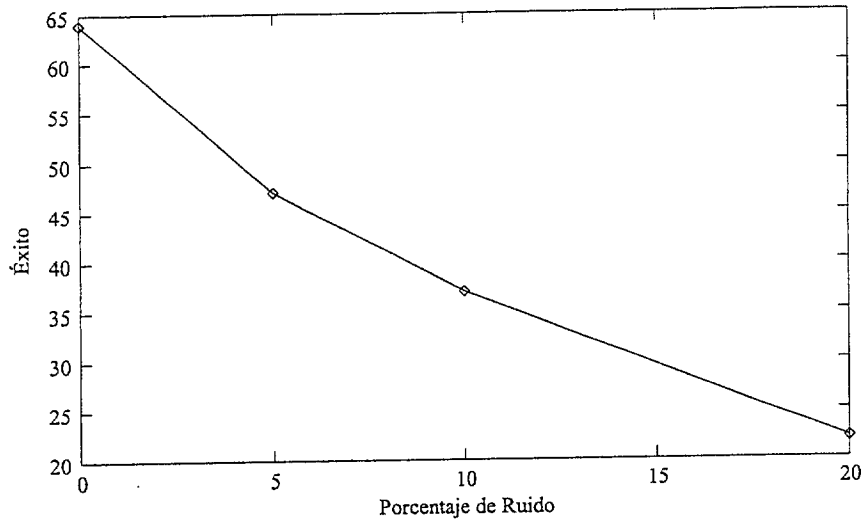


Figura 7.19: Porcentaje de veces que se llega a la meta, aplicando *Iterative Smooth Q-Learning* con árboles de decisión en el dominio de oficinas.

y otros son debidas a los errores que introduce el aproximador por el ruido de los datos.

La Figura 7.20 muestra el número de acciones que ha de ejecutar para alcanzar la meta en las ocasiones en que es capaz de alcanzarla, y además, este último valor pero penalizando las veces que no se ha llegado a la meta con un valor de 100. Dado que el porcentaje de veces que se llega a la meta es bajo, el número de acciones medio necesario para alcanzarla es pequeño y decrece según va decrementando el primer valor, debido a que las pocas veces que llega, lo hace desde situaciones ya cercanas a la meta. Por eso, si se penaliza con 100 las veces en que no se llega a la meta, este valor es muy alto.

En cuanto las características de los árboles de decisión generados, cabe destacar que el tamaño de éstos se mantiene bastante estable. En la Figura 7.21 se muestra que el número medio de hojas en los árboles generados para cada acción se mantiene entre 500 y 552 hojas, mientras que en lo que al tamaño del árbol se refiere, este valor varía desde 996 hasta aproximadamente 1300. El número de hojas es un valor muy significativo, porque indica el número de regiones en las que el árbol de decisión ha dividido el entorno.

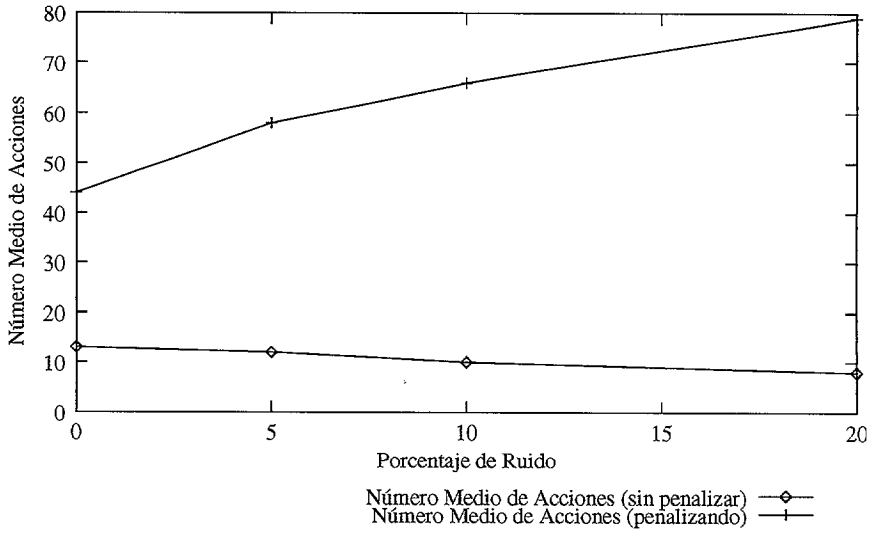


Figura 7.20: Número de acciones para llegar a la meta, aplicando *Iterative Smooth Q-Learning* con árboles de decisión en el dominio de oficinas.

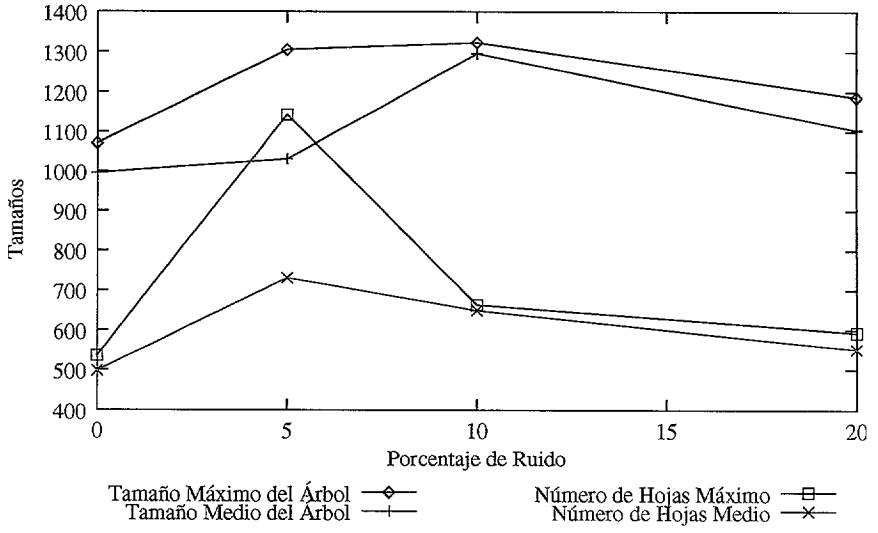


Figura 7.21: Tamaño de los árboles de decisión generados en el dominio de oficinas.

7.2.4. ENNC-QL

La Figura 7.22 muestra los resultados de aplicar el algoritmo ENNC-QL en el dominio de navegación por oficinas. Al igual que en los casos anteriores, se ha realizado la experimentación para el domino con los distintos niveles de ruido. Además, se muestran dos resultados. El primero de ellos tras ejecutar la primera fase de aprendizaje del algoritmo, y el segundo de ellos tras ejecutar las dos fases de aprendizaje, tal y como se hizo para el dominio *Car on the Hill*. Para realizar la segunda fase de aprendizaje, se han utilizado los mismos parámetros que para la discretización uniforme y para VQQL, si bien, en este caso, la tabla Q se inicializa con la aproximación obtenida en la fase anterior, tal y como define el algoritmo ENNC-QL.

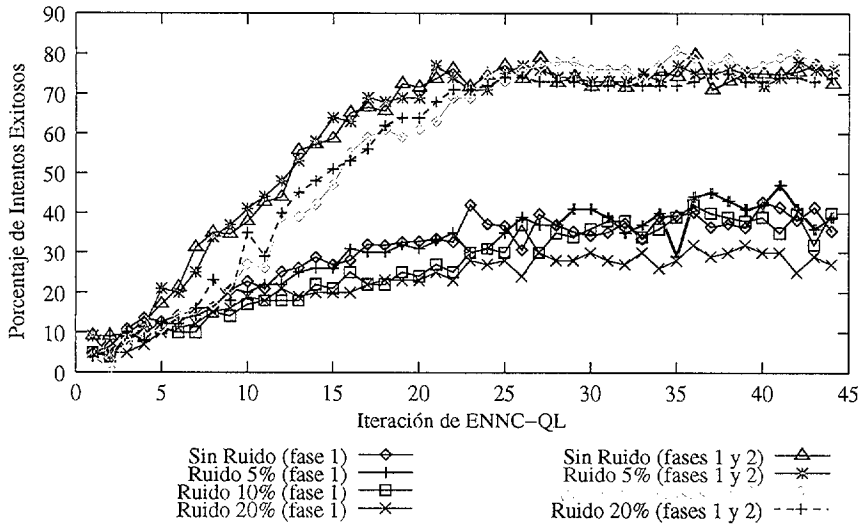


Figura 7.22: Porcentaje de intentos exitosos con ENNC-QL y con distintos niveles de ruido.

