UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA

Departamento de Computación

TESIS DOCTORAL

MDB: MECANISMO COGNITIVO DARWINISTA PARA AGENTES AUTÓNOMOS

Autor: Francisco Javier Bellas Bouza

Director: Richard Duro Fernández

Fecha: Julio de 2003

Agradecimientos

En primer lugar, quiero agradecer muy especialmente a mi director de Tesis, Richard Duro Fernández, por su encomiable labor de dirección a nivel formativo y humano, logrando, con un permanente trato de igualdad, que en estos años haya desarrollado verdadero entusiasmo por la investigación en el campo de la Inteligencia Artificial. Debo resaltar el esfuerzo que ha realizado para financiar mi trabajo, cumpliendo en todo momento y a pesar de las dificultades, las promesas a este respecto.

A mis compañeros del laboratorio del Grupo de Sistemas Autónomos, con los que he compartido todo lo bueno y lo malo que implica el nacimiento de un grupo de investigación, y que ha desembocado en verdadera amistad. Los mejores recuerdos de esta tesis los guarda el laboratorio de la Escuela Politécnica Superior.

A mi mujer, Marta, por respetar y entender mi lucha, a veces frustrante, defendiendo que la investigación es una forma muy digna de ganarse la vida. Sobre todo, le agradezco su paciencia y comprensión estos caóticos últimos seis meses que trataré de compensar en adelante.

A mi madre, por mantenerme durante 25 años como sólo una madre sabe hacerlo.

Publicaciones

Durante la realización de este trabajo se han llevado a cabo las publicaciones científicas que se enumeran a continuación:

- J. Santos, R. J. Duro, J. A. Becerra, J. L. Crespo and F. Bellas (2000) “Aspects of evolution for obtaining real robot controllers” *Proceedings FEA 2000*, pp 1023-1026, ISBN 0-9643456-9-2.
- R. J. Duro, J. Santos, J. A. Becerra, F. Bellas and J. L. Crespo (2000) “Using Higher Order Synapses and Nodes to Improve the Sensing Capabilities of Mobile Robots” *Proceedings ESANN 2000*, pp 81-88, ISBN 2-930307-00-5.
- F. Bellas, J. A. Becerra, J. Santos and R. J. Duro (2000) "Applying Synaptic Delays for Virtual Sensing and Actuation in Mobile Robots" *Proceedings CD-ROM IJCNN 2000,* ISBN 0-7695-0619-4.
- R. J. Duro, J. Santos, F. Bellas, A. Lamas (2000) "On Line Darwinist Cognitive Mechanism for an Artificial Organism" *Proceeding Supplement Book SAB2000,* pp 215-224, The International Society for Adaptive Behavior, ISBN 0-9704673-0-3.
- J. Santos, R. J. Duro, J.A. Becerra, J.L. Crespo, F. Bellas (2001) “Considerations in the Application of Evolution to the Generation of Robot Controllers” *Information Science vol. 133*, pp 127-148, ISBN 0020-0255-01.
- F. Bellas, J.A. Becerra, R. J. Duro (2001) "Using evolution for thinking and deciding" *Advances in Fuzzy Systems and Evolutionary Computation,* pp 242-247, ISBN 960-8052-27-0.
- F. Bellas, A. Lamas, R. J. Duro (2001) "Adaptive behavior through a Darwinist machine" *Advances in Artificial Life. Proc. 6th European Conference on Artificial Life*. Jozef Kelemen and Petr Sosik editors, pp 86-90, ISBN 3-540-42567-5.
- F. Bellas, A. Lamas, R.J. Duro (2001) "Evolutionary Cognition for Autonomously Obtaining Primal Behaviors" *Advances in Signal Procesing, Robotics and Communications,* pp398-404, ISBN 960-8052-42-4.
- F. Bellas, A. Lamas, R.J. Duro (2001) "Multilevel Darwinist Brain and Autonomously Learning to Walk" *Proceedings CIRAS 2001,* pp 392-396, ISSN 0219-6131.
- F. Bellas, R. J. Duro, J.A. Becerra (2002) “Funciones de calidad muestreadas en algoritmos evolucionistas” *Actas del primer congreso español de algoritmos evolutivos y bioinspirados,* pp 375-382, ISBN 84-607-3913-9.
- F. Bellas, J.A. Becerra, R. J. Duro (2002) “Sampled fitness functions in complex problems(part I): Influence of short term memory size” *Proceedings 6th joint Conference on Information Science,* pp 631-634, ISBN 0-9707890-1-7.
- F. Bellas, J.A. Becerra, R. J. Duro (2002)“Sampled fitness functions in complex problems(part II): Critical points and constructing” *Proceedings 6th joint conference on information science,* pp 635-638, ISBN 0-9707890-1-7.
- F. Bellas, R. J. Duro “Statistically neutral promoter based GA for evolution with dynamic fitness functions” (2002) *Proceedings of the 2nd IASTED international conference,* pp 335-340, ISBN 0-88986-352-0.

- F. Bellas, R. J. Duro (2002) “Modelling the world with statistically neutral PBGAs. Enhancement and real applications” *Proceedings of the 9th international conference on Neural Information Processing,* pp 2093-2098, ISBN 981-04-7536-5.
- F. Bellas, R. J. Duro (2003) “Generación Automática de Modelos de Mundo mediante Algoritmos Genéticos basados en genes promotores” *Actas del segundo congreso español sobre Metaheurísticas, Algoritmos evolutivos y bioinspirados,* pp101-108, ISBN 84-607-65-26-1.
- F. Bellas, R. J. Duro (2003) “Introducing Long Term Memory in an ANN based Multilevel Darwinist Brain” *Computational Methods in Neural Modeling*, pp 590-598, ISBN 3-540-40210-1.
- F. López Peña, F. Bellas, R. J. Duro, M. Sánchez (2003) "Reconstructing Irregularly Sampled Laser Doppler Velocimetry Signals by Using Artificial Neural Networks*", Proceedings IDAACS 2003*, pp 99-105, ISBN 0-7803-8138-6.

Índice

Introducción

1 Introducción

El presente trabajo se enmarca en el campo de las Ciencias Cognitivas y los Agentes Inteligentes y en él presentamos un Mecanismo Cognitivo Darwinista para Agentes Autónomos denominado **MDB (Multilevel Darwinist Brain)** que ha sido desarrollado en el Grupo de Sistemas Autónomos de la Universidade da Coruña.

El MDB es un Mecanismo Cognitivo diseñado para permitir que cualquier tipo de agente se adapte a su entorno y a sus motivaciones de forma autónoma. Los posibles cambios, tanto externos como internos, conllevan la necesidad de que el agente produzca soluciones originales y creativas para satisfacer sus objetivos. El MDB dotará a un determinado agente de las capacidades de autonomía e inteligencia que lo catalogan como Agente Autónomo Inteligente (AAI), plataforma sobre la que será aplicado el mecanismo. La principal característica del MDB reside en que la adaptación del agente ante posibles cambios en su entorno o en él mismo es totalmente autónoma.

La estructura general del MDB se concreta en una jerarquía de dos niveles:

- **Nivel de razonamiento** donde se resuelve de forma continua un problema de aprendizaje de modelos de mundo y modelos internos que son utilizados en un proceso de selección de acciones que debe ser optimizado.
- **Nivel de interacción** donde la acción es aplicada al entorno obteniendo información que es realimentada al nivel de razonamiento.

Esta estructura corresponde a un modelo cognitivo genérico donde internamente se trabaja con *modelos de mundo* (que relacionan las percepciones del entorno en un instante de tiempo con las percepciones en el instante siguiente tras la realización de una acción), *modelos internos* (que relacionan las percepciones tras la realización de una acción con el grado de satisfacción de los objetivos) y *estrategias* (secuencias de acciones que se aplican en el entorno) que se escogen para llevar a cabo los objetivos marcados. En la interacción con el entorno real se comprueba si la estrategia adoptada ha sido correcta, y el resultado que ésta produce proporciona una nueva muestra representativa de la relación agente-entorno que nos permite ir perfeccionando los modelos en tiempo de ejecución. Se hace necesaria, por tanto, la utilización de un mecanismo que gestione las muestras que se van obteniendo y su almacenamiento en algún tipo de memoria.

La motivación que subyace al abordar el diseño de un Mecanismo Cognitivo es dotar a un sistema artificial de un funcionamiento autónomo inteligente tomando como referencia el comportamiento de ciertos seres vivos. En este sentido, estableceremos una clara distinción entre la operación externa (objetiva) e interna (subjetiva) de un agente, de tal modo que sea la combinación de la información del entorno y del estado interno la que le lleve a tomar una determinada decisión.

El trabajo se ha estructurado en seis partes fundamentales, cada una de la cuales contiene distintos subapartados que serán desarrollados en profundidad:

1. **Definición de conceptos básicos y objetivos:** donde definimos los principales conceptos en que se fundamenta el estudio y establecemos con claridad los objetivos de esta Tesis Doctoral.
2. **Estado actual del tema:** en este apartado se realiza una revisión del estado actual de la investigación en el campo de los mecanismos y modelos

cognitivos, centrándonos en su aplicación a los Agentes Autónomos Inteligentes.

3. **Fundamentos de un Mecanismo Cognitivo:** donde se establecerán los elementos básicos de un modelo cognitivo y se realizará un planteamiento del problema matemático a resolver.
4. **Implementación del MDB:** se desarrolla la implementación práctica del mecanismo justificando la utilización de las distintas técnicas de cómputo. Este apartado engloba la parte fundamental del trabajo de investigación llevado a cabo.
5. **Ejemplos de aplicación del MDB:** se muestra el funcionamiento del MDB aplicado a agentes autónomos reales.
6. **Conclusiones y trabajo futuro:** se hace un balance final del trabajo y del grado de cumplimiento de los objetivos iniciales. Además, se plantean las líneas de desarrollo para el futuro.

Definición de conceptos básicos y objetivos

2 Definición de conceptos básicos

Antes de plantear los objetivos de esta Tesis Doctoral, debemos definir con claridad el marco en el que nos movemos, es decir, qué es un Mecanismo Cognitivo y dónde será aplicado, esto es, qué es un Agente Autónomo Inteligente.

2.1 Mecanismo Cognitivo

La comprensión del concepto de Mecanismo Cognitivo es intuitiva, pero su definición unívoca es compleja. De hecho, esta definición es diferente dependiendo del punto de vista desde el que se enfoque:

Teoría de Sistemas: es un sistema realimentado que proporciona una salida o respuesta no predeterminada a una entrada o condición de funcionamiento. Lo más relevante en esta definición reside en que la respuesta es no predeterminada y, en distintos instantes de tiempo ante las mismas entradas, puede ser diferente por la existencia del estado interno (que no es una entrada explícita al sistema).

Computación: es un autómata de estados finitos donde los estados y las conexiones entre dichos estados son adaptativos. El tipo de Mecanismo Cognitivo que trataremos en este trabajo, se construye con un lenguaje de programación computacional, de modo que puede ser representado mediante un autómata de estados finitos. El hecho de que las conexiones y los estados cambien, refleja el proceso de aprendizaje. En este caso, el autómata sí necesita todas las entradas para poder obtener una salida, por lo que desde el punto de vista computacional, el estado interno debe ser conocido.

Ciencias Cognitivas: es un esquema de funcionamiento mental, es decir, un patrón de aprendizaje y razonamiento. Es en este campo donde más relevancia cobran las motivaciones subjetivas del comportamiento reflejadas por el estado interno.

Vida Artificial: es un mecanismo que proporciona a un agente artificial la capacidad de aprender de forma autónoma a desenvolverse en su entorno para llevar a cabo algún tipo de tarea utilizando la experiencia adquirida. Desde este punto de vista, realiza la función de un cerebro.

Estas cuatro definiciones engloban las principales características del mecanismo que se desarrolla en esta Tesis Doctoral y, a lo largo del trabajo, se hará uso de cada una de ellas dependiendo de la fase del estudio en que nos encontremos.

2.2 Agente Autónomo Inteligente

Al igual que en la definición de Mecanismo Cognitivo, el concepto de Agente Autónomo Inteligente (AAI) es susceptible de ser definido desde distintos puntos de vista en función del área de aplicación. Aún así hemos podido establecer una definición general e independiente del dominio de aplicación:

> *Un Agente Autónomo Inteligente es un sistema computacional independiente dotado de sensores y actuadores que habita en un entorno dinámico y que posee la capacidad de razonar y adaptarse a condiciones internas y externas cambiantes.*

En esta definición se incluye la noción de agente autónomo como un sistema computacional artificial independiente dotado de capacidades sensoriales y actuadoras

propias, de tal forma que no requiere ningún tipo de supervisión por parte del diseñador o del usuario. Además, hemos incluido la capacidad adaptativa del agente que refleja la posibilidad de aprender de forma autónoma a partir de su propia experiencia y así poder resolver de forma original los problemas que se le presentan. Esto incorpora la característica de inteligencia al agente autónomo.

3 Objetivos

El principal objetivo de esta Tesis Doctoral es el siguiente:

> *Desarrollar un Mecanismo Cognitivo que pueda ser aplicado a cualquier agente de modo que le proporcione capacidades adaptativas y de razonamiento inteligente de forma autónoma.*

Para llevarlo a cabo, nos planteamos una serie de objetivos parciales, que detallamos a continuación:

1. Establecer con claridad la estructura de un Mecanismo Cognitivo que permita a cualquier agente aprender de su experiencia obteniendo soluciones originales de manera totalmente autónoma a los problemas que se le presentan cuando intenta satisfacer sus motivaciones. Para ello, debemos alcanzar los siguientes subobjetivos:
   a) Encontrar los elementos básicos que forman parte de un proceso de razonamiento de alto nivel.
   b) Relacionar dichos elementos básicos entre sí creando un esquema de funcionamiento cognitivo.
   c) Formalizar matemáticamente el problema global de optimización de alto nivel que pretendemos abordar, formalizando para ello tanto los elementos básicos del mecanismo como su interrelación.
2. Implementar computacionalmente el conjunto del mecanismo desarrollado con anterioridad dotándolo de capacidad práctica real. A su vez, este objetivo, consta de los siguientes subobjetivos:
   a) Encontrar, entre las existentes en el campo de las Ciencias Cognitivas y la Inteligencia Artificial, la técnica computacional más adecuada tanto para el sustrato de los elementos básicos que conforman el Mecanismo Cognitivo como para el proceso de optimización global que se plantea. Para ello analizaremos las virtudes y deficiencias de las distintas opciones existentes.
   b) Comprobar el funcionamiento de cada uno de los elementos del mecanismo por separado estableciendo el grado de influencia de unos sobre otros y su relevancia en la aplicación conjunta.
3. Probar el mecanismo en un agente real. Para ello utilizaremos robots autónomos móviles que deberán llevar a cabo una cierta tarea aprendiendo de su interacción con el entorno. Hemos escogido un robot móvil como plataforma por ser un sistema con alto grado de autonomía (sobre todo de movimiento) y a la vez por sus características sensoriales y de actuación en entornos reales, que los convierte en uno de los agentes autónomos de más compleja implantación.

   El objetivo global implica que el robot ha de poder generar de forma autónoma modelos de mundo y modelos internos, así como poder inducir estos últimos a través de un aprendizaje utilizando algún tipo de modelo de comunicación.

Estado actual

4 Estado actual

En este apartado realizaremos un repaso de los principales trabajos llevados a cabo por la comunidad científica en el campo de los agentes, y de los intentos realizados tratando de dotarlos de autonomía y capacidades inteligentes. Hemos estructurado el estudio de tal forma que, en primer lugar, llevaremos a cabo una revisión del campo de los agentes desde su aparición hasta la actualidad, centrándonos en los agentes autónomos inteligentes y analizando las distintas aproximaciones posibles para proporcionarles capacidades de alto nivel. A continuación, llevaremos el planteamiento hacia la robótica autónoma y, en este campo, realizaremos una revisión del estado actual de los trabajos más relevantes utilizando Mecanismos Cognitivos.

4.1 Agentes Autónomos Inteligentes (AAI)

A continuación centraremos nuestro estudio situándolo en el marco actual del campo de los Agentes Autónomos Inteligentes. Para ello, lo hemos estructurado en cuatro subapartados: definición, tipos de agentes, aplicaciones y taxonomía propia.

4.1.1 Definición de agente

Al abordar la lectura de los trabajos realizados hasta el momento en el campo de los agentes, encontramos que cada investigador utiliza una definición de agente adecuada a las características de su aplicación particular, por lo que se complica un poco la labor de centrar el tema que vamos a tratar. Los primeros trabajos en el campo surgieron como una generalización del concepto de programa informático tratando de dotarlo de capacidades autónomas e inteligentes. Así, los primeros artículos donde se menciona el concepto de agente que utilizaremos en esta introducción, están relacionados con el uso de agentes software, que serán analizados con detenimiento en el siguiente apartado. En este campo, cabe destacar el trabajo del Grupo de Agentes Software del MIT que dirige la Dra. Patti Maes y que, desde principios de los años 90 hasta la actualidad, ha trabajado en el uso de agentes aplicados a filtrado de información personal, compra-venta automática y, en general, a la utilización de agentes software en comercio electrónico y transacciones por internet. En el libro [Maes, 90] se realiza la siguiente definición de agente:

> *Los Agentes Autónomos son sistemas computacionales que habitan en un entorno dinámico complejo, sienten y actúan de forma autónoma en este entorno y de este modo llevan a cabo un conjunto de objetivos o tareas para los que han sido diseñados.*

En esta definición, se incluyen una serie de términos fundamentales como son el concepto de sistema computacional, que lo diferencia de otro tipo de sistema biológico y lo clasifica dentro de los seres artificiales, la condición de que el entorno sea dinámico complejo, que sitúa al agente en un ambiente real, la capacidad de sentir y actuar de forma autónoma, que lo diferencia de los programas informáticos simples y, por último, la existencia de un objetivo que guía el funcionamiento. Los trabajos de este grupo de investigación en el campo de los agentes son muchos y muy relevantes, y en la actualidad llevan a cabo multitud de proyectos asociados con los agentes (APT Decision, Development e-commerce, Electronic Profiles, etc). Uno de los integrantes de dicho grupo, Leonard Foner, propone en [Foner, 97] otra definición de agente basándose en las características de un ser virtual llamado *Julia*, que cumple todos los

requisitos que Foner considera imprescindibles para un agente. Entre ellos, se encuentran algunos de alto nivel de razonamiento como la capacidad de asumir riesgos y la certeza de lograr el objetivo. En este trabajo se le otorga una alta relevancia a la capacidad de comunicación con otros agentes y con el usuario.

Otro grupo que ha trabajado de forma intensa en el campo de los agentes es el Laboratorio de Sistemas del Conocimiento de la Universidad de Stanford, donde la Dra. Barbara Hayes-Roth trabaja en el desarrollo de "mentes software" para personajes virtuales que poseen capacidades humanas como identidad y personalidad. Cada uno de estos personajes se representa mediante un agente y la definición de éste que Hayes-Roth proporciona en [Hayes-Roth, 95] es la siguiente:

> *Los agentes inteligentes realizan continuamente 3 funciones: percepción de las condiciones dinámicas del entorno, actuación que afecta las condiciones del entorno y razonamiento para interpretar las percepciones, resolver los problemas, extraer conclusiones y escoger acciones.*

Vemos que la idea general de agente es similar a la planteada por Maes, aunque es más concreta desde el punto de vista inteligente, ya que enumera las partes básicas del proceso de razonamiento. La principal novedad que aporta es que el entorno se modifica tras la actuación, de tal forma que una acción que en el pasado resultó satisfactoria no tiene por qué serlo en el futuro y se requiere un proceso de razonamiento en tiempo real.

En el Grupo de Agentes, Inteligencia y Multimedia de la Universidad de Southampton, han estado desarrollando técnicas para aplicaciones reales de los agentes en control de procesos, comercio electrónico, administración de redes de telecomunicaciones, etc. El investigador principal de este grupo es el Dr Nick Jennings y en el artículo [Jennings, 95] realizan la siguiente definición de agente:

> *Sistema computacional hardware basado en software que posee las siguientes propiedades:*
>
> *Autonomía: los agentes operan sin la intervención directa de los humanos y tienen algún tipo de control sobre sus acciones y estado interno.*
>
> *Sociabilidad: los agentes interactúan con otros agentes (y posiblemente con humanos) por medio de algún tipo de lenguaje de comunicación de agentes.*
>
> *Reactividad: los agentes perciben su entorno (que puede ser el mundo físico, un usuario mediante un interfaz gráfico de usuario, un conjunto de agentes, internet o una combinación de todos) y responden de forma temporal a los cambios que ocurren en él.*
>
> *Proactividad: los agentes no actúan simplemente en respuesta a su entorno, son capaces de mostrar comportamientos dirigidos por un objetivo tomando la iniciativa.*

En esta definición se incluyen los mismos conceptos básicos que en las dos anteriores y, además, se incluye un factor social de convivencia entre agentes que implica una capacidad de comunicación e incluso la necesidad de utilizar un lenguaje propio.

En la línea de estos tres autores, desde mediados hasta finales de los años 90, se han publicado distintos artículos y libros que sentaban las bases del campo de los agentes inteligentes desde el punto de vista teórico y conceptual. Además de los mencionados, otros como [Woolridge, 95], [Etzioni, 95], [Nwana, 96], [Brenner, 98] tratan de sentar las bases y resumir los distintos trabajos realizados hasta ese instante pero sin entrar en aplicaciones prácticas concretas. De todos ellos, destacamos el artículo [Franklin, 96] de Stan Franklin y Art Graesser, director el primero del Instituto de Sistemas Inteligentes de la Universidad de Memphis. Este estudio es uno de los más relevantes a la hora de concretar el concepto de agente y aporta una de las definiciones más generales que se han dado:

> *Un agente autónomo es un sistema situado en y parte de un entorno que siente ese entorno y actúa sobre él, a través del tiempo, persiguiendo sus propios objetivos de forma que afecte lo que siente en el futuro.*

Esta definición es muy general y puede ser aplicada a sistemas muy diferentes. Por ejemplo, en el nivel más alto tenemos a los seres humanos y a ciertos animales con objetivos, sensorizaciones, actuaciones y estructuras de control muy complejas, y en el nivel más bajo podemos encontrar un simple termostato (o una bacteria), que también se ajustan a la definición de agente anterior.

Como resumen, podemos decir que existen una serie de características básicas que todos los agentes autónomos inteligentes han de cumplir en mayor o menor medida, extraídas del análisis de los distintos trabajos y recopiladas en [Bradshaw, 97]:

- **Reactividad:** habilidad para sensar y actuar de forma selectiva.
- **Autonomía:** sensorial y actuadora guiada por un objetivo.
- **Comportamientos colaboradores:** con otros agentes para lograr un bien común.
- **Capacidad de comunicación a nivel de conocimiento:** a un nivel más alto que los protocolos programa-programa basados en un lenguaje de símbolos.
- **Capacidad inferencial:** para ir más allá de la información que obtiene por los sensores creando modelos de sí mismo, del usuario, de otros agentes y del entorno.
- **Continuidad temporal:** persistencia de la identidad y el estado en el tiempo.
- **Personalidad:** capacidad de mostrar características propias de un comportamiento subjetivo, como la emoción.
- **Adaptabilidad:** capacidad de aprender y mejorar con la experiencia.
- **Movilidad:** capacidad de migrar de forma directa de una plataforma a otra.

A continuación, pasamos a revisar las distintas clasificaciones existentes sobre los agentes autónomos inteligentes y a analizar las características de cada tipo concreto.

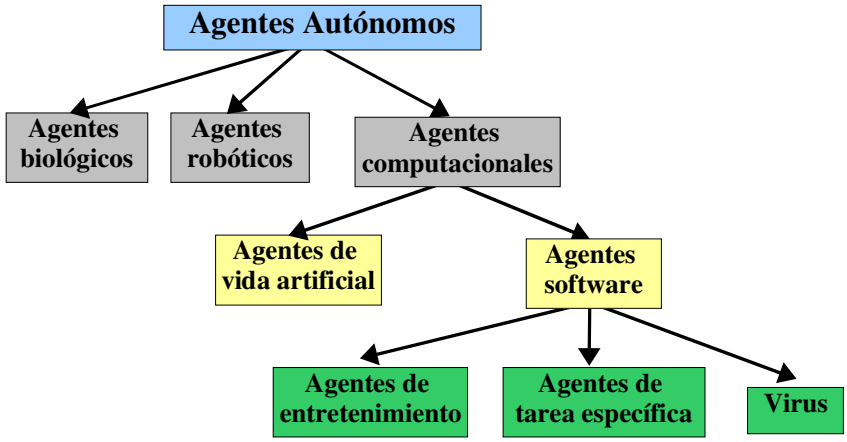
4.1.2 Tipos de agentes

Los dos primeros tipos de agentes que podemos distinguir, independientemente de su estructura interna, son los agentes móviles y los agentes estáticos. Dentro de los agentes móviles, tal y como se especifica en [Etzioni, 95], se incluyen todos los que son capaces de migrar de una plataforma a otra, que en el caso de un programa informático se puede traducir como la capacidad para ser transmitido de un lugar a otro o simplemente

moverse por internet de una máquina a otra. En el caso de un agente hardware, por ejemplo, un robot, esa capacidad de migración se transforma en una capacidad de desplazamiento físico.

Otra clasificación simple de los agentes, los agrupa en reactivos y deliberativos tal y como se establece en el trabajo de Brooks [Brooks, 91]. Los agentes deliberativos se derivan del paradigma de pensamiento deliberativo: los agentes poseen modelos internos del entorno en el que se encuentra y de sus propias motivaciones, que utilizan para planificar las acciones a aplicar. Los agentes reactivos no poseen modelos internos explícitos y actúan mediante un comportamiento de tipo estímulo/respuesta que corresponde al estado actual en el que se encuentran.