En la figura se muestra cómo evoluciona el aprendizaje según se van ejecutando distintas iteraciones del algoritmo ENNC-QL. Se ha realizado el experimento para el dominio con un nivel de ruido distinto, y se han ejecutado un total de 44 iteraciones. Se observa que los resultados cuando se ejecuta sólo la primera fase de aprendizaje oscilan entre un 35 % y un 45 %, mientras que al ejecutar el algoritmo completo, los porcentajes alcanzan el rango de 70-80 %. Los resultados de la primera fase confirman los esperados para un aproximador de función supervisado. En este caso, se puede decir que el algoritmo converge a una función de valor y una política, pero esta política es pobre. Los errores transmitidos propagados por el método supervisado se

pueden solventar al considerar las discretizaciones obtenidas por el clasificador ENNC, y realizando la segunda fase del aprendizaje, llegando a niveles mucho más altos. Por tanto, se puede concluir que aunque la aproximación de la función de valor-acción obtenida por el clasificador ENNC es pobre, la discretización del espacio de estados que genera sí es muy útil, permitiendo un aprendizaje con altos niveles de éxito.

Al igual que ocurría con el caso uniforme y con VQQL, el dominio con ruido da mejores resultados en cuanto al porcentaje de intentos exitosos debido a la mayor exploración que permiten. La Figura 7.23 muestra el tiempo medio en llegar a la meta para aquellos intentos en que en verdad se ha llegado a la meta.

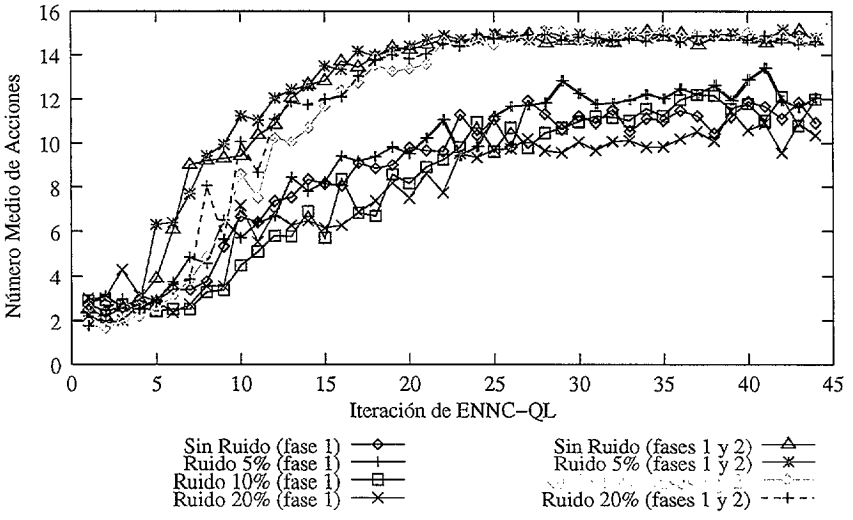


Figura 7.23: Número de acciones medio en llegar a la meta de entre los intentos exitosos con ENNC-QL y con distintos niveles de ruido.

En la figura se observa que con niveles de ruido del 20 % en el dominio, el tiempo medio en llegar a la meta es mayor. la Figura 7.24 muestra el número de acciones medio, pero en el que se ha penalizado con un valor de 100 aquellos intentos no exitosos.

Como se introdujo anteriormente, el número de prototipos utilizado es calculado automáticamente por el algoritmo ENNC. En la Figura 7.25 se muestra el número de prototipos medio de entre las discretizaciones generadas para cada acción, al igual que el número máximo de ellos.

La figura muestra que, según se va aumentando el nivel de ruido en el dominio, los datos con los que se genera el clasificador y, por tanto, el conjunto de prototipos, son más ruidosos. Para el algoritmo ENNC esto supone una

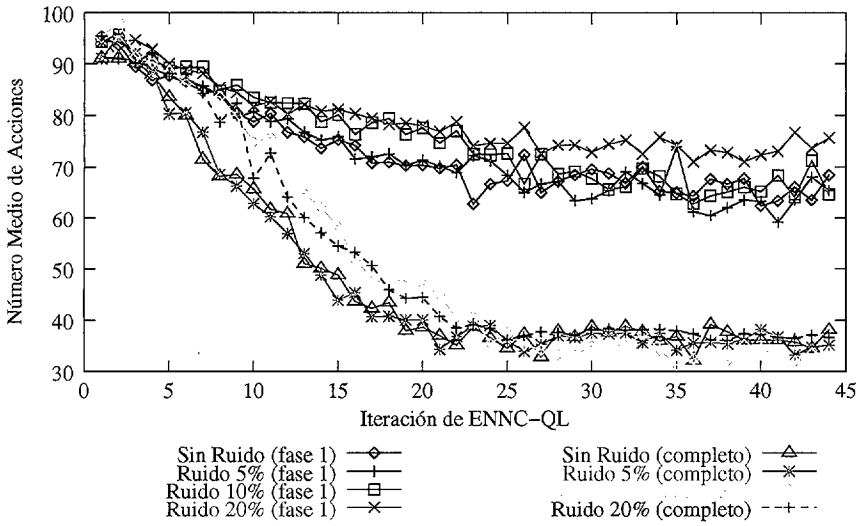


Figura 7.24: Número de acciones medio en llegar a la meta de entre los todos los intentos con ENNC-QL y con distintos niveles de ruido.

sobreadaptación sobre los datos ruidosos, lo que se convierte en que introduce más prototipos para intentar incrementar los porcentajes de clasificación. Sin embargo, cabe destacar que, aunque el número de regiones en las discretizaciones puede crecer enormemente, como se observa para el dominio con un 20 % de ruido, donde el número de prototipos supera los 7000, los resultados siguen siendo muy buenos, en lo que a éxito en resolver la tarea de navegación se refiere.

Por otro lado, queda aún un punto por definir. En estos experimentos se han ejecutado un total de 44 iteraciones dentro del algoritmo ENNC-QL, pero, ¿cuál es el número adecuado? Obviamente, dado que el clasificado ENNC es estocástico (podría generar resultados distintos incluso con los mismos datos de entrada), es muy difícil concretar una condición de fin basándonos en el aproximador. Sin embargo, esto sí se puede realizar si tenemos en cuenta la suposición de que el dominio es determinista y que tarda en aprender todo el dominio el tiempo necesario para propagar los refuerzos positivos obtenidos al llegar a la meta hasta el lugar más lejano, en lo que a número de acciones en llegar a la meta se refiere. Dado que este valor es desconocido "a priori", una forma sencilla de implementarlo es cuando, en el proceso iterativo, ya no se generan estados para los cuales se tiene un valor nulo. Es decir, para todos los estados de ejemplo del dominio, ya se dispone de su valor Q. Esta aproximación ya fue utilizada con los experimentos realizados con *Iterative Smooth Q-Learning* con árboles de decisión.

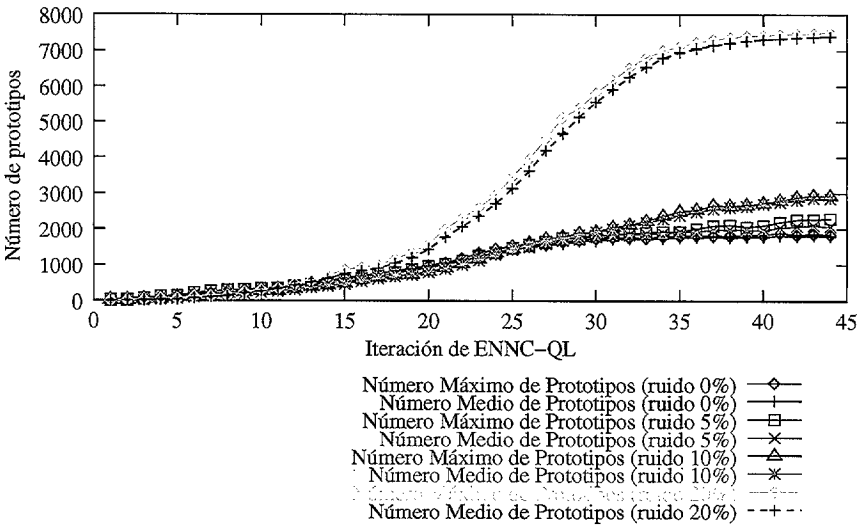


Figura 7.25: Prototipos generados con ENNC-QL para el dominio con distintos niveles de ruido.

En este ejemplo, la convergencia se produce en iteraciones distintas según el nivel de ruido. Así para el dominio determinista y con un 5 % de ruido, se consigue en la iteración 25, mientras que para el dominio con valores de ruido del 10 % y del 20 % se consigue en la iteración 28. Las diferencias entre unos y otros vienen dadas, nuevamente, por el error en la propagación de los refuerzos. Si consideramos estas iteraciones como las definitivas, en la Tabla 7.2 se puede ver una comparativa entre los resultados obtenidos para todos los métodos. En la tabla se muestra el mejor resultado obtenido por VQQL y por la discretización uniforme, si bien para esta última se ha excluido la discretización óptima.

	Uniforme		VQQL		C4.5		ENNC-QL	
% Ruido	Éxito	Prot.	Éxito	Prot.	Éxito	Hojas	Éxito	Prot.
0	63,08	2116	63,75	2116	64,8	498,5	77	1437,5
5	68,2	2116	69,65	2116	47,1	731	75,4	1443
10	67,65	2116	69,27	1024	37,6	649	78,62	1737,25
20	64,00	2116	63,05	1024	22,4	552,5	73,45	5123,75

Tabla 7.2: Comparativa de los distintos métodos en el dominio de navegación por oficinas.

Además, se observa que VQQL produce, en general, niveles de éxito similares a los que produce la discretización uniforme. Sin embargo, en general,

requiere de menor número de estados. Esto, que en este dominio bidimensional parece no ser muy importante, puede ser crítico con espacios de más dimensiones. El algoritmo ENNC-QL genera mejores resultados en lo que a la política obtenida se refiere, siendo superior en todos los casos. Además, esto lo hace manteniendo un número de prototipos similar al que dan los mejores resultados tanto para discretizaciones uniformes, como para VQQL.

En el capítulo 5 se mostró que el clasificador ENNC es un clasificador evolutivo con una alta componente estocástica. A pesar de ello, en los experimentos mostrados en ese capítulo se comprobó que las soluciones obtenidas eran muy similares en distintas ejecuciones del algoritmo, por lo que se planteaba que una única ejecución del algoritmo es suficiente para asegurar altos porcentajes de clasificación, con conjuntos de prototipos reducidos. El algoritmo ENNC-QL, en cada iteración, realiza una ejecución del algoritmo ENNC, por lo que se convierte también en un algoritmo estocástico que en distintas ejecuciones podría obtener resultados también distintos. Para comprobar si las diferencias entre unas ejecuciones y otras son grandes, se ha ejecutado el algoritmo ENNC-QL sobre el dominio de navegación por oficinas determinista (es decir, con ruido 0) otras 4 veces. Los resultados se muestran en la Tabla 7.3.

Iteraciones	Prototipos	Éxito	Acciones
25	1437,5	77	14,89
28	1384	74,7	14,81
28	1503	73,15	14,41
29	1416,25	72,37	14,80
29	1470	75,57	14,7

Tabla 7.3: Resultados de distintas ejecuciones de ENNC-QL en el dominio de oficinas con ruido 0.

La tabla muestra, para cada ejecución, el número de iteraciones que fueron necesarias ejecutar en la primera fase del algoritmo, el número medio de discretizaciones de entre las 4 que se generan, el éxito o porcentaje de veces que se llega a la meta, y el número de acciones que fueron necesarias ejecutar para alcanzar dicha meta. Entre el peor y el mejor resultado de las 5 ejecuciones hay una diferencia de casi 5 puntos en el porcentaje de éxito; un valor esperado, ya que estas diferencias se dan incluso dentro de una misma ejecución entre distintas iteraciones en la fase de aprendizaje, tal y como mostraba la Figura 7.22. No obstante, todos los resultados superan a las otras aproximaciones, con un mínimo del 72,37 %. En lo que se refiere al

número de iteraciones, también se observa que varía de unas ejecuciones a otras, al igual que el número de prototipos.

Como conclusión, se puede decir que el algoritmo ENNC-QL es capaz de resolver el problema de navegación por oficinas de una forma muy eficaz, obteniendo mejores resultados que el resto de los métodos aplicados. Además, esto se cumple en el dominio determinista, pero también cuando el dominio tiene una fuerte componente estocástica, como es, en este caso, la percepción de la situación en la que se encuentra el robot en cada momento. Se ha mostrado que incluso con un ruido de un 20 %, los resultados siguen siendo similares a los del caso determinista, si bien se produce un aumento en el número de regiones que se utilizan en las discretizaciones.

7.3. CMOMMT

El último dominio en el que se han realizado experimentos es el de CMOMMT. Este dominio fue introducido en la sección 2.7.3, y los primeros experimentos sobre él, utilizando el algoritmo *VQQL*, fueron ya expuestos en la sección 4.4.2. El objetivo de este último experimento es verificar si el algoritmo ENNC-QL es aplicable también en este entorno, ya que plantea tres problemas importantes. En primer lugar, es un dominio extremadamente ruidoso, ya que las transiciones de estado no dependen sólo de las acciones del agente que está aprendiendo, sino del resto de los agentes, y de los objetivos que están siguiendo. En segundo lugar, tal y como se planteó en la sección 4.4.2, el aprendizaje se realiza en gran medida a partir de refuerzos negativos recibidos cuando se deja de observar a los objetivos. Y estos refuerzos no se propagan al resto del entorno. Por último, el espacio de acciones es mayor que en los experimentos anteriores, ya que este dominio se ha definido para 8 acciones, y el espacio de estados también puede ver incrementada su dimensionalidad. A continuación, se describe el uso de *ENNC-QL* para los dos experimentos planteados en la sección 4.4.2.

7.3.1. Experimento 1

En este experimento cada estado se compone de 2 atributos, correspondientes a las coordenadas x e y del objetivo más lejano. El proceso de aprendizaje es de 1000 intentos, durante los cuales el agente se mueve de forma aleatoria en el entorno, hasta un máximo de 100 acciones. Durante este aprendizaje, el número de objetivos es 10, mientras que un único robot es el encargado de aprender la política de acción. Posteriormente, la política de acción aprendida por este único robot será utilizada también por los demás.

Al final de cada intento, se recibe como refuerzo el número de objetivos que se están observando en ese momento. Además, durante los intentos, si el robot deja de ver los objetivos, recibe un refuerzo de -100.

La Figura 7.26 muestra los ejemplos con los que debe aprender el clasificador ENNC en la primera iteración de la primera fase de aprendizaje de ENNC-QL, para la acción en la que el robot se mueve en dirección Sur. En la figura se observa que la mayoría de los ejemplos corresponden a instancias que tienen un refuerzo inmediato asociado de 0. Sin embargo, en la zona norte, abundan también instancias con refuerzo inmediato -100. Estas instancias se corresponden con la situación en la que el robot tiene un objetivo en el Norte, y ejecuta la acción de "Ir Sur". En este caso, dependiendo del movimiento del objetivo, es probable que el robot deje de verlo, ya que se mueve en dirección contraria a donde está dicho objetivo. Además, se aprecia que el número de instancias con un refuerzo positivo de 100 es muy escaso, y prácticamente inexistentes las que tienen un refuerzo de 200.

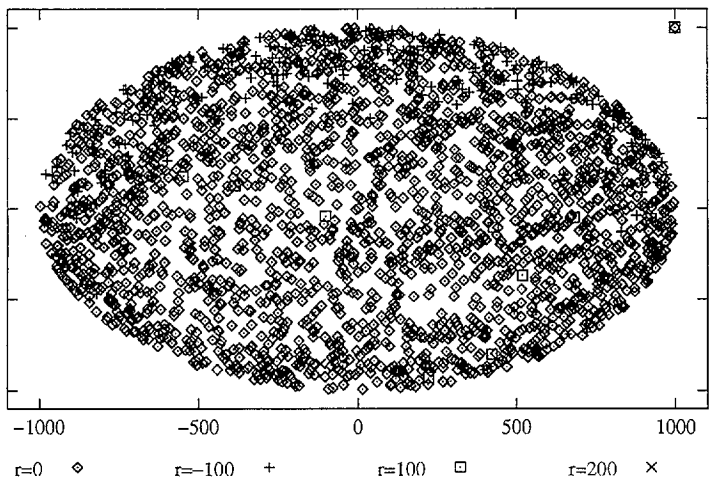


Figura 7.26: Instancias de entrenamiento para ENNC para aproximar la función de valor-acción $Q_{IrSur}(s)$.