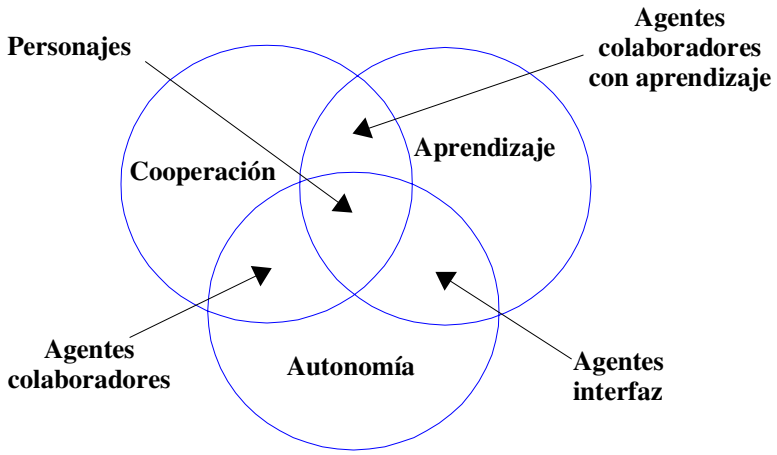
A la hora de establecer clasificaciones más detalladas de los distintos tipos de agentes, hemos encontrado dos maneras diferentes de enfocar dicho proceso. Por una parte, ciertos autores realizan taxonomías generales que organizan los agentes en grandes grupos mientras que otros los clasifican en función del dominio de aplicación. Entre los primeros, cabe destacar la realizada por Stan Franklin en [Franklin, 95] y [Franklin, 96]. En ella se dividen los agentes, en primer lugar, en tres grandes grupos: *biológicos, robóticos* y *computacionales*, que representan respectivamente lo que Keil denomina [Keil, 89] tipos o entidades naturales, es decir organismos animados, artefactos y conceptos abstractos. A su vez, los agentes *computacionales* se pueden dividir en agentes con *vida artificial* y agentes *software*. Por último, los agentes software se pueden clasificar en agentes de *entretenimiento*, agentes de *tarea específica* y *virus*. Esquemáticamente, esta clasificación resultaría:



Vemos cómo Franklin realiza una distinción entre los agentes software y los agentes de vida artificial que otros autores engloban en una misma categoría, concediendo así mayor generalidad a los agentes software. Pero dado el esquema anterior de Franklin, dichos agentes software se reducen a simples programas informáticos.

De los autores que realizan una taxonomía basada en la aplicación destacamos a Hyacinth Nwana, que agrupa los agentes inicialmente en cuatro categorías [Nwana, 96] basándose para ello en tres atributos primarios que todo agente debería exhibir: *autonomía, aprendizaje y cooperación*. Las cuatro categorías resultantes provienen del grado de presencia de cada uno de estos atributos básicos y son: *agentes colaboradores, agentes colaboradores con aprendizaje, agentes de interfaz y agentes de tipo personaje*

(seres virtuales). De forma esquemática, Nwana representa en un espacio bidimensional los atributos primarios encerrados en círculos, y los cuatro tipo de agentes básicos fuera de dichos círculos con flechas indicando la presencia de cada atributo en cada tipo de agente y el grado de solapamiento de los mismos:



Además de estos cuatro tipos, Nwana incluye la distinción entre agentes *móviles* y *estáticos* y entre *reactivos* y *deliberativos* que hemos comentado anteriormente. También incluye una nueva categoría para los agentes distinguiéndolos según su aplicación, por ejemplo, los agentes de *información en internet* que realizan búsquedas masivas de información. Por tanto, la clasificación final más general que aporta contiene siete tipos de agentes básicos:

- **Agentes colaboradores:** cuya función básica es las de negociar para resolver conflictos y colaborar con otros agentes para integrar información de forma distribuida. Un ejemplo de este tipo de agentes es el agente RETSINA ([Sycara, 96]).
- **Agentes de interfaz:** que cooperan con el usuario para llevar a cabo una cierta tarea aconsejándole y aprendiendo de él. Los asistentes de ayuda que aparecen en muchos programas informáticos son un ejemplo simple (ya que carecen de aprendizaje) de agente de interfaz. En el nivel más alto de interacción con el usuario se sitúan los seres virtuales, como el descrito por Foner [Foner, 97].
- **Agentes móviles:** programas que pueden migrar de una plataforma a otra para llevar a cabo su tarea de forma eficiente como, por ejemplo, un programa de enrutamiento de telecomunicaciones.
- **Agentes de información:** recopilan y manipulan grandes cantidades de información de diversas fuentes como, por ejemplo, los softbots [Etzioni, 94] desarrollados en la Universidad de Washington.
- **Agentes reactivos:** están basados en la arquitectura subsumida de Brooks [Brooks, 86] y no utilizan modelos simbólicos internos, de modo que actúan por estímulo-respuesta.
- **Agentes híbridos:** agentes creados como combinación de dos o más de los anteriores, que puede requerir el uso de un lenguaje común para agentes (ACL) a la hora de comunicarse.

Se pueden encontrar en la bibliografía otras taxonomías para agentes basadas en su aplicación, como las planteadas por Brenner en [Brenner, 98] o por Weiss en [Weiss, 99]. Ésta última se basa en la realizada por Nwana que acabamos de presentar, pero parte de la observación de que la investigación en este campo se ha venido realizando con tres tendencias diferentes:

1. **En los años 80:** Inteligencia Artificial Distribuida (DAI) – Sistemas multiagente (MAS).
2. **En los años 90:** con una noción de agente más extensa han aparecido los agentes interfaz, reactivos, móviles, de información, etc.
3. **En los últimos años:** modelado basado en agentes (vida artificial) y otro tipo de agentes comerciales (agentes económicos).

De esta forma, Weiss utiliza tres categorías de agentes básicos extraídas del análisis del trabajo realizado por la comunidad científica:

- **Los agentes colaboradores**
- **Los agentes interfaz**
- **Los agentes económicos**

Además de estos tres tipos de agentes fundamentales, utiliza también los agentes *móviles*, los agentes *reactivos*, los agentes *de información* y los agentes *híbridos* que habían sido utilizados por Nwana. Lo más novedoso de esta nueva taxonomía es el concepto de agente *económico*, que ha aparecido en los últimos años motivado por las aplicaciones de los agentes a problemas de comercio electrónico. Los agentes económicos poseen una serie de características comunes:.

- Se basan en la evolución de normas sociales.
- La autorregulación.
- La adaptación rápida a los cambios.
- Necesidad de alta eficiencia en los resultados.

Como conclusión a este apartado, diremos que las distintas clasificaciones de agentes que se han realizado en la bibliografía del campo, inciden en el hecho de que los agentes son diferentes en función de la aplicación para la que se han diseñado. Así, un agente genérico podrá poseer todas las propiedades que hemos introducido al final del apartado anterior [Bradshaw, 97] pero, en función de su aplicación, los agentes reales no necesitan parte de dichas propiedades, de modo que es la posesión o carencia de éstas la base para la creación de taxonomías.

En el siguiente apartado, haremos una revisión de las principales aplicaciones de los agentes en la actualidad.

4.1.3 Aplicaciones de los agentes

Como se deduce del apartado anterior, cada vez es mayor el número de aplicaciones basadas en agentes, sobre todo en el ámbito de internet. Además, de las clasificaciones dadas por Nwana o por Weiss podemos observar las principales aplicaciones realizadas con agentes en la actualidad.

El concepto más importante tras la aplicación práctica de un agente a un problema es la capacidad de *distribución* de la tarea. Los agentes poseen unas características en cuanto a independencia, capacidad de colaboración y tolerancia a fallos que los hace ideales para ser aplicados a problemas no centralizados tales como el control del tráfico aéreo, el comercio electrónico, etc. Pero manteniendo esta propiedad básica en mente, podemos distinguir 5 grandes grupos de aplicaciones con agentes:

1. **Problemas cooperativos e Inteligencia Artificial Distribuida (DAI):** las aplicaciones en este ámbito se basan en lograr que un grupo de agentes colaboren para llevar a cabo una tarea común de forma coordinada. Por ejemplo, en la administración de sistemas de energía [Varga, 94], en el control de tráfico aéreo [Steeb, 88], administración de redes de telecomunicaciones [Weihmayer, 98], administración de transportes [Fischer, 93], sistemas de fabricación [Parunak, 00], etc. Como vemos, todas las aplicaciones se caracterizan por el alto grado de distribución de los elementos básicos de las mismas. Existen varios textos sobre la Inteligencia Artificial Distribuida (DAI) que realizan un resumen de los principales logros y trabajos llevados a cabo. Dos de los más relevantes y actuales son [O'Hare, 96] y [Weiss, 99].
2. **Agentes interfaz:** como hemos introducido en el apartado anterior, un agente de interfaz actúa como asistente para el usuario en una aplicación informática. Entre los trabajos más destacados en este campo hemos encontrado el realizado por Liebermman, que describe en [Lieberman, 95] un agente llamado *Letizia* que trabaja como asistente en las búsquedas por internet guiando al usuario a base de aprender de búsquedas pasadas. Recientemente ha publicado un libro [Lieberman, 03] donde se recogen todos sus avances en este campo dentro del MIT. Y es que en el grupo de agentes software del MIT residen los trabajos más relevantes con agentes interfaz de los últimos años como, por ejemplo, el proyecto *Kasbah* que se puede consultar en [Chavez, 96] y que consiste en un sitio web completo que representa una tienda electrónica donde los vendedores son agentes que muestran al cliente los productos que le pueden interesar e incluso negocian con ellos el precio de la venta. Fuera del laboratorio del MIT cabe destacar el calendario inteligente (*CAP*) [Mitchell, 94] desarrollado en el Centro de Aprendizaje Automático de la Universidad Carnegie Mellon, dirigido por el Dr. Mitchell y que es capaz de aprender y evolucionar un conjunto de miles de reglas que modelan la planificación de tareas de los usuarios.
3. **Agentes de información:** son agentes que tienen acceso a una o más fuentes de información y que son capaces de manipular dichas fuentes en respuesta a las consultas realizadas por los usuarios y por otros agentes de información. Etzioni y Weld describen en [Etzioni, 94] un agente de este tipo denominado *internet softbot*, que permite al usuario la realización de una solicitud de alto nivel, y el *softbot* utiliza conocimiento inferencial para determinar como satisfacer dicha solicitud en internet. Para ello, maneja conceptos como ambigüedad, omisión de datos y errores en las solicitudes. Otro sistema importante en este área es el denominado *Carnot* [Huhns, 97], que permite que los sistemas de bases de datos existentes y heterogéneos trabajen de forma conjunta para resolver consultas más allá del dominio de cada base de datos por separado.
4. **Seres virtuales (personajes):** existen agentes que simulan el comportamiento de un ser humano proporcionando al usuario la ilusión de tratar con una persona de verdad. Un ejemplo de este tipo de aplicación es el *Proyecto Persona* de Microsoft [Ball, 97] que está desarrollando asistentes de conversación que interactúan con el usuario

mediante un diálogo natural. Este tipo de seres no están pensados para realizar tareas de venta o asistencia al usuario, sino que son simplemente seres con los que conversar sobre un cierto tema.

5. **Agentes móviles:** en este grupo, los autores incluyen agentes con la capacidad de migrar de una plataforma a otra para, por ejemplo, gestionar una reserva de un vuelo o una red de telecomunicaciones moviéndose por una red WAN o por internet. La aplicación más interesante en este ámbito ha sido desarrollada por Sony y es el sistema *Sony Magic Link* [Sony, 03], que asiste a los usuarios en la administración de e-mail, fax, teléfono, etc así como la conexión del usuario a los servicios Telescript habilitados de mensajería y comunicación tales como America Online y AT&T PersonaLink. Telescript es un lenguaje de programación interpretado, orientado a objetos y al procesamiento remoto que permite el desarrollo de aplicaciones distribuidas. Las aplicaciones de Telescript constan de agentes de Telescript que funcionan dentro de un mundo virtual de lugares, de motores, de nubes y de regiones [Nwana, 96].

Aunque podríamos seguir comentando ejemplos de aplicaciones con agentes, con los mostrados hasta ahora es suficiente para enmarcar el presente trabajo. Subrayar que el número de tales aplicaciones es mayor cada vez y el campo de la utilización práctica de los agentes está en pleno auge.

Tras haber revisado las definiciones, clasificaciones y aplicaciones más destacadas que se han llevado a cabo utilizando agentes, en el siguiente apartado estableceremos nuestra propia definición y clasificación, que nos servirá más adelante para centrarnos en un tipo especial de agentes: los robots móviles.

4.1.4 Definición y taxonomía propia

El objetivo general de todo agente consiste en ejecutar acciones que le lleven a realizar una cierta tarea de forma adecuada. En el apartado siguiente trataremos el problema de decidir quién establece la tarea que se debe llevar a cabo y qué significa resolverla de forma adecuada. El concepto de agente con el que trabajaremos, se resume en la definición introducida en el apartado 2.2:

> *Un Agente Autónomo Inteligente es un sistema computacional independiente dotado de sensores y actuadores que habita en un entorno dinámico y que posee la capacidad de razonar y adaptarse a condiciones internas y externas cambiantes.*

Hemos utilizado una definición general basada en tres **propiedades básicas** que, desde nuestro enfoque, definen a todo agente:

1. **Sistema computacional:** un agente es un sistema artificial basado en reglas computacionales y nuestra labor es de diseñadores o ingenieros para este tipo de sistemas. Esto excluye a los seres vivos de la definición de agente.
2. **Independencia:** un agente debe poseer autonomía funcional, es decir, debe poseer sensores y actuadores propios y algún tipo de mecanismo interno que le permita escoger sus acciones.
3. **Inteligencia:** un agente debe poseer la capacidad de razonar y obtener soluciones originales por sí mismo a los problemas que se le presentan para llevar a cabo la

tarea que rige el comportamiento. Para ello, debe tener la capacidad de aprender de sus situaciones pasadas, esto es, recombinar información parcial y adaptarla a nuevas situaciones. Este es el concepto de inteligencia que manejaremos a lo largo del presente estudio.