Ante este conjunto de entrenamiento, el algoritmo ENNC genera para la acción "Ir Sur", (para un máximo de 1.000 iteraciones), un clasificador que se muestra en la Figura 7.27. En la figura se observa que se han generado principalmente prototipos con un valor del refuerzo de 0, pero además se han generado algunos prototipos de refuerzo -100 en la zona norte. Las instancias de entrenamiento pertenecientes a otras clases han sido eliminadas como ruido. Por tanto, al sólo generar refuerzos nulos y negativos en esta primera iteración, ya no tiene sentido seguir ejecutando más iteraciones, ya que, como

se ha comentado anteriormente, los refuerzos negativos no se propagan.

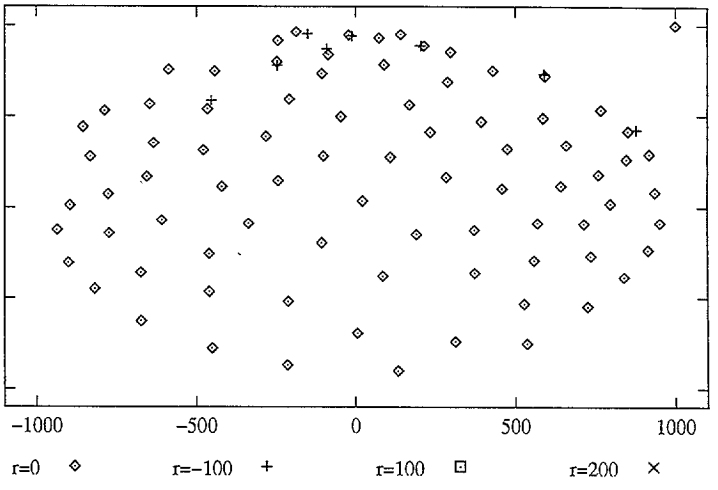


Figura 7.27: Clasificador obtenido por ENNC como aproximador de la función de valor-acción $Q_{IrSur}(s)$.

Cabe destacar que en este caso, salir del mínimo local de otorgar a todo el dominio el mismo valor, es decir, un valor 0, no es fácil. Es un problema de clasificación en el que más de un 91 % de la instancias pertenecen a la misma clase, con valor 0, y el resto de instancias que no tienen ese valor son difícilmente diferenciables de las primeras, debido a que los atributos de que se disponen no son suficientes para realizar esa separación. Sirva como ejemplo, que si se aplica C4.5 sobre este conjunto de datos, el clasificador devuelve un árbol con una única hoja de clase 0. En este sentido, sería necesario incluir información sobre la dirección de los objetivos para que el clasificador pudiera separar las instancias de cada clase.

En lo que al algoritmo ENNC se refiere, se podría llegar a obtener clasificadores de un único prototipo que clasificaran al dominio completo como perteneciente a la misma clase, en este caso, la de refuerzo nulo. Esto es debido a que los datos distintos a los de la clase mayoritaria, serían entendidos como ruido de los datos de la clase mayoritaria. Esta situación es la única que puede hacer fallar totalmente el modelo ENNC-QL, ya que generaría discretizaciones del espacio de estados de un único prototipo, solución que es inaceptable desde el punto de vista del aprendizaje por refuerzo.

Todos estos inconvenientes se derivan de que se está tratando un dominio no Markoviano como tal, y eso se refleja en un alto grado de indeterminismo. Un problema equivalente al de la introducción de indeterminsmo derivada de la pérdida de la propiedad de Markov al realizar discretizaciones del espacio

de estados, pero que debe ser afrontada desde puntos de vista distintos.

Una vez ejecutada la primera fase de aprendizaje del algoritmo ENNC-QL, se ejecuta la segunda fase sobre los conjuntos de prototipos generados anteriormente, utilizando en el aprendizaje los mismos parámetros que fueron definidos en la sección 4.4.2, es decir, $\gamma = 0,6$, un α fijado a 0,05, y las tuplas de experiencia utilizadas en la primera fase. El porcentaje medio de objetivos bajo el rango de visión de uno de los 10 robots utilizados en la fase del test (todos usando el mismo comportamiento aprendido) a lo largo de 10 ejecuciones de una duración de 1.000 ciclos cada una, es del 54,12 %. Para esta solución, se han generado un número de prototipos medio de 107,2 prototipos. Esta solución es ligeramente inferior a la de VQQL mostrada en la Figura 4.18, en la que se mostraron resultados de aproximadamente el 60 %, aunque sí se mejoran los resultados obtenidos en otras aproximaciones anteriores. El porqué VQQL genera unos resultados mejores, se puede comprender fácilmente si se observa la Figura 4.17, donde se mostraban dos discretizaciones del espacio de estados obtenidas mediante VQQL, para 16 y 64 prototipos, y cómo esas discretizaciones están muy adaptadas al problema que se trata, ya que permiten diferenciar muy exactamente las zonas desde las cuales se dejarán de ver los objetivos (zona exterior) de las que no.

Para verificar si la componente estocástica del algoritmo ENNC tiene o no influencia en el resultado obtenido, se ha ejecutado el algoritmo ENNC-QL en este dominio otras 4 veces más. Los resultados, en lo que se refiere a tamaño medio de las discretizaciones obtenidas en cada ejecución, así como en el éxito en la tarea de seguimiento, se muestran en la tabla 7.4.

Prototipos	Éxito
107,2	54,12
91,37*	61,42
110	58,22
90,5*	58,88
110,62	59,62

Tabla 7.4: Resultados de distintas ejecuciones de ENNC-QL en el dominio *CMOMMT*

Cabe destacar, que en 2 de ellas (con el número de prototipos marcado con un '*'), hay una acción para la que el clasificador ENNC no ha sido capaz de salir del mínimo local de un prototipo y una única clase. Eso produce que todo el espacio, para esa acción, tiene el mismo valor de Q. No obstante, dado que hay 7 acciones más, el resultado global aprendido es similar al del resto de los casos, no viéndose perjudicado aparentemente. En general, se observa

que los resultados varían entre un 54 y un 61 %, para discretizaciones de aproximadamente 110 regiones, este último valor obtenido si no se tienen en cuenta los casos en los que se obtuvieron discretizaciones de una única región.

7.3.2. Experimento 2

El segundo experimento está basado en los descritos en la sección 4.4.2, en el que se planteaba la introducción de comportamientos cooperativos basándose en dos puntos esenciales. El primero de ellos es que el estado del robot aumenta su dimensión para incluir también información, no sólo del objetivo más lejano, sino también del objetivo más cercano y del robot más cercano. Por tanto, la dimensión es ampliada hasta 6. El segundo es que este aumento en el conocimiento que el agente tiene del entorno permite modificar la función de refuerzo, penalizando situaciones en las que el robot en cuestión veía a otros robots, tal y como definía la ecuación 4.10.

El proceso de aprendizaje es el mismo que se siguió para VQQL. La única diferencia es que en ese caso, los 10 agentes aprendían a la vez sobre la misma tabla Q durante 500 ejecuciones, de 100 acciones cada una. En este caso, cada uno de los robots puede aprender su propia política, y por tanto, su propia aproximación de Q, durante un total de 5000 ejecuciones. En lo que al número de actualizaciones sobre la tabla Q se refiere, ambos experimentos se consideran equivalentes, ya que en el primero de ellos hay 10 robots actualizando la misma tabla durante 500 ejecuciones, y en el segundo cada robot hace sus propias actualizaciones durante las 5000 ejecuciones.

Al igual que en el experimento anterior, la primera fase de aprendizaje de ENNC-QL sólo se ejecuta una iteración. En dicha fase, el número de iteraciones máximo fijado para el algoritmo ENNC para obtener los prototipos de cada una de las 8 representaciones del espacio de estados es de 2.000, generando un número medio de 187,37 prototipos en esta primera ejecución.

Tras ejecutar la primera fase de aprendizaje, existen dos opciones para la segunda fase. La primera opción es que cada robot, a partir de la discretización obtendida en la fase anterior, aprenda su propia tabla Q. La segunda opción es que se aprenda una única tabla. En este caso, se obtendrán resultados para las dos opciones. Con la primera opción, tras realizar el proceso de aprendizaje completo, el test muestra un resultado de un 67,41 % de objetivos bajo observación, ligeramente superior al que se obtuvo con VQQL. Además, si en vez de utilizar una política por agente, todos utilizan la misma, como ocurría en el experimento realizado con VQQL (opción 2), este valor sólo desciende a un 66,44 %. Se comprueba, por tanto, que el algoritmo ha sido capaz, en este caso, de mejorar todas las aproximaciones anteriores (excepto la aproximación generada a mano), manteniendo además un núme-

ro de prototipos bastante reducido. Además, cabe destacar que el número de prototipos de las discretizaciones obtenidas, con los que se han alcanzado estos resultados (187,37 como media del tamaño de las 8 discretizaciones), es muy inferior a los requeridos por VQQL (2116).