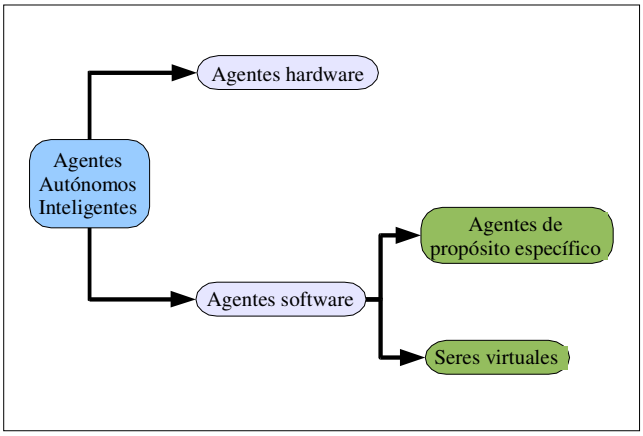
Ademas de estas tres propiedades básicas que todo agente debe exhibir, pueden poseer o no **otras dos propiedades**:

- **Comunicación:** entre los sensores y actuadores, un agente puede tener aquellos que le permitan físicamente realizar la comunicación con otros agentes o con un usuario. Por tanto, esta propiedad es un caso particular de la de independencia que hemos catalogado como básica. La capacidad de comunicarse implica además la existencia de un lenguaje común y técnicas para poder utilizar dicho lenguaje y colaborar, capacidad que no es imprescindible para ser catalogado como agente.
- **Movilidad:** los agentes pueden ser móviles en dos sentidos. Por una parte, un agente móvil desde el punto de vista computacional posee la propiedad de poder migrar de una plataforma a otra, de modo que el agente se puede transmitir de un punto a otro. Por otro lado, desde el punto de vista físico, un agente se puede desplazar. En cualquier caso, ésta no es una propiedad que impida la clasificación como agente.

De la totalidad de propiedades que los distintos autores han atribuido a los agentes, como por ejemplo las mencionadas en el apartado 4.1.1, nos hemos quedado con estas cinco que las engloban todas. Vemos cómo la definición de agente que hemos establecido se basa en las tres propiedades que hemos catalogado como básicas, no existiendo mención explícita a la *comunicación* ni a la *movilidad*. De todas ellas, la *inteligencia,* en la acepción que hemos considerado, es la más importante y es la que estudiaremos en profundidad en el presente trabajo.

Hemos incluido como propiedad básica la *independencia*, que es un concepto similar a la autonomía pero no idéntico. Tal y como establecen Russell y Norvig en [Russell, 96], un sistema será autónomo en la medida en que su conducta está definida por su propia experiencia. Pero esta definición, que nos parece muy acertada, no incluye la autonomía sensorial y actuadora que sí es una propiedad diferenciadora de un agentes. Con objeto de generalizar, por tanto, el concepto de autonomía hemos definido la independencia tal y como se indica anteriormente.

Sentadas estas bases, la taxonomía que planteamos para los agentes es la siguiente:



Es una taxonomía simple y muy general, basada en características diferenciadoras relevantes y no en la aplicación del agente. Así, los AAI se pueden dividir en dos grandes grupos, los agentes hardware y los agentes software. Los primeros se caracterizan por ser artefactos autosuficientes en el sentido de que poseen una computadora propia, sensores y actuadores físicamente unidos al agente que le permiten manipulaciones independientes y que "habitan" el mundo real. Lógicamente, no son autosuficientes energéticamente hablando. Es decir, un agente hardware es un *robot* dotado de un Mecanismo Cognitivo que le proporcione capacidades inteligentes, de tal forma que un robot mecánico automático (un autómata) no puede ser considerado como un AAI. Este concepto de robot inteligente es muy similar al utilizado por Arkin en [Arkin, 98], texto de referencia obligada en este campo. Los robots que catalogamos como Agentes Autónomos Inteligentes pueden ser mecánicos, eléctricos e incluso pueden tener componentes orgánicos o biológicos, pero sin olvidar la restricción que hemos impuesto: un AAI es un ser artificial, nunca un ser vivo. Los agentes hardware pueden tener la capacidad de comunicarse con otros agentes o no, del mismo modo que se pueden desplazar de un punto a otro o permanecer estáticos sin por ello dejar de ser considerados como Agentes Autónomos Inteligentes.

Los agentes software por su parte "habitan" en un entorno virtual, es decir, dentro de una computadora (en general, un sistema computacional) y no tienen sentido fuera de ella. Dentro de los agentes software, sólo hemos distinguido dos categorías: los agentes de propósito específico y los seres virtuales. En el primer grupo se engloban prácticamente todos los agentes que existen en la bibliografía del campo y que hemos introducido en el apartado 4.1.2: agentes de información, agentes interfaz, agentes reactivos, agentes económicos, etc. En esta categoría se incluye desde un agente guía en un programa informático hasta un sistema de agentes que controlen los elementos domóticos de una vivienda. Todos ellos son agentes con una tarea específica claramente diferenciables de los agentes hardware por no tener presencia física.

El único agente software que no tiene una tarea concreta es el denominado ser virtual o personaje (lifelike character). Este tipo de agente únicamente habita en un mundo virtual (internet, una computadora, un PDA, etc) aprendiendo de su usuario sin ninguna intención más allá del puro entretenimiento. Este tipo de agentes son utilizados en la mayor parte de los casos como experimento sociológico o psicológico.

Como se observa, nuestra definición y clasificación de agente está próxima a la dada por Franklin en [Franklin, 96] y que se muestran en los apartados 4.1.1 y 4.1.2 anteriores, aunque no incluimos los agentes biológicos ni realizamos una distinción tan clara entre agentes software según su aplicación. Pero por otro lado sí realizamos la separación clara entre agente hardware y software e incluimos el agente con vida artificial en nuestra taxonomía.

Con esto damos por finalizado este apartado de introducción a los Agentes Autónomos Inteligentes. En el apartado siguiente, pasamos a revisar las distintas aproximaciones que se han utilizado para tratar de dotar a dichos agentes de inteligencia y autonomía, es decir, los distintos mecanismos y modelos cognitivos que se han desarrollado. Además, realizaremos un análisis general de todos los elementos que han de ser tenidos en cuenta a la hora de abordar una tarea tan compleja.

4.2 Modelos cognitivos en agentes

De las tres propiedades básicas que debe poseer todo agente, el que sea un sistema computacional artificial y la independencia sensorial y actuadora son requerimientos simples de diseño hoy en día. Digamos que cualquier agente posee estas dos propiedades. En cambio la tercera, la inteligencia, es mucho más compleja de obtener y debe aportar al agente autonomía funcional (ya que la autonomía sensorial y actuadora por separado son poco útiles) y capacidad de razonamiento y aprendizaje. La organización estructural de los agentes que hemos desarrollado en base a una serie de propiedades, se traduce en una distinción clara entre la arquitectura de un agente y un mecanismo o programa que explota dicha arquitectura. Es, por tanto, en este último aspecto en el que se han centrado los mayores esfuerzos de los distintos investigadores, de tal forma que las diferentes aplicaciones realizadas con agentes, algunas de las cuales ha sido presentadas en el apartado 4.1.3, se fundamentan en el mecanismo utilizado para dotar a los agentes de inteligencia.

Previo al estudio de dichos mecanismos, realizaremos una revisión sobre las teorías de agentes, que analizan la construcción de formalismos y las propiedades de los agentes expresados en tales formalismos.

4.2.1 Teorías de agentes

Existen muchos trabajos diferentes centrados en el estudio y el planteamiento teórico de modelos de actuación para los agentes previos al desarrollo del mecanismo interno. Por ejemplo, Moore fue uno de los primeros investigadores en tratar de responder la pregunta de qué precisa saber un agente de antemano para poder escoger la acción a aplicar de forma adecuada [Moore, 85]. Formalizó un modelo de *habilidad* a la hora de buscar la mejor acción utilizando para ello una lógica basada en conocimiento. Este formalismo permite que un agente que no posee información completa sobre cómo alcanzar un objetivo realice acciones que le permitan llegar a alcanzarlo. Otro trabajo muy influyente en la teoría sobre agentes es el realizado por Cohen y Levesque, que desarrollaron en [Cohen, 90] una teoría basada en *intenciones*, que ha sido muy utilizada y criticada desde entonces. A la hora de caracterizar el comportamiento básico de un agente racional, establecen la existencia de únicamente dos actitudes, una basada en *creencias* (que representan el estado del mundo en el que se encuentra o cree encontrarse el agente) y otra en *metas* (estado donde le gustaría estar). Las *intenciones*, que rigen el comportamiento, se derivan de éstas dos. Basándose a su vez en el trabajo de Bratman [Bratman, 90], identifican siete propiedades que se deben satisfacer si se pretende resolver un problema:

1. Las *intenciones* resultan un problema para los agentes, que deben encontrar formas de alcanzarlas.
2. Las *intenciones* proporcionan un filtro para otras *intenciones*, de tal forma que no debe existir conflicto.
3. Los agentes buscan satisfacer sus *intenciones* y tienden a realizar varios intentos hasta lograrlo.
4. Los agentes creen que sus *intenciones* son posibles (*creencias*).
5. Los agentes no creen que ellos produzcan sus *intenciones*
6. Bajo ciertas circunstancias, los agentes creen que ellos producen sus *intenciones*.

7. Los agentes no tienen en cuenta todos los efectos colaterales de sus *intenciones*.

Dados estos criterios, Cohen y Levesque construyen una lógica racional para los agentes basada en un conjunto de bloques básicos, uno de los cuales es la *intención*.

En una linea similar destacamos el trabajo de Rao y Georgeff, que establecen la existencias de tres actitudes básicas en el comportamiento: *creencias*, *objetivos* e *intenciones*, elevando así las *intenciones* a la categoría base [Rao, 93]. Es particularmente interesante su noción de realismo basada en el hecho de cómo las *creencias* de un agente sobre el futuro pueden afectar a sus *deseos* e *intenciones*.

A la hora de formalizar el comportamiento de los agentes desarrollando para ello una lógica que modele los elementos básicos de la actuación, destacan también los trabajos de Sighn [Singh, 94], Woolridge [Woolridge, 92] y Allen [Allen, 91].

Un texto que merece especial mención es el realizado por Russell y Norvig [Russell, 96] que analiza todos los aspectos relativos a la inteligencia aplicada a agentes sentando para ello las bases teóricas del conocimiento y el aprendizaje, tarea altamente compleja. Esta es una referencia más general que las anteriores que estaban más centradas en la lógica que rige los comportamientos, pero ofrece una perspectiva más moderna del concepto de inteligencia y formalización de la misma, más cercana a la psicología y a las teorías evolutivas que a la lógica matemática. Russell y Norvig establecen que a la hora de actuar, un agente se debe basar en un comportamiento racional, y el carácter de racionalidad de lo que hace en un momento dado depende de cuatro factores:

1. De la medida con la que se evalúa el éxito.
2. De las percepciones pasadas del agente.
3. Del conocimiento que posee del medio.
4. De las acciones que el agente puede realizar.

En base a estos factores definen el agente racional ideal del modo siguiente:

> *En todos los casos de posibles secuencias de percepciones, un agente racional deberá emprender todas aquellas acciones que favorezcan obtener el máximo de su medida de rendimiento, basándose en las evidencias aportadas por la secuencia de percepciones y en todo conocimiento incorporado al agente.*

A continuación pasamos a revisar las distintas aproximaciones empleadas para pasar de estas teorías a la implementación práctica de mecanismos de razonamiento e inteligencia para agentes, lo que se denomina en la bibliografía del campo como arquitectura interna de agentes.

4.2.2 Arquitectura interna de agentes

En [Russell, 96] se establece que los agentes se pueden clasificar en cuatro tipos diferentes en función del mecanismo de selección de acciones que utilizan:

1. **Agentes de reflejo simple:** donde las acciones son seleccionadas mediante unas reglas de condición-acción, que relacionan percepciones y acciones, y que pueden ser impuestas o aprendidas.

2. **Agentes bien informados de todo lo que pasa:** similares a los anteriores con la adición de un estado interno que mantiene actualizada una representación del ambiente que se utiliza para seleccionar la mejor acción a aplicar.
3. **Agentes basados en metas:** donde la selección de la acción a ejecutar ya no se basa en reglas, sino que se escoge la acción que más satisfaga las metas u objetivos del comportamiento.
4. **Agentes basados en utilidad:** este tipo de mecanismos, además de las metas, utilizan un criterio de utilidad para escoger la acción a aplicar, de tal forma que el grado de satisfacción del agente se podría basar en un criterio subjetivo.

Pero la clasificación clásicamente más utilizada distingue únicamente tres tipos de arquitecturas diferentes a la hora de dotar de capacidades inteligentes y de razonamiento a un agente: arquitecturas deliberativas, arquitecturas reactivas y arquitecturas híbridas.

Arquitecturas deliberativas

Esta es una aproximación clásica, que se basa en construir agentes como si fuesen sistemas basados en conocimiento. El paradigma de la Inteligencia Artificial Simbólica se basa en la *hipótesis de sistemas de símbolos físicos* formulada por Newell y Simon [Newell, 76] y establece que una serie de símbolos pueden codificar secuencias de instrucciones que, combinadas de forma adecuada, dan lugar a acciones inteligentes. Así, se define una arquitectura deliberativa como aquella que contiene un modelo simbólico del mundo representado de forma explícita y donde las acciones a realizar se seleccionan por razonamiento lógico basado en patrones y manipulación simbólica.

En la Inteligencia Artificial tradicional ha dominado este paradigma de manipulación de símbolos. En el caso particular de la selección de acciones, esto implica que el agente tiene una representación interna de las acciones, los objetivos y las situaciones que se pueden presentar y que esta representación se utiliza en un proceso de planificación para determinar qué secuencia de acciones alcanzará los objetivos fijados. Dicho plan lo utiliza el proceso de ejecución que lleva a cabo la secuencia de forma más o menos ciega. Aunque este planteamiento ha tenido éxito en algunas tareas, en el caso de la planificación los resultados han sido muy pobres, especialmente cuando se aplica a agentes autónomos actuando en entornos complejos y dinámicos. En estas situaciones los sistemas probados presentan grandes deficiencias tales como fragilidad, falta de flexibilidad y tiempos de respuesta muy largos.

Entre los primeros trabajos dedicados a aplicar Inteligencia simbólica a los agentes, debemos mencionar los agentes basados en planes, es decir, basados en un programa que representa una secuencia de acciones que al ser ejecutadas conducen al logro de un cierto objetivo. Los planes requieren un modelado del mundo y este modelo es difícil de obtener, entre otras cosas porque el mundo sólo se puede conocer en parte, y es muy difícil de mantener y actualizar dado que cambia de forma continua. Desde el punto de vista computacional, nos encontramos con el hecho de que las aproximaciones basadas en planes tienen problemas de explosión combinatoria, lo que impide a los agentes actuar en tiempo real. Existen diferentes estudios [Chapman, 87] que indican que la complejidad de los sistemas que serían necesarios para abordar problemas de alto nivel de razonamiento, convierte a los sistemas basados en planes en soluciones adecuadas únicamente en casos simples. Por tanto, este tipo de aproximaciones basadas en planes han sido desechadas, en general, por los investigadores, aunque podemos mencionar los

sofbots de Etzioni, que son agentes software que utilizan un planificador para actuar en un entorno Unix [Etzioni, 94a] como una de las aplicaciones más recientes.

La arquitectura *IRMA* [Bratman, 88] fue una de las pioneras al utilizar los conceptos de *creencias*, *deseos* e intenciones que mencionamos en el apartado anterior en las teorías de agente. En *IRMA* existen cuatro estructuras de datos simbólicas básicas: una librería de planes y una representación explícita de *creencias*, *deseos* e intenciones. Además, se utiliza una estructura de razonamiento, un analizador de planes, un analizador de oportunidades, un proceso de filtrado y un proceso deliberativo.

Otra aproximación basada en símbolos que podemos destacar, es la realizada por Vere en el desarrollo del agente *HOMER* [Vere, 90]. Dicho agente es un robot submarino simulado que habita en un mundo marino bidimensional del que tiene un conocimiento parcial, y que toma instrucciones del usuario en un lenguaje simple. *HOMER* puede planificar cómo realizar las instrucciones que le dan y ejecutar dichos planes, con la capacidad de modificarlos durante su ejecución. Además posee una memoria, aunque muy limitada, que le permite responder preguntas sobre sus experiencias pasadas.

Como resumen a este apartado, diremos que los investigadores se han encontrado, en general, con demasiados problemas en la aproximación deliberativa simbólica que les ha llevado a un cambio del enfoque del problema que ha dado lugar a las arquitecturas reactivas, que pasamos a comentar.

Arquitecturas reactivas

El énfasis de estas implementaciones se centra en el enlace directo entre la percepción y la acción, en la descentralización, en las interacciones dinámicas con el entorno y en la implicación de mecanismos intrínsecos para manejar la falta de recursos y el conocimiento incompleto que normalmente se tiene del entorno o problema. Por otra parte, todas estas arquitecturas comparten la idea de funcionalidad emergente, es decir, la funcionalidad del agente es una propiedad resultante de la interacción intensiva entre el sistema y el entorno dinámico. La simple especificación del comportamiento no explica la funcionalidad mostrada por el agente actuando en entornos reales. Esto es, el entorno no se toma como un lugar donde se actúa, sino como una parte integrante del sistema, un elemento o recurso a ser explotado por el mismo.

De entre todos los trabajos existentes en este campo, destaca el presentado por Rodney Brooks en [Brooks, 86] donde desarrolla la *arquitectura subsumida*. Ésta consiste en una jerarquía de comportamientos cada uno asociado a una tarea diferente donde cada comportamiento compite con los demás para hacerse con el control del agente. Volveremos a mencionar este tipo de aproximación en un apartado posterior por pertenecer originalmente al campo de la robótica autónoma al que dedicaremos especial atención.

En esta linea de investigación ha estado trabajando la Dra. Patti Maes, que ha desarrollado una arquitectura para agentes en la que un agente se define como un conjunto de módulos de competencia [Maes, 91]. Cada uno de ellos es especificado por el diseñador en términos de pre y postcondiciones y un nivel de activación de dicho módulo, que cuanto mayor sea más influirá en el comportamiento del agente.

Además de los trabajos basados en el de Brooks, destacamos el de Agre y Chapman, que desarrollaron una arquitectura para un agente [Agre, 87] partiendo de la idea de que

las decisiones que toma dicho agente se vuelven rutinarias. Por este motivo, dichas decisiones se codifican en una estructura de bajo nivel, por ejemplo un circuito, que sólo necesita ser actualizado de forma periódica.

Una aproximación más sofisticada fue realizada por Rosenschein y Kaelbling y bautizada como *autómata situado* [Kaelbling, 91]. En este caso, los agentes se especifican en términos declarativos y son aplicados en una máquina digital que no necesita realizar ninguna manipulación de símbolos. Dado que, de nuevo, el desarrollo de este tipo de arquitectura tuvo lugar en el campo de la robótica autónoma, volveremos sobre él más adelante.

Todas estas arquitecturas cognitivas suponen implícitamente que, replicando de forma organizada un elemento que se considera paradigma de procesado, se puede llegar a una computación de alto nivel. No contienen elementos explícitos de tipo deliberativo o reflexivo y han demostrado ser capaces de realizar múltiples tareas y de adaptarse a entornos cambiantes, pero se puede decir que presentan carencias en dos aspectos fundamentales: una limitación inherente a la falta de elementos reflexivos y, dado que la mayoría han de diseñarse a mano, un problema de explosión combinatoria de complejidad cuando se aplican a entornos reales y tareas complejas.

Cuando el campo de las arquitecturas reactivas surgió como contrapartida a las deliberativas que tantos problemas estaban dando a los investigadores, se les llamó aproximaciones modernas en oposición a las simbólicas que eran las aproximaciones clásicas. Pero hoy en día las tendencias han variado de nuevo, y se ha optado cada vez más por aproximaciones intermedias o híbridas que conservan lo mejor de cada campo.

Arquitecturas híbridas

Una de las primeras arquitecturas de este tipo fue la desarrollada en 1987 por Georgeff y Lansky y denominada *Sistema de Razonamiento Procedimental* (PRS) [Georgeff, 87] que es una arquitectura similar a *IRMA* ya que se basa en *creencias*, *deseos* e intenciones que son representadas de forma explícita. Las *creencias* se representan mediante una lógica clásica de primer orden y los *deseos* se representan como sistemas de comportamiento. Además, el PRS incluye una librería que contiene un conjunto de planes parcialmente elaborados llamados *áreas de conocimiento* que pueden ser reactivos, permitiendo así respuestas rápidas a los cambios en el entorno. El conjunto de *áreas de conocimiento* activas en un sistema representa sus intenciones.

Otra arquitectura a destacar es la propuesta por Ferguson en su tesis doctoral [Ferguson, 92], que consiste en dos subsistemas de percepción y acción que interfieren directamente con el entorno del agente y tres capas de control (reactiva, planificadora y modeladora). Cada una de estas capas produce y ejecuta de forma concurrente diferentes procesos.

INTERRAP es una arquitectura basada en capas similar a la anterior donde cada capa sucesiva representa un nivel de abstracción mayor [Müller, 94]. Cada una de estas capas se subdividen en dos verticales, una que contiene bases de conocimiento y otra que contiene componentes de control. En el nivel más bajo de estos componentes de control se sitúa la base de conocimiento de modelos de mundo, que controla la relación entre el agente y su entorno.

El mecanismo que presentamos en esta tesis, se enmarca dentro de este tipo de arquitecturas híbridas aunque no utilizamos ningún tipo de planificación basada en

símbolos ni es un sistema puramente reactivo. Como iremos viendo a lo largo del trabajo, el MDB es un mecanismo deliberativo en el sentido de que utiliza modelos internos que son aplicados a la hora de seleccionar las acciones a ejecutar y posee capacidades reactivas ya que puede aplicar directamente acciones que en el pasado resultaron adecuadas.

Hasta este punto ha llegado la revisión que realizamos de los trabajos más destacados en el campo de los agentes autónomos durante los últimos años. El gran auge de las investigaciones en este campo se produjo a principios y mediados de los años 90 como se puede observar de la mayor parte de las referencias presentadas. En la actualidad, lo más destacado son las aplicaciones que se llevan a cabo con agentes y que auguran un gran futuro a este campo.

A continuación, realizaremos un análisis de los Mecanismos Cognitivos que se han desarrollado aplicados a un tipo concreto de agente de especial interés en nuestra investigación, los agentes hardware o robots autónomos.

4.3 Agentes hardware

A la hora de implementar en la práctica las distintas arquitecturas de agente que acabamos de presentar, los autores tienden a aplicarlas en agentes software o bien en agentes hardware simulados dada la gran complejidad de los agentes hardware reales. Tal y como se discute en [Russell, 96], el paso de este tipo de agentes hardware simulados (por tanto, agentes software) al entorno real implica gran cantidad de problemas y fallos derivados de las grandes diferencias existentes entre los mundos virtuales, perfectamente controlables y manipulables, y el mundo real donde los agentes hardware o robots son aplicados. Russell establece que cualquier entorno real posee cinco características que lo hace especialmente complejo:

1. **Es inaccesible:** los sensores sólo perciben los estímulos cercanos al agente.
2. **Es no determinista:** desde el punto de vista del robot, no existe total certeza de que una acción bien seleccionada vaya a funcionar (problemas de hardware).
3. **Es no episódico:** los efectos que producen las acciones cambian con el tiempo. Por esta razón los robots deben no sólo aprender de su experiencia sino que deben ser capaces de reaccionar en tiempo real.
4. **Es dinámico:** y, en general, no modelable en su totalidad.
5. **Es continuo:** no es posible enumerar el total de posibles acciones y consecuencias.

Además de la complejidad de los entornos, los agentes hardware se caracterizan porque todas sus partes físicas son susceptibles de fallar y, por tanto, la propia estructura del agente puede cambiar en tiempo de ejecución. Es decir, tanto la sensorización como la actuación son controlables y predecibles en los agentes software o en agentes hardware simulados, pero no lo son en agentes hardware.

En resumen, de los distintos tipos de agentes que podríamos considerar para la aplicación del mecanismo que se presenta, los agentes hardware o robots autónomos son los más adecuados, fundamentalmente, por las siguientes razones:

1. Son sistemas que cumplen todas las propiedades asociadas a los agentes que hemos impuesto en el apartado 4.1.4 ya que, además de las tres básicas que son necesarias simplemente para ser catalogado como agente, poseen movilidad física y capacidad de comunicación.
2. Son plataformas de aplicación práctica complejas tanto por sus deficiencias sensoriales como actuadoras. Esto nos permite probar el MDB en las condiciones más adversas posibles dado que la robustez del mecanismo es fundamental.
3. Son agentes que pueden interaccionar con el usuario en su mismo mundo, lo que nos da la opción de realizar experimentos de alto nivel de razonamiento donde el entorno no es fácilmente manipulable y se vuelve dinámico y complejo.

De acuerdo con esta elección, a continuación realizaremos una revisión del campo de la robótica autónoma centrándonos en los Mecanismos Cognitivos que se han desarrollado a lo largo de estos últimos años.

4.3.1 Robótica autónoma

Una definición ajustada del concepto de robot inteligente es la aportada por Arkin en [Arkin, 98]:

> *Un robot inteligente es una máquina capaz de extraer información de su entorno y utilizar conocimiento sobre su mundo para moverse de forma segura, justificada e intencionada.*

Como ya hemos comentado en el apartado 4.1.4, un robot autónomo se caracteriza porque posee actuadores físicos independientes que le permiten realizar acciones en el mundo real. Las partes fundamentales de todo robot son tres:

1. **El cuerpo o estructura base:** donde se encuentran los componentes computacionales hardware (procesadores, microcontroladores, placas base, etc) sobre los que se implementa la arquitectura de agente. Una revisión muy completa sobre la construcción de los robots y los distintos tipos que se han construido se puede encontrar en [McKerrow, 91]
2. **Los sensores:** que proporcionan al robot los estímulos de su entorno. Constituyen la forma en que el robot percibe el mundo, de tal modo que cuanto mayor número de sensores posea y más precisos y robustos sean dichos sensores, más fácil será la tarea de llevar a cabo un determinado comportamiento y, desde el punto de vista del diseñador, más complejos serán los problemas que se pueden abordar. Existe una gran variedad de sensores en la robótica actual: sensores de luz, de sonido, de infrarrojos, táctiles, etc [Everett, 95], [Ghosh, 99]. La mayor parte de estos sensores son extremadamente simples y la información que proporcionan es pobre, además de incluir señales de ruido solapadas con la señal percibida. Hay una gran cantidad de autores trabajando para dotar a los robots de visión artificial con cámaras [Ayache, 89], [Nayar, 96] tratando de realizar segmentación de imágenes en tiempo real. Este es un campo muy extenso y donde los avances se encuentran a la orden del día dada la gran complejidad de la tarea de poder manipular tales cantidades de información. Los autores tienden a crear sistemas que se basan en la división del espacio en zonas de mayor o menor interés para la creación de mapas del entorno [Davison, 02] o en mecanismos de atención, como el trabajo que estamos desarrollando en el Grupo de Sistemas Autónomos [Crespo, 00]. En el momento en que la robótica pueda utilizar

toda la información que contiene una imagen de forma computacionalmente eficiente, la relevancia de las tareas a realizar aumentará espectacularmente.