Al igual que se hizo en el experimento anterior, se ha ejecutado el algoritmo ENNC-QL en este dominio otras 4 veces más para verificar si la componente estocástica del algoritmo ENNC tiene o no influencia en el resultado obtenido. Los resultados se muestran en la tabla 7.5.

Prototipos	Éxito (opción 1)	Éxito (opción 2)
187,37	67,41	66,44
192,87	66,96	61,34
192,12	65,99	60,5
196,5	66,38	62,35
149,75	65,91	61

Tabla 7.5: Resultados de distintas ejecuciones de ENNC-QL en el dominio *CMOMMT*

En la tabla se muestra que los resultados oscilan practicamente entre el 66 y el 67 % para la opción 1 (cada robot aprende su propia tabla Q), produciendo un valor medio de 66,53 % de objetivos bajo observación. Para la opción 2 (todos los robots usan la misma tabla Q), los resultados bajan ligeramente hasta el 62,35 %. Todos estos resultados se resumen en la Figura 7.28, donde se incluyen los resultados obtenidos por todos los métodos aplicados hasta el momento en este dominio, algunos de ellos introducidos en la sección 4.4.2, como la solución programada a mano (A-CMOMMT), *Pessimistic Lazy Learning* y VQQL. Para ENNC-QL, se introduce como valor la media de las cinco ejecuciones realizadas.

7.4. Conclusiones

Los experimentos mostrados en este capítulo permiten evaluar el cumplimiento de los objetivos planteados en el capítulo 3. Para ello, se ha realizado una extensa experimentación en tres de los cuatro dominios planteados en la sección 2.7, cada uno de ellos con objetivos distintos.

El primero de ellos, *Car on the Hill*, ha comprobado que cuando se utilizan métodos basados en la discretización del espacio de estados, encontrar una discretización adecuada puede no ser sencillo. Se ha mostrado que utilizando discretizaciones uniformes y VQQL se consiguen resultados aceptables, en

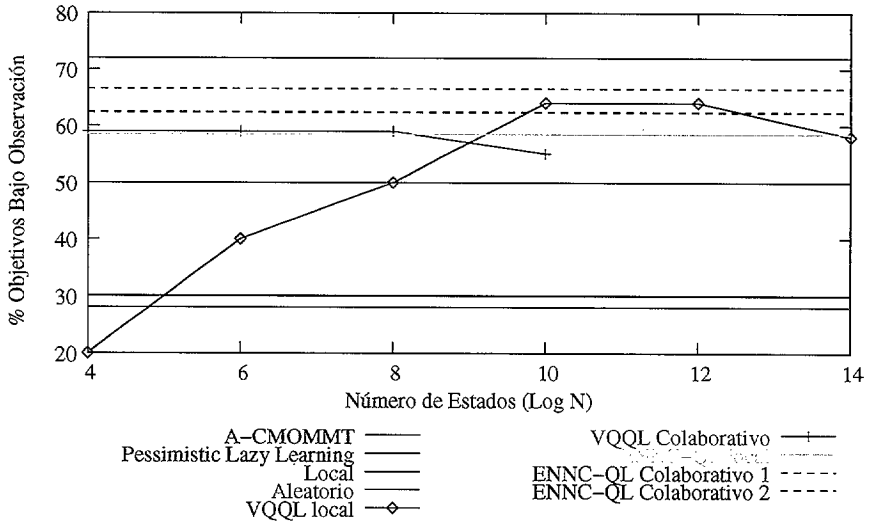


Figura 7.28: Éxito de distintas aproximaciones en el dominio *CMOMMT*.

cuanto al porcentaje de veces que se llegaba a la cima de la montaña, con discretizaciones de un tamaño pequeño (16 ó 64 estados). Sin embargo, si además se desea que la velocidad de llegada sea mínima, ese número de estados debe ser incrementado. El algoritmo *ENNC-QL* alcanza una solución adecuada de forma automática, definiendo por sí mismo el tamaño de la discretización. Es, por tanto, capaz de resolver el problema eficientemente, con el número de tuplas de experiencia igual al de los otros dos métodos, y de forma cercana al óptimo. Además, estos resultados eran obtenidos incluyendo un único parámetro en la ejecución del algoritmo ENNC, en uno de los pasos internos de ENNC-QL, como es el número de iteraciones.

En segundo lugar, se ha realizado una experimentación en el dominio de navegación por oficinas definido en la sección 2.7.1. El objetivo en este caso era verificar cómo afectaba el ruido a la eficacia de los algoritmos, es decir, si eran válidos no sólo en dominios deterministas, sino también en dominios estocásticos. En este caso, el algoritmo *ENNC-QL* ha mostrado ser muy estable, obteniendo de forma automática resultados muy similares en todos los casos, independientemente del indeterminismo del dominio. Esta situación, sin embargo, no se conseguía con los otros métodos aplicados, especialmente con el uso de C4.5 como aproximador de funciones en *Iterative Smooth Q-Learning*, que se veía muy afectado por el ruido. Esto muestra la necesidad de parametrizar el aproximador utilizado, teniendo en cuenta la cantidad de ruido que debe absorber. No obstante, el clasificador *ENNC* ha

mostrado que es capaz de absorber ese ruido hasta porcentajes bastante altos, sin necesidad de parámetros adicionales.

Por último, el dominio CMOMMT ha sido también objeto de estudio, partiendo de los resultados planteados en la sección 4.4.2 para el modelo VQQL y otras aproximaciones anteriores. El objetivo de este experimento ha sido verificar el comportamiento de ENNC-QL en un dominio de aplicación real, en el que sus capacidades son forzadas. Así, se ha llegado a utilizar el algoritmo sobre un espacio de 8 acciones, con hasta 6 dimensiones en el espacio de estados, y con unos niveles de indeterminismo en el dominio muy altos. El algoritmo también ha encontrado soluciones satisfactorias en estas condiciones, mostrando que es un método independiente del dominio, y que introduciendo pequeñas restricciones sobre algunos elementos, como la discretización del espacio de acciones, o de la función de refuerzo, permite su utilización en cualquier dominio en el que sea aplicable el aprendizaje por refuerzo.

Capítulo 8

Conclusiones, Aportaciones y Trabajos Futuros

A lo largo de esta memoria se han presentado dos modelos de aprendizaje por refuerzo que permiten aprender comportamientos de forma efectiva. El modelo VQQL se basa en la discretización no supervisada del espacio de estados, lo cual permite de forma sencilla utilizar representaciones tabulares de las funciones de valor. El modelo ENNC-QL toma ventajas de la aproximación de funciones supervisada y de los métodos basados en discretizaciones para llegar a resultados por encima de los que se obtiene con los primeros por separado. Por último, la búsqueda de aproximadores de funciones que cumplieran los requisitos adecuados, llevó al diseño del algoritmo de clasificación supervisada ENNC, utilizado de forma exitosa en dominios clásicos de clasificación, así como en su aplicación dentro del algoritmo *ENNC-QL*.

En este capítulo se revisan las principales conclusiones extraídas de esta tesis doctoral. En los capítulos dedicados a VQQL y ENNC, ya se extrajeron algunas conclusiones referentes al comportamiento de dichos algoritmos, de igual forma que se ha hecho con ENNC-QL en el capítulo anterior. Por tanto, este capítulo se centra en algunas conclusiones más generales.

Además, en este capítulo también se recogen las principales aportaciones de esta tesis doctoral, centrándose principalmente en lo que se refiere al desarrollo del aprendizaje por refuerzo. Además, se describirán algunas líneas de investigación que quedan abiertas a partir de esta tesis doctoral, así como algunas líneas nuevas de trabajo futuro. Por último, se enumeran las principales publicaciones obtenidas durante la realización de la tesis doctoral.

8.1. Conclusiones

La conclusión fundamental que se puede extraer de esta tesis doctoral es un nuevo método de aprendizaje por refuerzo para dominios con espacios de estados continuos. Este nuevo método, al que se ha denominado *ENNC-QL*, tiene dos principales ventajas. Por un lado, es un método que permite aprender tareas, en entornos de varias dimensiones, con una exploración limitada, y con una eficacia superior a otros métodos. Por otro lado, el número de parámetros y/o conocimiento que debe suministrarse al método para que funcione correctamente es mínimo, y por tanto, es fácilmente aplicable a nuevos dominios.