3. **Los actuadores:** que permiten que el robot interactúe con su entorno. Al igual que en el caso de los sensores, cuanto mejores sean dichos actuadores (más fiables, precisos y útiles), mayor complejidad en las tareas que puede llevar a cabo. De los distintos actuadores que se han implementado en robots, entre los más comunes se encuentran los siguientes: ruedas, piernas y brazos articulados, pinzas, altavoces, etc. La posesión de actuadores que manipulan el mundo real es la principal característica de los robots, y es lo que les confiere el alto grado de independencia que nos interesa explotar.

Existen en la bibliografía de este campo distintos textos introductorios a los robots autónomos como por ejemplo [Kortenkamp, 98] y, sobre todo, [Arkin, 98]. En relación con el trabajo que queremos presentar, nos centraremos en los diferentes intentos que se han llevado a cabo para dotar de un mecanismo interno que permita una utilización eficiente de las capacidades que posee el robot con el objetivo de realizar tareas inteligentes en un entorno real.

4.3.2 Mecanismos Cognitivos en robótica

En el estudio e implementación de la inteligencia, clásicamente, dentro de la comunidad de la Inteligencia Artificial, se ha tendido a dividir a un ser inteligente en componentes individuales y a tratar de implementar cada parte por separado. Es decir, se ha dado mucho énfasis al razonamiento en dominios concretos altamente especializados, y esto ha llevado a una definición de inteligencia basada en la capacidad de resolver problemas concretos en dominios específicos.

Recientemente, se han introducido nuevas aproximaciones al problema consistentes en considerar los seres inteligentes como un todo en interacción constante con un entorno y con capacidad de adaptarse a circunstancias cambiantes e impredecibles de dicho entorno. Esto conlleva una premisa inherente a la definición de inteligencia como capacidad de adaptación al medio y generación de los recursos cognitivos necesarios para sobrevivir. O sea, la inteligencia se contempla como la habilidad para enfrentarse a un mundo dinámico y poco predecible, y sobrevivir.

Dentro de esta última tendencia, se proponen dos aproximaciones diferentes, por una parte evolucionar seres hasta obtener inteligencia y por otra diseñar sistemas nerviosos artificiales que controlan los comportamientos de dichos seres. En el primer caso, se ha contemplado el hecho de la emergencia de comportamientos inteligentes como resultado de evoluciones de comunidades de seres simples enfrentados a entornos de distintos grados de complejidad y que han de sobrevivir y adaptarse a ellos. Esta aproximación es la más cercana a la forma en que se supone que la inteligencia ha surgido en nuestro mundo, esto es, la naturaleza nunca se preocupó de realizar ningún tipo de diseño, simplemente generó muchos ejemplos posibles de seres y las circunstancias fueron seleccionando los que mejor se adaptaban a situaciones cambiantes, consiguiendo de este modo reunir en ciertos organismos las características necesarias y los recursos precisos para sobrevivir. Evidentemente, este camino no predetermina cómo han de ser los mecanismos ni la base física de la inteligencia, pero el proceso podría llevar unos cuantos millones de años además de ser muy poco práctico en el caso de robots reales.

Desde el segundo punto de vista, similar a los métodos tradicionales de implementación de funciones, se busca un procedimiento para el diseño de un sistema nervioso artificial que controle el comportamiento. Esta aproximación, aunque mucho más cercana a la forma en que tradicionalmente los humanos hemos hecho las cosas, presenta el problema de que predeterminamos cómo ha de funcionar el ser, llegando al extremo de no saber muy bien si el comportamiento resultante está condicionado por el diseño, especialmente en los aspectos de utilización de experiencia ante circunstancias nuevas y en la generación de soluciones originales. En este ámbito destaca el trabajo de Randall Beer que desarrolla un sistema nervioso completo para un insecto artificial [Beer, 97], [Chiel, 97]. Aunque en trabajos más recientes consigue desarrollar comportamientos robustos ante cambios de entorno, siguen siendo situaciones creadas de antemano y las posibles diferencias entre entornos están muy limitadas.

Una de las principales preocupaciones de la robótica y la Inteligencia Artificial actual es el desarrollo de mecanismos para la selección de acciones que puedan ser utilizados por agentes autónomos con el objeto de determinar lo que deben hacer en el siguiente instante de tiempo. Cualquier modelo cognitivo tiene como función principal la selección de acciones en función de las condiciones internas y externas de un cierto sistema inteligente. Debemos distinguir en el enfoque de este problema entre las aproximaciones más clásicas (tanto reactivas como deliberativas) donde la participación del diseñador es constante y las nuevas tendencias mezcla de distintos paradigmas que buscan el mayor grado de autonomía posible.

Aproximaciones clásicas

Dentro de este tipo de sistemas se engloban los trabajos basados en arquitecturas reactivas como los autómatas situados [Kaelblin, 87], arquitecturas subsumidas [Brooks, 84] o [Rosenschein, 87], arquitecturas basadas en comportamientos [Brooks, 86] planes universales [Brooks, 91], redes de acciones [Nilsson, 89], etc.

A menudo, estas arquitecturas son jerarquías de decisión que se inspiran en el mundo animal [Booker, 90], [Cherian, 92], [Digney, 94], [Beer, 97] que es un punto de referencia común en todo el campo de la robótica autónoma. Constan de varios centros, cada uno dedicado a una función específica, que se organizan en jerarquías de elementos de decisión. Cada centro en cada nivel puede tomar entradas de muchos estímulos sensoriales, tanto internos como externos, y puede activar centros en niveles inferiores. Otras propuestas implementan un elemento de arbitraje cuya tarea es obtener una acción única como la unión de las propuestas realizadas por muchos módulos dedicados [Tenenberg, 92].

Los sistemas motivacionales de otras arquitecturas no son jerárquicos y difieren en el grado de arbitraje al que se someten sus módulos de comportamiento. Por ejemplo, en las arquitecturas controladas por campos de fuerzas, no hay ningún arbitraje [Arbib, 92], [Arkin, 94] y cada módulo contribuye a un vector relacionado con el objetivo actual del robot. En las arquitecturas subsumidas [Brooks, 84], todos los módulos están activos pero sus salidas pueden ser inhibidas por otros módulos [Mataric, 92], [Donnart, 94].

Todos estos sistemas basados en metodologías estáticas, definidas por un diseñador, tienden a ser escasamente adaptativos dado que la representación del mundo y el conocimiento que poseen han de ser definidos a priori, el diseñador debe transferir su conocimiento al sistema. Esto, además de no ser práctico salvo en dominios muy limitados, adolece del problema de que el mundo cambia constantemente y sólo se

puede predecir de forma parcial. Lo que es más importante, si utilizamos objetivos y planes en nuestro agente en el sentido tradicional o sea, expresiones simbólicas, tenemos que establecer la conexión entre estos símbolos y el mundo exterior (symbol grounding). Este problema ha sido una cuestión extremadamente difícil en la historia de los sistemas basados en Inteligencia Artificial en interacción con el mundo.

Otra aproximación al problema ha consistido en mezclar elementos de procesamiento correspondientes a distintos paradigmas según las necesidades y situaciones para las que se diseña el robot autónomo. A menudo, estas arquitecturas incluyen aspectos de control reactivo y deliberativo, y se suelen dividir en diferentes niveles cualitativamente distintos. Dentro de este tipo de arquitecturas podemos encontrar el sistema AuRA [Arkin, 90], el Atlantis [Gat, 92] o el RAPs [Firby, 89] entre otros. El problema de estas aproximaciones es el decidir o definir qué combinación de paradigmas es adecuado en cada caso y cómo coordinarlos. Por otra parte, seguimos encontrando el problema de explosión combinatoria de complejidad ya mencionado que nos limita la escalabilidad de los sistemas, y el hecho de que necesitamos un diseñador para un entorno o tarea particular.

Aproximaciones recientes

El problema de tratar de evitar un diseño "ad hoc" se ha abordado recientemente desde dos puntos de vista. Por una parte se han diseñado sistemas con capacidad de aprendizaje [Van der Smagt, 95] y por otra se han llevado a cabo procesos evolutivos para la obtención de sistemas completos que realicen las tareas requeridas como [Floreano, 96], [Mataric, 01]. Estas dos propuestas han tenido éxitos parciales. Por una parte, los sistemas con aprendizaje han resultado viables en casos donde se pudiesen definir conjuntos a aprender con claridad, cosa harto difícil en entornos no estructurados y variables donde, en general, los ejemplos de entrenamiento aparecen de forma desordenada y entre grandes cantidades de información irrelevante. Por otra, los sistemas evolutivos han resuelto problemas en entornos simples, mientras que en entornos complejos se encuentra el problema de evoluciones que implicarían tiempos inaceptables, además de problemas de coherencia. Todo ello se describe en el informe de Mataric y Cliff [Mataric, 96].

Desde el punto de vista de las percepciones, se han realizado multitud de trabajos utilizando estrategias algorítmicas tradicionales y estrategias con aprendizaje (distribuidas o evolutivas) para la determinación de la ocurrencia de eventos en conjuntos de percepciones [Sarkar, 93], tales como las que podemos encontrar en la extensa bibliografía sobre procesado de imágenes, reconocimiento de habla, detección de sonidos, etc.

La mayor parte de estos trabajos se han desarrollado como módulos específicos de percepción siguiendo el esquema tradicional de Inteligencia Artificial, es decir, tomando la percepción como algo totalmente independiente del resto del procesamiento necesario para que un robot realice las acciones o comportamientos que requiera. Este planteamiento adolece de una serie de problemas de difícil solución, tales como que se han de decidir en el proceso de diseño cuales son las características de lo percibido que se transmiten al resto de la arquitectura cognitiva, la dificultad de establecer un camino de vuelta de la reflexión a la percepción, la falta de adaptabilidad de estos sistemas en relación con el robot, etc.

En el marco de la robótica basada en comportamiento, se han realizado algunos trabajos que pretenden integrar los sensores y las percepciones dentro de la estructura cognitiva de una forma natural. Algunos autores han tomado ejemplos animales y realizado trabajos que demuestran el valor adaptativo y la integración cognitiva de los sensores de los animales, tales como los de Roitblat [Roitblat, 92] con el oído, Mura [Mura, 94] con la visión de los insectos y la relación de la percepción visual con los eventos motores que permiten el seguimiento de un objetivo o Delcomyn [Delcomyn, 96] con el uso por parte de los insectos de sus órganos de percepción en las patas para caminar.

La mayoría de los sistemas obtenidos adolecen de extrema simplicidad sensorial, es decir, los sensores utilizados han sido extremadamente simples en mundos no muy complicados. Para poder tratar sistemas sensoriales o mundos más complejos, es necesario recurrir a algún tipo de elemento que permita seleccionar la parte de la información que es relevante para los objetivos del robot. En relación con estos mecanismos de atención, Parker [Parker, 92] estudió como se podían utilizar filtros sensoriales como mecanismos selectores de atención, y en esta linea ha habido gran número de trabajos posteriores [Cao, 97], aplicados en su mayoría a tareas cooperativas entre robots. Maes y Foner [Foner, 94] han estudiado otro mecanismo para la selección de atención, por medio del cual su robot aprende a predecir los efectos de las acciones que realiza sobre sus sensores. Estos mecanismos de atención son muy importantes, ya que permiten al robot seleccionar aquella parte del espacio sensorial que es relevante para su supervivencia y objetivos. En vez de tener que aprender todo lo que se puede saber del mundo, sólo ha de aprender aquello que es relevante para él, buscarlo y seleccionarlo. Finalmente, otros autores muestran cómo se podrían diseñar de forma evolutiva los sensores o actuadores de un robot [Dellaert, 95], [Harvey, 94] y [Kodjabachian, 98].

Todas estas aproximaciones a la sensorización de un robot y a su integración dentro de la arquitectura cognitiva se han realizado con sensores simples y comportamientos no muy complicados. Cuando se han tratado sensores más complejos, la integración con el resto de la arquitectura cognitiva ha resultado muy limitada. El problema de fondo es la complejidad de los entornos en los que ha de actuar el robot y el hecho de que si se le quiere dotar de una funcionalidad que pueda ser útil para algo más que problemas triviales, la sensorización del robot y su capacidad de actuación han de superar un mínimo nivel, complicando la estructura interna. Todo esto lleva a un espacio sensorial y de acción muy grande, con la consecuente dificultad de aprendizaje al aparecer lo que es necesario aprender enmascarado por multitud de otras acciones y sensaciones, y por lo tanto dificultando el establecimiento de una relación entre sensorización, acción y consecuencia de la acción que permita llegar a un modelo de mundo utilizable.

Watson [Watson, 94] presenta un sistema que, partiendo de una serie de bloques básicos y secuencias pre-entrenadas, añade la experiencia adquirida en un entorno real a una memoria. Este sistema construye nuevas secuencias de comportamiento y desecha las que no utiliza mediante un algoritmo genético. Nordin [Nordin, 98] emplea un mecanismo con memoria basado en programación genética para obtener una arquitectura cognitiva para un robot Khepera que hace uso de experiencia previa en su interacción con el mundo. Básicamente, incorpora programación genética a un proceso de planificación que se utiliza para optimizar el mejor modelo de mundo actual. Recientemente, se ha aplicado también este sistema a un robot cuadrúpedo [Andersson, 00]. En [Steels, 97], el autor presenta un mecanismo de selección llamado *Selectron*

para evolucionar nuevas tareas de comportamiento en robots. Este mecanismo se utiliza en tiempo real en un robot que opera en un entorno cambiante.

Estos tres mecanismos imponen también una subdivisión de tareas en el sentido de que construyen un modelo de mundo y lo utilizan para probar las posibles estrategias antes de aplicarlas en el mundo real, y aunque están mucho más cerca del objetivo del MDB que los presentados anteriormente, los modelos de mundo que utilizan son dependientes de la tarea a realizar, de modo que si cambia el objetivo no sirve lo aprendido.

Como conclusión a esta primera parte de la revisión del estado actual de la investigaciones, parece que ninguna de la aproximaciones existentes en la robótica actual satisface por completo nuestras exigencias a la hora de conseguir un mecanismo que permita una adaptabilidad eficaz a los cambios del entorno y del objetivo. Pretendemos principalmente que, cuando el agente se desenvuelva en su mundo, la participación del diseñador sea mínima, es decir, queremos proporcionar el soporte del conocimiento, no su estructura. Intentaremos buscar inspiración en la biología y la psicología, áreas donde se ha trabajado desde perspectivas muy diferentes en teorías sobre la inteligencia.

4.4 Bases biológicas

La cuestión general que nos planteamos es la siguiente: ¿qué se puede hacer para dotar a un organismo artificial de un mecanismo mental subyacente, que no predetermine su comportamiento o aprendizaje y que le permita encontrar soluciones creativas de forma autónoma en un tiempo razonable?.

Nos proponemos abordar esta cuestión a través de un grupo de teorías que han sido poco utilizadas por la comunidad de la Inteligencia Artificial y que están basadas en conceptos evolutivos en distintas escalas de tiempo. Según estas teorías, los procesos neuronales de aprendizaje y desarrollo del cerebro dependen fuertemente de una base evolutiva con mecanismos tales como selección y mutación.

Teorías evolutivas del aprendizaje

Cuando un ser nace, lleva en su código genético una serie de mecanismos básicos de manipulación de conocimiento que le permiten adquirir información y clasificarla desde que se forman su cerebro y sus sensores. Utilizando estos mecanismos básicos, el ser aprende a controlar sus funciones motoras, a modelar y predecir su entorno, de forma que puede ir formando representaciones de orden superior, que a su vez le permiten conocer el efecto de las acciones que realiza sobre su estado interno y su entorno. Incluso se puede especular si está dotado de mecanismos que le permiten modificar la forma que toman estos mecanismos básicos.

Tal y como afirma Calvin [Calvin, 87] el ser humano, por lo menos a nivel consciente y para un observador externo, funciona como una maquina serie. Siempre estamos encadenando ideas, objetos, representaciones en nuestra mente, y cuando razonamos lo hacemos en forma de secuencias que parten de un origen y llevan a un fin. El mayor exponente es el lenguaje humano, cuyas reglas llevan una serialidad implícita. Además, cuando pensamos, proyectamos series de acciones sobre escenarios conocidos, pasamos cada acción de la serie por una especie de "analizador de efectos sobre el entorno" y, en función del resultado, pasamos a ver el efecto de la siguiente acción de la cadena. Así

hasta que de un modo u otro encontramos una secuencia que nos lleva al objetivo deseado.

Por otra parte, hay que considerar que, a nivel biológico, el cerebro está constituido por millones de circuitos neuronales que funcionan en paralelo. El problema reside en cómo pasamos de un modo de funcionamiento a otro, o sea, de un paralelismo masivo a una serialidad consciente.

Dentro del campo de las Ciencias Cognitivas, hay cuatro teorías básicas que relacionan la estructura cerebral o neuronal con su funcionamiento desde un punto de vista darwinista:

1. **TSSS: Theory of Selective Stabilization of Synapses [Changeux, 73], [Changeux, 76]:** en esta teoría, se define el aprendizaje como el proceso por el cual un organismo complejo adquiere una capacidad asociativa estable y bien definida como resultado de la interacción específica con su entorno. Está basada en un mecanismo evolutivo que actúa al nivel de sinapsis y contribuye al cableado del cerebro adulto, donde el programa genético dirige la interacción entre grupos específicos de neuronas.

   Los primeros contactos sinápticos, tras la formación del cerebro, aparecen en tres estados (*lábil, estable o degenerado*) pero sólo los dos primeros permiten la transmisión de impulsos nerviosos. El estado *lábil* se puede transformar en *estable* o *degenerado* y esta degeneración es irreversible. Una premisa básica de la teoría establece que las transiciones del estado *lábil* están reguladas por la actividad total de la celda postsináptica, y así la actividad precoz de la red que se está desarrollando estabilizaría de forma selectiva las distintas sinapsis que emergen durante el crecimiento, y crearía diversidad y especificidad.

   Tras la formación del cerebro, hay redundancias en la configuración ya que existe una gran conectividad entre neuronas. La actividad total de la red lleva a la estabilización selectiva de algunas sinapsis y a la regresión de sus equivalentes funcionales, disminuyendo así la redundancia. Es decir, durante el desarrollo del cerebro se eliminan conexiones sinápticas.

2. **TSSP: Theory of Selective Stabilization of Pre-Representations [Changeux, 84]:** esta teoría postula que el aprendizaje y desarrollo en el cerebro adulto tiene lugar a nivel de redes neuronales y se basa en la existencia de *objetos mentales* o representaciones neuronales en el cerebro. Un objeto mental se define como el estado físico alcanzado por medio de la actividad correlacionada y transitoria (tanto eléctrica como química) de un conjunto de células, un gran número de neuronas con distintas singularidades.

   Hay tres clases de *objetos mentales* en la TSSP:

   1- Conceptos primarios: objetos mentales lábiles cuya activación depende de la interacción directa con el mundo exterior a través de estímulos sensoriales.

   2- Representaciones almacenadas: son objetos de memoria cuya evocación no depende de la representación del estímulo que se utiliza para el almacenamiento. No requieren interacción con el entorno y tienen un comportamiento de actividad todo o nada que resulta de un acoplamiento estable y cooperativo entre neuronas. El aprendizaje puede ser visto como el proceso de establecimiento de este acoplamiento físico.

3- Pre-Representaciones: son objetos mentales generados de forma espontánea previamente al contacto con el mundo real. En un instante dado, el cerebro tiene grupos de neuronas activas e inactivas y la activación de grupos de ellas se produce continuamente. La selección de una cierta pre-representación y su almacenamiento en memoria requiere de la interacción con el entorno.

El aprendizaje en un cerebro adulto corresponde a una estabilización selectiva de pre-representaciones. Esto requiere la interacción con el entorno, donde el criterio de selección es la concordancia entre la percepción y la pre-representación.

Una máquina selectiva debe contener dos dispositivos básicos: un *generador de diversidad interna* que utilice un proceso combinatorio y un *mecanismo selectivo darwinista*, dado que un organismo genera de forma espontánea una multiplicidad de diferentes organizaciones previamente a la interacción con el entorno. El generador de diversidad se puede identificar con la producción cerebral de pre-representaciones, y el mecanismo de selección resultaría del acoplamiento entre las percepciones del entorno y la pre-representación.

3. **TELC: Theory of Evolutionary Learning Circuits [Conrad, 74], [Conrad, 76]:** según esta teoría, las capacidades funcionales del cerebro se pueden caracterizar formalmente en términos de un autómata de estados finitos junto con un espacio de memoria que pueda manipular. Son posibles dos tipos de aprendizaje:

   - Aprendizaje basado en modificaciones.
   - Aprendizaje basado en memoria, que es el que trata esta teoría.

   El cerebro debe tener una serie de propiedades, desde el punto de vista molecular, para poder soportar transformaciones graduales que lo adapten a las distintas funciones que debe aproximar. Al mismo tiempo, debe soportar capacidades de cálculo equivalentes a un autómata de estados finitos. El modelo cerebral de Conrad establece tres postulados básicos:

   1- Redundancia: hay muchas copias muy similares de cada tipo de red neuronal local.

   2- Especificidad: las excitasas son capaces de reconocer patrones químicos concretos y hacen que neuronas enzimáticas se disparen como respuesta a patrones de entrada específicos.

   3- Existencia de circuitos internos de selección: que dirigen la modificación de las configuraciones de excitasas en función de cuán buena sea cada red local.

   Se concluye que el aprendizaje por prueba y error está basado en los mismos mecanismos que la evolución natural, aplicado en este caso a circuitos neuronales.

   Conrad presenta, por tanto, un esquema de selección de circuitos, basado en las teorías evolutivas darwinistas, que es la implementación del autómata de estados finitos que caracteriza una parte del funcionamiento cerebral. Presenta además un esquema de neuronas de referencia que le añade al modelo cerebral la capacidad de memoria espacial. De esta forma, el cerebro queda modelado como una máquina de Turing que evoluciona.

4. **TNGS: Theory of Neuronal Group Selection o Neural Darwinism [Edelman, 87]:** esta teoría postula que la selección somática es el mecanismo clave que establece la conexión entre la estructura y la función del cerebro. Durante el

desarrollo pre y post-natal se forma un repertorio primario de grupos neuronales degenerado (grupos no isomórficos pero más o menos isofuncionales) que protege contra la perdida de tejido cerebral. En el periodo post-natal se forma además un repertorio secundario de grupos neuronales operativos por medio de la selección entre los anteriores. Las conexiones entre distintos repertorios de grupos neuronales permite la correlación espacio-temporal de las respuestas de los repertorios a todos los niveles del cerebro.

De estas 4 teorías, se pueden extraer una serie de conclusiones comunes a todas ellas que usaremos como fundamento a la hora de construir el Mecanismo Cognitivo:

- *El cerebro adapta sus conexiones neuronales en tiempo real mediante procesos evolucionistas o seleccionistas.*
- *El aprendizaje y el desarrollo en el cerebro adulto tiene lugar a nivel neuronal.*
- *La selección somática determina la conexión entre la estructura del cerebro y su función.*
- *El cerebro es capaz de adaptarse a la funciones que debe llevar a cabo mediante transformaciones graduales.*

Con estas ideas básicas hemos tratado de dotar a un organismo artificial (agente) de unos mecanismos de funcionamiento cognitivo lo menos predeterminantes posibles, que le permitan un comportamiento creativo y adaptable a los entornos dinámicos en los pueda estar inmerso. El desarrollo mental del organismo sigue un proceso darwinista y es la interacción con el entorno la que va moldeando el cerebro mediante la activación o desactivación de ciertas conexiones cerebrales, de modo análogo al que proponen las teorías expuestas anteriormente. En el MDB, estas conexiones no tienen por qué ser específicamente sinapsis neuronales, sino que de forma más general se van moldeando las capacidades mentales (capacidad de generalización de funciones) del organismo siguiendo un proceso evolutivo.

De este modo damos por terminado este apartado introductorio donde se ha enmarcado el presente trabajo y se ha justificado la originalidad del mismo mediante una revisión de las investigaciones llevadas a cabo en este campo. A continuación pasamos a desarrollar el primer apartado dedicado a la construcción del mecanismo en sí.

Fundamentos de un Mecanismo Cognitivo

5 Fundamentos de un Mecanismo Cognitivo

Tras haber situado el mecanismo que se presenta en este trabajo en el marco actual de las Ciencias Cognitivas, pasamos a formalizar las ideas básicas, que ya fueron presentadas en [Bellas, 01], así como a definir cada uno de los elementos que lo conforman relacionándolos con los conceptos que los inspiran.

5.1 Esquema teórico

A la hora de construir un modelo cognitivo para un agente autónomo, debemos establecer con claridad los condicionantes de la interacción de dicho agente con su entorno:

- Para llevar a cabo cualquier tarea debe existir, en primer lugar, una **motivación** que guíe el comportamiento en función del grado de satisfacción de la misma. Definimos la **motivación** de la siguiente forma:

  Necesidad o deseo que lleva al agente a actuar.

  Todo agente debe poseer una motivación implícita que le lleva a actuar y que puede cambiar con el paso del tiempo.
- De forma general, asumimos la existencia de más de una motivación al mismo tiempo, que vendrán representadas por un vector de motivaciones. Al referirnos a la motivación del comportamiento nos estaremos refiriendo a dicho vector.
- La **percepción externa** s(t) de un agente está constituida por la información sensorial que es capaz de adquirir del entorno en el que se desenvuelve a través de sus sensores externos.
- La **percepción interna** i(t) de un agente está constituida por la información sensorial que posee sobre sus sensaciones internas.
- Definimos la **percepción global** G(t) del agente, que está formada tanto por la **percepción externa** s(t) como por la **percepción interna** i(t).
- El **estado interno** I(t) establece el grado de satisfacción de la **motivación**. Definimos el **estado interno** como una función I de la percepción global, y por tanto, de la percepción interna y externa:

$$I(t) = I[G(t)] = I[s(t), i(t)]$$

- La **percepción externa** depende, a su vez, de la última acción realizada A(t-1) por el agente y de la percepción sensorial del mundo exterior en el instante de tiempo anterior s(t-1):

$$s(t) = W[s(t-1), A(t-1)]$$

- De forma análoga, la **percepción interna** depende de la percepción interna anterior i(t-1) y de la acción aplicada A(t-1):

$$i(t) = P[i(t-1), A(t-1)]$$

- Tenemos, en consecuencia, que el estado interno del agente se puede expresar en función de las entradas sensoriales internas y externas en el instante t-1 y de la acción aplicada en dicho instante:

$$I(t) = I[s(t), i(t)] = I\{[s(t-1), A(t-1)], [i(t-1), A(t-1)]\}$$

- La interacción del agente con su entorno le debe llevar a la satisfacción de la **motivación** que, sin pérdida de generalidad, se puede expresar como la maximización del estado interno, por tanto:

$$\text{máx}[I(t)] = \text{máx}(I\{W[s(t-1), A(t-1)], P[i(t-1), A(t-1)]\})$$

  La única variable susceptible de ser modificada en el proceso de maximización es la *acción*, ya que la **percepción externa y la percepción interna** no se pueden manipular en general.