El desarrollo de este método tuvo su motivación en la pérdida de la propiedad de Markov y/o la introducción de indeterminismo que se encontraba en los métodos de discretización del espacio de estados, como se mostraba en la utilización del algoritmo VQQL. Este método, que puede considerarse como el primer paso del desarrollo de la tesis doctoral, por sí mismo ofrece muchas ventajas sobre otros métodos anteriores, y ha producido también muy buenos resultados en gran cantidad de dominios, como *CMOMMT*.

Esta tesis doctoral tiene además un caracter multidisciplinar, en el sentido de que, a pesar de que se centra en el aprendizaje por refuerzo, toma ideas, métodos y técnicas de otras disciplinas. En este sentido, se puede destacar el uso de técnicas de aprendizaje no supervisado como el Algoritmo de Lloyd generalizado utilizado en VQQL, la incorporación de técnicas basadas en la computación evolutiva en el algoritmo ENNC, o el uso de métodos de aprendizaje supervisado como aproximador de las funciones de valor en aprendizaje por refuerzo. Además, el algoritmo de clasificación ENNC es un elemento muy importante en el desarrollo de esta tesis doctoral, ya que ha demostrado ser un algoritmo muy útil en muchos dominios de aplicación, incluido el de aproximador de la función de valor-acción, recogido en esta tesis.

8.2. Aportaciones de la Tesis Doctoral

A continuación se detallan las principales aportaciones que introduce esta tesis doctoral al área del aprendizaje por refuerzo.

- **Aproximaciones del prototipo más cercano.**
  A lo largo de esta tesis doctoral se ha planteado el uso de métodos del prototipo más cercano para realizar la discretización del entorno y como aproximador de funciones. Este tipo de aproximaciones se han

mostrado muy útiles y sencillas de aplicar e integrar, tanto en su versión no supervisada con el Algoritmo de Lloyd Generalizado, como en su versión supervisada con el algoritmo ENNC. Se ha mostrado que este tipo de aproximaciones es aplicable en dominios de hasta 6 dimensiones, dominios donde las discretizaciones uniformes o los métodos de discretización variable no son útiles [Munos and Moore, 2002].

- **Adecuación de los límites entre regiones.**
  A la hora de discretizar el entorno, los métodos de resolución variable pueden incrementar enormemente el número de regiones introducidas, ya que encontrar límites o fronteras en la función de valor puede producir un enorme incremento en el número de regiones en esas zonas [Munos and Moore, 2002, Reynolds, 2000]. Por el contrario, métodos que sean capaces de detectar esos límites y definirlos correctamente desplazando dichos límites, y no mediante el incremento de la resolución, pueden ser mucho más eficientes. Esto permite obtener representaciones del espacio de estados más reducidas, y por tanto, el uso más eficiente de la experiencia.

- **Dominios estocásticos y dominios deterministas con ruido.**
  Con el modelo ENNC-QL se ha mostrado que el considerar el indeterminismo de un dominio como ruido de un dominio determinista puede permitir tratar el dominio como determinista, si el aproximador de funciones utilizado es capaz de absorber ese ruido. Esto se ha mostrado muy útil para definir los límites entre las zonas del espacio de estados con bordes en la función de valor y, sobre todo, para obtener las políticas de comportamiento óptimas.

- **Aplicación sobre problemas deterministas y estocásticos.**
  Se ha mostrado que tanto el modelo VQQL como el ENNC-QL son válidos tanto para dominios deterministas como estocásticos, y producen en ambos buenos resultados. En lo que se refiere al algoritmo ENNC-QL esto se debe a que se considera el indeterminismo como ruido, tal y como se ha descrito en el punto anterior.

- **Propiedad de Markov e Indeterminismo.**
  En esta tesis doctoral se ha resaltado la relación entre la pérdida de la propiedad de Markov cuando se utilizan discretizaciones del espacio de estados y la introducción de indeterminismo en dominios que originariamente eran deterministas. De esta forma, la pérdida de la propiedad de Markov puede verse como un problema de introducción de indeterminismo que, además, puede ser tratado de forma eficiente, tal y como

se ha mostrado.

- **Modelo Mixto.**
  Una de las principales características del algoritimo ENNC-QL es que se presenta como un modelo mixto de los métodos basados en aproximadores de funciones con métodos basados en la discretización del espacio de estados, en el que se muestra que ambos pueden complementarse. Este modelo mixto tiene una primera fase típica de la aplicación de métodos supervisados a la aproximación de las funciones de valor, y una segunda fase típica de los métodos basados en la discretización del espacio de estados

- **Discretización por acción.**
  En la mayoría de los trabajos anteriores basados en discretizaciones del espacio de estados se suele utilizar una única discretización del espacio de estados. Con el algoritmo ENNC-QL se plantea que dado que en los métodos libres de modelo es necesario aproximar la función de valor-acción $Q(s, a)$, es posible que cada acción requiera unos recursos distintos en lo que se refiere a la discretización del espacio de estados, y por tanto, que cada una podría requerir una discretización propia. Esto, en unos casos, podría complicar el diseño, ya que se necesita una discretización por acción. Sin embargo, esas discretizaciones probablemente son más sencillas que la que se necesitaría en caso de utilizar sólamente una, que debería ser capaz de representar toda la información requerida por todas las acciones.

8.3. Trabajos y Líneas de Investigación Futuros

Esta tesis doctoral tiene como elemento principal el desarrollo de algoritmos de aprendizaje por refuerzo, como son el algoritmo VQQL y el algoritmo ENNC-QL. Incluye, por tanto, su motivación, descripción y validación como puntos esenciales de su desarrollo. El trabajo desarrollado plantea líneas de trabajo que pueden ser estudiadas, desarrolladas y verificadas en el futuro. A continuación se exponen brevemente algunas de estas posibles líneas de trabajo, algunas de ellas muy relacionadas entre sí.

- **Estrategias de exploración/explotación.**
  Uno de los principales inconvenientes que ofrecen, tanto el algoritmo VQQL como el ENNC-QL, es que requieren de una fase inicial de exploración del dominio, a partir de las cuales generar, respectivamente, el

nuevo espacio de estados, o una primera aproximación de la función de valor-accion. Esto es un inconveniente por dos razones fundamentales. La primera de ellas es que no siempre es posible hacer esta exploración inicial, ya que la obtención de experiencia es muy costosa, y es preferible realizar un mayor aprovechamiento de la experiencia, en forma de estrategias de exploración más avariciosas. En segundo lugar, aunque se pueda hacer la exploración, en ocasiones la relación entre experiencias que reciben un refuerzo positivo, y aquéllas que reciben un refuerzo nulo, está muy desequilibrado hacia el segundo, lo que genera aproximadores que se adaptan a esa situación, y no aprenden de los refuerzos positivos. Estos dos elementos motivan la necesidad de integrar estrategias de exploración más adecuadas, que eviten los problemas antes planteados, y que permitan aprender desde las primeras experiencias.

- **Versión incremental del algoritmo** ENNC-QL.
  El punto anterior motiva el desarrollo de versiones incrementales del algoritmo ENNC-QL, que permitan aprender de la experiencia que se adquiere desde las primeras exploraciones. Una de las posibilidades que se plantean es ejecutar la primera fase iterativa del algoritmo desde que se obtienen las primeras tuplas de experiencia. Las nuevas experiencias se irían acumulando a las anteriores, de forma que se incrementaría poco a poco el número de experiencias con las que se aprende en cada iteración.

- **Uso de otros aproximadores.**
  Siguiendo el esquema de ENNC-QL, también se plantea la sustitución del sistema de clasificación basada en el vecino más cercano, es decir, del algoritmo ENNC, por otro método que mantenga las propiedades de ofrecer un buen aproximador de las funciones de valor y de definir una discretización del espacio de estados. De esta forma, se mantienen las dos fases del algoritmo ENNC-QL, pero utilizando otro método de clasificación supervisada en la primera fase. Uno de los métodos que parecen más apropiados son los árboles de decisión. En los experimentos planteados en esta tesis, se ha mostrado que un algoritmo como C4.5 genera soluciones ya buenas, utilizado únicamente como aproximador. Si los nodos del árbol se liberan de la etiqueta de clase, se pueden entender como regiones del espacio, que pueden ser utilizadas como discretizaciones en la segunda fase de aprendizaje.

8.4. Publicaciones

En esta sección se enumeran las publicaciones obtenidas en relación a esta tesis doctoral. Dichas publicaciones se han dividido en dos grupos. Por un lado, publicaciones o documentos científico-técnicos en general, y por otro, ponencias en congresos o *workshops*.

8.4.1. Publicaciones o Documentos Científico-Técnicos

(CLAVE: CL = capítulo de libro, A = artículo, S = documento científico-técnico restringido.)

Autores (p.o. de firma): Fernando Fernández y Pedro Isasi
Titulo: *Automatic Finding of Good Classifiers Following a Biologically Inspired Metaphor*
Ref. revista/libro: Computing and Informatics
Clave: A Volumen: 21(3) Páginas, inicial: 1 final: 16 Fecha: 2002
Editorial: Slovak Academic Press Ltd.
Lugar de publicación: Eslovaquia

Autores (p.o. de firma): Lynne E. Parker, Claude Touzet y Fernando Fernández
Titulo: *Techniques for Learning in Multi-Robot Teams*
Ref. revista/libro: Robot Teams: From Diversity to Polymorphism
Clave: CL Volumen: Páginas, inicial: 191 final: 236 Fecha: 2002
Editorial: A. K. Peters Publishers Ltd.
Lugar de publicación: Canadá

Autores (p.o. de firma): Fernando Fernández y Pedro Isasi
Titulo: *Evolutionary Design of Nearest Neighbour Classifiers*
Ref. revista/libro: Informes Técnicos
Clave: S Volumen: 1 Páginas, inicial: 271 final: 303 Fecha: 2001
Editorial: Departamento de Informática. Univ. Carlos III de Madrid
Lugar de publicación: España

Autores (p.o. de firma): Fernando Fernández y Lynne E. Parker
Titulo: *Learning in Large Cooperative Domains*
Ref. revista/libro: International Journal on Robotics and Automation
Clave: A Volumen: 16(4) Páginas, inicial: 211 final: 226 Fecha: 2001
Editorial: ACTA Press
Lugar de publicación: Canadá

8.4.2. Contribuciones a Congresos

Autores: Fernando Fernández y Daniel Borrajo
Título: *On Determinism Handling While Learning Reduced State Space Representations*
Congreso: European Conference on Artificial Intelligence (ECAI 2002)
Publicación: Proceedings of the ECAI 2002
Lugar de celebración: Lugano (Suiza)
Fecha: 2002

Autores: Fernando Fernández y Pedro Isasi
Título: *Perspectiva Biológica del Diseño de Clasificadores Basados en la Regla del Vecino más Cercano*
Congreso: Primer Congreso Español de Algoritmos Evolutivos y Bioinspirados (AEB'02)
Publicación: Libro de Actas de AEB'02
Lugar de celebración: Mérida, Badajoz (España)
Fecha: 2002

Autores: Fernando Fernández y Daniel Borrajo
Título: *Reducing the Introduction of Non-Determinism in Reinforcement Learning*
Congreso: Twentieth Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2001)
Publicación: Proceedings of the PLANSIG 2001
Lugar de celebración: Edimburgo (Reino Unido)
Fecha: 2001

Autores: Fernando Fernández y Pedro Isasi
Título: *Designing Nearest Neighbour Classifiers by the Evolution of a Population of Prototypes*
Congreso: 9^{th} European Symposium on Artificial Neural Networks (ESANN'01)
Publicación: Proceedings of ESANN'01
Lugar de celebración: Brujas (Bélgica)
Fecha: 2001

Autores: Fernando Fernández y Daniel Borrajo
Título: *Aprendizaje de Habilidades Mediante la Reducción del Entorno y la Explotación*
Congreso: Workshop Hispano-Luso en Agentes Físicos
Publicación: Actas del primer Workshop Hispano-Luso en Agentes Físicos
Lugar de celebración: Tarragona (España)
Fecha: 2000

Autores: Fernando Fernández y Daniel Borrajo
Título: *Iterative VQQL for Learning Skills*
Congreso: Learning'00
Publicación: Proceedings of Learning'00
Lugar de celebración: Leganés, Madrid (España)
Fecha: 2000

Bibliografía

[Aha and Kibler, 1991] D. Aha and K. Kibler. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[Aha, 1997] David Aha. *Lazy Learning.* Kluwer Academic Publishers, May 1997.

[Albus, 1971] J.S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61, 1971.

[Albus, 1981] J.S. Albus. *Brain, Behaviour, and Robotics.* Byte Books, Peterborough, NH, 1981.

[Asada *et al.*, 1996] Minoru Asada, Shoichi Noda, and Koh Hosoda. Action-based sensor space categorization for robot learning. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'96)*, volume 3, pages 1502–1509, 1996.

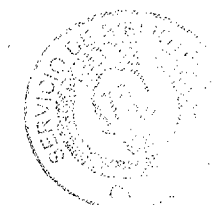
[Baird and Klopf, 1993] L.C. Baird and A.H. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical report, Wright-Patterson Air Force Base Ohio: Wright Laboratory, 1993.

[Barto *et al.*, 1995] A. G. Barto, S.J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 1(72), 1995.

[Bellman and Kalaba, 1965] R. Bellman and R. Kalaba. *Dynamic Programming and Modern Control Theory.* Academic Press, 1965.

[Bellman, 1957] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ., 1957.

[Bermejo and Cabestany, 2000] Sergio Bermejo and Joan Cabestany. A batch learning algorithm vector quantization algorithm for nearest neighbour classification. *Neural Processing Letters*, 11:173–184, 2000.



[Bertsekas and Tsitsiklis, 1996] D. P. Bertsekas and J.Ñ. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Bellmon, Massachusetts, 1996.

[Bezdek and Kuncheva, 2001] James C. Bezdek and Ludmila I. Kuncheva. Nearest neighbour classifier designs: An experimental study. *International Journal Of Intelligent Systems*, 16:1445–1473, 2001.

[Bezdek *et al.*, 1998] James C. Bezdek, Thomas R. Rechherzer, Gek Sok Lim, and Yianni Attikiouzel. Multiple-prototype classifier design. *IEEE Transactions on Systems, Man and Cybernetics*, 28(1):67–79, February 1998.

[Blake and Merz, 1998] C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.

[Boyan and Moore, 1995] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems*, 7, 1995.

[Brieman *et al.*, 1984] L. Brieman, J. H. Friedman, R. A. Olshen, and C. H. Stone. *Classification and Regression Trees.* Wadsworth, 1984.

[Broomhead and Lowe, 1988] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 1988.

[Burrascano, 1991] Pietro Burrascano. Learning vector quantization for the probabilistic neural network. *IEEE Transactions on Neural Networks*, 2(4):458–461, July 1991.

[Cagnoni and Valli, 1994] S. Cagnoni and G. Valli. OSLVQ: A training strategy for optimum-size learning vector quantization classifiers. In *IEEE International Conference in Neural Networks*, pages 762–775, 1994.

[Chapman and Kaelbling, 1991] David Chapman and Leslie P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Sydney (Australia), 1991.

[Claussen *et al.*, 1999] C. Claussen, S. Gutta, and H. Wechsler. Reinforcement learning using functional approximation for generalization and their application to cart centering and fractal compression. In Thomas Dean, editor, *Proceedings of Sixteenth International Joint Coference on Artificial Intelligence*, volume 2, pages 1362–1367, Stockholm, Sweden, August 1999.

[Dayan and Hinton, 1993] P. Dayan and G. E. Hinton. Feudal reinforcement learning. *Advances in Neural Information Processing Systems*, 5, 1993.

[Dayan, 1992] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362, 1992.

[Dietterich, 2000] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[Duda and Hart, 1973] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley And Sons, 1973.

[Fernández and Borrajo, 2000] Fernando Fernández and Daniel Borrajo. VQQL. Applying vector quantization to reinforcement learning. In *RoboCup-99: Robot Soccer World Cup III*, number 1856 in Lecture Notes in Artificial Intelligence, pages 292–303. Springer Verlag, 2000.

[Fernández, 1999] Fernando Fernández. VQQL: Un modelo de aprendizaje por refuerzo para dominios continuos e indeterministas. Proyecto Fin de Carrera en Ingeniería Informática, Departamento de Informática, Universidad Carlos III de Madrid, Leganés, Madrid, 1999.

[Frank and Witten, 1998] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learnin*, 1998.

[Fritzke, 1994] Bernd Fritzke. Growing cell structures -a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

[Gersho and Gray, 1992] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[Geva and Sitte, 1991] S. Geva and J. Sitte. Adaptive nearest neighbour pattern classification. *IEEE Transactions on Neural Networks*, 2(2):318–322, March 1991.

[Gullapalli, 1992] V. Gullapalli. *Reinforcement Learning and its application to control.* PhD thesis, University of Massachusetts, Amherst,MA., 1992.

[Hart, 1968] P. E. Hart. The condensed nearest neighbour rule. *IEEE Transactions on Information Theory*, 1968.

[Haykin, 1988] Simon Haykin. *Digital Communications.* Wiley, 1988.

[Hinton, 1984] G. E. Hinton. Distributed representations. Technical report, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1984.

[Holland, 1975] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press. (Second Edition: MIT Press), 1975.

[Howard, 1960] R. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA, 1960.

[Jaakkola *et al.*, 1994] T. Jaakkola, T. Jordan, and S. P. Singh. On the convergence of shochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

[John and Langley, 1995] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, 1995.

[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, 1996] L. P. Kaelbling. *Recent Advances in Reinforcement Learning.* Kluver, 1996.

[Kanerva, 1988] P. Kanerva. *Sparse Distributed Memory.* MIT Press, 1988.

[Kitano *et al.*, 1997] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada. The Robocup synthetic agent challenge. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 24–49, San Francisco, CA, 1997.

[Kohonen, 1984] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer, Berlin, Heidelberg, 1984. 3rd ed. 1989.

[Kohonen, 1995] Teuvo Kohonen. *Self-Organizing Maps.* Springer, Berlin, Heidelberg, 1995. (Second Extended Edition 1997).

[Kumar, 1986] P. R. Kumar. *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Prentice Hall, 1986.

[Kuncheva and Bezdek, 1998] Ludmila I. Kuncheva and James C. Bezdek. Nearest prototype classification: Clustering, genetic algorithms, or random search? *IEEE Transactions on Systems, Man and Cybernetics*, 28(1):160–164, February 1998.

[Lin, 1991] L.-J Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the Nineth National Conference on Artificial Intelligence*, 1991.

[Linde *et al.*, 1980] Yoseph Linde, André Buzo, and Robet M. Gray. An algorithm for vector quantizer design. In *IEEE Transactions on Communications, Vol1. Com-28, Nº 1*, pages 84–95, 1980.

[Lloyd, 1957] S. P. Lloyd. Least squares quantization in PCM. Unpublished Bell Laboratories Technical Note. Portions presented at the Institute of Mathematical Statistics Meeting, September 1957.

[Lloyd, 1982] S. P. Lloyd. Least squares quantization in PCM. In *IEEE Transactions on Information Theory*, number 28 in IT, pages 127–135, March 1982.

[Mahadevan and Connell, 1992] S. Mahadevan and J. Connell. Automatic programming of behaviour-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3):311–365, June 1992.

[Mao *et al.*, 2000] K. Z. Mao, K.-C. Tan, and W. Ser. Probabilistic neural-network structure determination for pattern classification. *IEEE Transactions on Neural Networks*, 11(4):1009–1016, July 2000.

[Merelo *et al.*, 1998] J. J. Merelo, A. Prieto, and F. Morán. Optimization of classifiers using genetic algorithms. In Patel Honavar, editor, *Advances in Evolutionary Synthesis of Neural Systems*. MIT press, 1998.

[Michie and Chambers, 1968] D. Michie and R. A. Chambers. BOXES: An experiment in adaptive control. *Machine Intelligence*, 2:137–152, 1968.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[Moore and Atkeson, 1993] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13, 1993.

[Moore and Atkeson, 1995] A. W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3):199–233, 1995.

[Moore, 1991] Andrew W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces. *Proceedings in Eighth International Machine Learning Workshop*, 1991.

[Munos and Moore, 1999] Rémi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In Thomas Dean, editor, *Proceedings of Sixteenth International Joint Coference on Artificial Intelligence*, volume 2, pages 1348–1355, Stockholm, Sweden, August 1999.

[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.

[Noda *et al.*, 1998] Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multi-agent systems. *Applied Artificial Intelligence*, 12(2-3):233–250, 1998.

[Pal *et al.*, 1993] Nikhil R. Pal, James C. Bezdek, and Eric C. K. Tsao. Generalized clustering networks and Kohonen's self-organizing scheme. *IEEE Transactions on Neural Networks*, 4(4):1993, July 1993.

[Parker and Touzet, 2000] Lynne Parker and Claude Touzet. Multi-robot learning in a cooperative observation task. In L.E. Parker, G. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotic Systems*, volume 4, pages 391–401. Springer, 2000.

[Parker, 1999] L. E. Parker. A case study for life-long learning and adaptation in cooperative robot teams. In *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, pages 92–101, 1999.

[Patanè and Russo, 2001] G. Patanè and M. Russo. The enhanced LBG algorithm. *Neural Networks*, 14:1219–1237, 2001.

[Patanè and Russo, 2002] G. Patanè and M. Russo. Fully automatic clustering system. *IEEE Transactions on Neural Networks*, 13(6):1285–1298, November 2002.

[Peng and Williams, 1993] J. Peng and R. J. Williams. Efficient learning and planning with the Dyna framework. *Adaptive Behaviour*, 1(4), 1993.

[Pérez and Vidal, 1993] J. C. Pérez and Enrique Vidal. Constructive design of LVQ and DSM classifiers. In J. Mira, J. Cabestany, and A. Prieto, editors, *New Trends in Neural Computation*, volume 686 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.

[Puterman, 1994] M. L. Puterman. *Markov Decision Processes - Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY., 1994.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Reynolds, 2000] S. I. Reynolds. Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *Proceedings of International Conference on Machine Learning*, 2000.

[Rummery and Niranjan, 1994] G. A. Rummery and M.Ñiranjan. On-line Q-learning using connectionist systems. Technical report, Cambridge University, 1994.

[Rummery, 1995] G. A. Rummery. *Problem Solving with Reinforcement Learning.* PhD thesis, Cambridge University, 1995.

[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(2):163–218, 1998.

[Sawada *et al.*, 1999] Tsutomu Sawada, Sumiaki Ichikawa, and Fumio Hara. Autonomous action-mode change in a two-mobile robotics system. s-temperature based on-line learning. In *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'99)*, volume 1, pages 393–399, 1999.

[Smart and Kaelbling, 2000] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the International Conference of Machine Learning*, pages 903–907, 2000.

[Smart, 2002] W. D. Smart. *Making Reinforcement Learning Work on Real Robots.* PhD thesis, Department of Computer Science at Brown University, Providence, Rhode Island, May 2002.

[Specht, 1990] D. F. Specht. Probabilistic neural networks. *Neural Networks*, 3(1):109–118, 1990.

[Stone and Sutton, 2001] P. Stone and R. S. Sutton. Scalling reinforcement learning toward robocup soccer. In *Proceedings of the 18th International Conference on Machine Learning*, 2001.

[Stone, 2000] P. Stone. *Layered Learning in Multiagent Systems.* MIT Pres, 2000.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, Massachusetts, 1998.

[Sutton, 1988] R. S. Sutton. Learning to predict by the method of temporal difference. *Machine Learning*, 3:9–44, 1988.

[Sutton, 1990] R. S. Sutton. Integrated arquitectures for learning, planning, and reacting based on approximating dynamic programming. In *Seventh International Conference on Machine Learning*, 1990.

[Sutton, 1991] R. S. Sutton. Planning by incremental dynamic programming. In *Eighth International Workshop on Machine Learning*, pages 353–357, 1991.

[Sutton, 1992] R. S. Sutton. *Reinforcement Learning.* Kluwer, 1992.

[Tesauro, 1992] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[Thorndike, 1911] E. L. Thorndike. *Animal Intelligence.* Hafner, Darien, CT, 1911.

[Touzet, 1997] Claude Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22:251–281, 1997.

[Tsitsiklis and Roy, 1996] J.Ñ. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

[Uchibe, 1999] E. Uchibe. *Cooerative Behavior Acquisition by Learning and Evolution in a Multi-Agent Environment for Mobile Robots.* PhD thesis, Osaka University, 1999.

[Ueno *et al.*, 1996] A. Ueno, K. Hori, and S. Nakasuda. Simultaneous learning of situation classification based on rewards and behavior selection based on the situation. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'96)*, volume 3, pages 1510–1517, 1996.

[Watkins and Dayan, 1992] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[Watkins, 1989] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, UK, 1989.

[Whitehead, 1991] S. D. Whitehead. Complexity and cooperation in Q-learning. In Morgan Kauffman, editor, *Proceedings of the Eight International Workshop on Machine Learning*, 1991.

[Witten and Frank, 2000] I. H. Witten and E. Frank. *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 2000.

[Yao, 1993] X. Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8:539–567, 1993.

[Zhao and Higuchi, 1996] Q. Zhao and T. Higuchi. Evolutionary learning of nearest neighbour MLP. *IEEE Transactions on Neural Networks*, 7(3):762–767, 1996.