*El Mecanismo Cognitivo debe explorar el espacio de acciones posibles para maximizar el **estado interno*** (dado que la motivación siempre es conocida, el estado interno también lo será). Esto implica la utilización de un algoritmo de búsqueda de extremos sobre la función I que, a su vez, depende de W y P. Previamente a poder llevar a cabo este proceso, debemos encontrar la función I y, por tanto, las funciones W y P que hemos presentado, aplicando de nuevo un algoritmo de búsqueda de extremos, pero en este caso en un espacio de funciones. En este segundo proceso de búsqueda, se trata de maximizar la similitud de las funciones I, W y P con las funciones reales que representan el estado interno del agente, el entorno y la percepción interna. Es decir, tratamos de modelar la realidad del agente.

Tendremos así cuatro algoritmos de búsqueda de extremos, dos en paralelo para encontrar las mejores funciones W y P, otro para encontrar I y un último que utiliza esta función I para hallar la mejor acción. Las funciones I y W serán renombradas a partir de ahora como **modelo interno** (I) y **modelo de mundo** (W), y en el apartado 6.2.1 discutiremos sus posibles representaciones matemáticas.

5.2 Esquema funcional

A la hora de implementar el desarrollo teórico anterior, hemos tenido que adaptar nuestros deseos a los medios, y el resultado ha sido un mecanismo computacionalmente complejo que requiere de una serie de definiciones previas:

ESTRATEGIA*: Conjunto de n acciones $A_1, A_2,...,A_n$ aplicadas a los k actuadores del agente, donde n es mayor o igual que k, ya que puede existir más de una acción por actuador.*

Es decir, a partir de este momento consideramos una acción individual como la mínima actuación que un agente puede llevar a cabo sobre cada uno de sus actuadores. Así, una acción es una estrategia de longitud uno.

MODELO: *Función que relaciona eventos y percepciones en el instante de tiempo t con percepciones en el instante siguiente t+1, de forma que el conocimiento de esta función permite el modelado temporal del comportamiento de un determinado sistema.*

La búsqueda de un modelo matemático que resuma la interacción de un agente con su entorno en términos de las consecuencias que tiene su comportamiento en el mundo que lo rodea y en sí mismo, es uno de los procesos fundamentales de todo Mecanismo Cognitivo. En general, buscamos una función que relacione la percepción externa *s* (a la que nos referiremos a partir de ahora como entrada sensorial) e interna *i* con el estado interno *I* en los instantes t y t+1 tras aplicar una acción individual *A*. Tratar de obtener un modelo general de este tipo directamente es una tarea inabordable porque relaciona

magnitudes con dependencias entre sí muy complejas, por lo que hemos optado por simplificar el problema dividiendo este modelo general en dos partes, un modelo de mundo y un modelo interno:

MODELO DE MUNDO: *función que permite evaluar las consecuencias de la ejecución de una acción en el entorno. Es decir, dadas m entradas sensoriales* $s(t)_1$, $s(t)_2,...,s(t)_m$ *y una acción A(t) aplicada en el instante de tiempo t, nos proporciona los valores sensoriales en el instante de tiempo siguiente* $s(t+1)_1,s(t+1)_2,...,s(t+1)_m$.

Vemos que coincide con la definición de la función W del apartado anterior.

El mecanismo trabaja con estrategias, es decir, con series de acciones. Una estrategia de longitud *n* (formada por *n* acciones) será aplicada en el modelo de mundo en *n* pasos, de tal forma que, tras aplicar la primera acción en el instante t, obtenemos las entradas sensoriales en t+1, lo que nos permite obtener mediante el mismo modelo de mundo las entradas sensoriales en t+2, y así sucesivamente.

MODELO DE PERCEPCIÓN INTERNA: *función que permite evaluar las consecuencias de la ejecución de una acción en las sensaciones internas del agente. Es decir, dadas m entradas sensoriales internas* $i(t)_1,i(t)_2,...,i(t)_m$ *y una acción A(t) aplicada en el instante de tiempo t, nos proporciona los valores sensoriales internos en el instante de tiempo siguiente* $i(t+1)_1,i(t+1)_2,...,i(t+1)_m$.

El modelo de percepción interna coincide con la definición de la función P del apartado anterior. La existencia de este modelo depende de los sensores internos que posea el agente, algo que no es común. Ejemplos de este tipo de sensores serían: un sensor de hambre, de sed, de deseo sexual, de dolor, etc, es decir sensaciones que no dependen del mundo exterior al agente y que influyen en el estado interno. Así, este modelo proporcionaría, por ejemplo, la sensación de hambre actual en función de la sensación anterior y de la acción, es decir, si el agente ha comido o no. Las acciones A(t) que se utilizan en el modelo de percepción interna no tienen por qué ser las mismas que se utilizan en el modelo de mundo, sino que serán únicamente las relevantes desde el punto de vista de las sensaciones internas.

Los sensores internos se caracterizan por la subjetividad de los parámetros con los que están relacionados. Como primera aproximación, en el esquema funcional del MDB tendremos en cuenta modelos de percepción interna directos, de modo que la relación entre i(t) e i(t+1) será trivial.

MODELO INTERNO: *función que permite predecir el estado interno I(t) a partir de las entradas sensoriales* $s(t)_1,s(t)_2,...,s(t)_m$.

Un modelo interno relaciona entradas sensoriales y estado interno en el mismo instante de tiempo. En la práctica, los modelos internos nos proporcionarán el estado interno tras aplicar cada acción de una estrategia de longitud *n*.

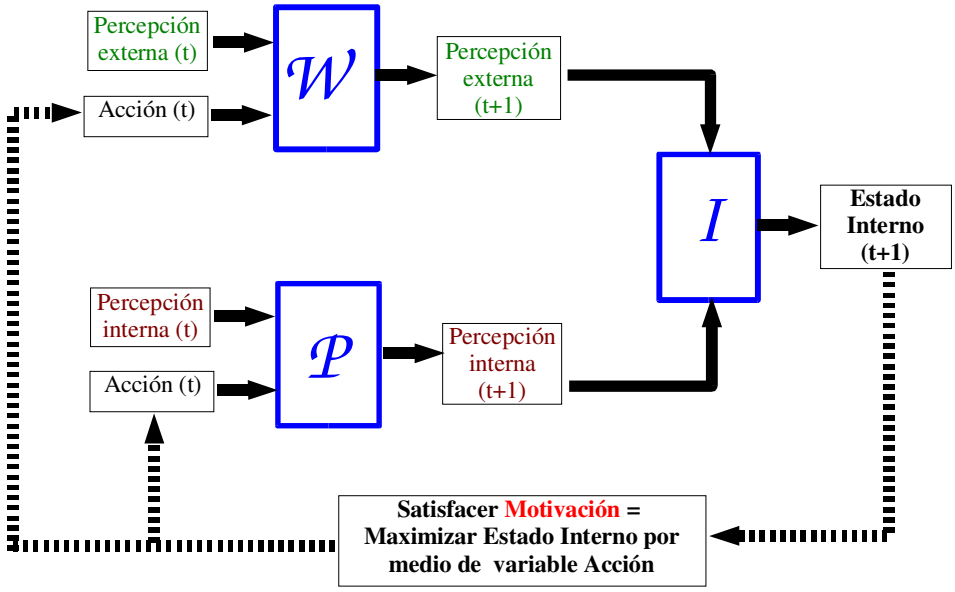
Por tanto, utilizaremos un concepto de modelo interno que coincide con la definición de la función I del apartado anterior, con la salvedad de que no existe percepción interna en nuestros agentes. Es decir, estamos asumiendo que la entrada sensorial predicha por el modelo de mundo coincide con la percepción interna en dicho instante. Por ejemplo, en el caso de un sensor interno de hambre y uno externo de distancia, estamos asumiendo que si la distancia es, supongamos, pequeña, la sensación de hambre disminuye.

PAR ACCIÓN-PERCEPCIÓN: *conjunto de valores formado por las entradas sensoriales y el estado interno relacionados con la aplicación de una estrategia en el entorno real, que se utiliza como patrón a la hora de perfeccionar los modelos:*

Entradas Sensoriales (t)	Estrategia de longitud n	Entradas Sensoriales (t+n)	Estado interno (t+n)

Debemos incidir en el hecho de que los modelos en que se fundamenta este mecanismo son funciones mientras que los pares acción-percepción son conjuntos de valores formados por las entradas y salidas (muestras conocidas del espacio de trabajo) de dichos modelos. Cada par acción-percepción contiene una muestra del modelo de mundo y una muestra del modelo interno, reflejando el hecho de que el conocimiento del que se dispone de la función real que representa al entorno y la función real que representa el estado interno es discreto.

El esquema funcional que concreta el modelo cognitivo establecido en el apartado anterior es el siguiente:



donde hemos realizado ciertas modificaciones sobre el esquema teórico, ya que las entradas al modelo de mundo W y al modelo de percepción interna P corresponden al instante actual t, mientras que la salida corresponde al instante posterior t+1 reflejando así el hecho de que se trata de predicciones. Lo mismo ocurre con el modelo interno I, que proporciona el estado interno predicho en t+1 a partir de los datos de percepción externa e interna en dicho instante.

Hemos de recalcar que, aunque el modelo de percepción interna P se muestra de forma explícita en el esquema anterior por generalidad, a la hora del estudio de todos los aspectos que inciden en su operación, su comportamiento es totalmente análogo al modelo de mundo salvo las entradas a las que afecta. Por lo tanto, en el desarrollo posterior se considerará únicamente el modelo de mundo y, de acuerdo con la definición

de modelo interno anterior, la percepción interna no constituye una entrada al modelo interno I. También es importante indicar que en muchos casos de aplicación real, los modelos de percepción interna son funciones directas de situaciones predichas para el entorno y, en consecuencia, las funciones que los describen son triviales.

De acuerdo con el esquema teórico, la búsqueda en el espacio de acciones se lleva a cabo para maximizar el estado interno que, como vemos en el esquema funcional, se obtiene como resultado de las predicciones obtenidas por el modelo de mundo y el modelo interno. Es decir, tras la obtención del modelo de mundo e interno en sendos procesos de búsqueda en el espacio de funciones, pasamos a realizar la búsqueda en el espacio de acciones tratando de maximizar la función que resulta de combinar ambos modelos, combinación que ha surgido por la dificultad de obtener un único modelo matemático para predecir directamente el estado interno tras la aplicación de una acción.

Tal y como hemos establecido, la búsqueda en el espacio de modelos toma como patrón la realidad. Por un lado, los modelos de mundo toman como patrón el entorno real, de modo que la maximización de la función que modela a dicho entorno real nos proporciona el mejor modelo de mundo posible. A su vez, los modelos internos toman como patrón la satisfacción de la motivación del agente, es decir, otra función real que se debe maximizar. La forma en que el agente puede conocer ambas funciones reales es a través de sus sensores. Los sensores externos le proporcionan datos discretos de la función real que modela el entorno, mientras que la combinación de los sensores externos e internos le proporcionan datos sobre la función real que modela su estado interno, o lo que es lo mismo, sobre el grado de cumplimiento de las motivaciones. De esta forma, podemos decir que el agente conoce la realidad a base de muestras (de forma discreta), y que a partir de estas muestras, debe llevar a cabo el proceso de búsqueda de los modelos.

A continuación pasamos a analizar brevemente uno de los aspectos más interesantes y fundamentales del modelo cognitivo que acabamos de presentar, las motivaciones que guían el comportamiento.

5.3 Motivaciones del comportamiento

A la hora de plantearse un mecanismo de este tipo para un organismo, surge el problema de las motivaciones del comportamiento, ya que debemos preguntarnos qué lleva a un ser a escoger una cierta estrategia. Para que cualquier agente se adapte a un entorno o realice una actividad, ha de tener algún tipo de motivación que le permita establecer sus objetivos, que le de una razón para gastar energía y que le permita ordenar su representación del mundo en el que se desenvuelve.

Uno de los caminos más interesantes para el estudio de la motivación plantea que el conjunto posible de alternativas de comportamiento estén compitiendo por la atención [Heiligenberg, 74]. En este tipo de estudios, se define un sistema de prioridades de forma que algunos tipos de estímulos tengan prioridad sobre otros. Además, las prioridades de comportamiento se pueden ajustar de forma que se equilibren la urgencia de las necesidades respecto a las oportunidades en el entorno y la calidad de éstas [Gould, 84].

Nuestro planteamiento respecto a la motivaciones está próximo al *Utilitarismo* [Bentham, 1789], teoría ética que afirma que una acción es buena si lleva a estados de utilidad positiva mayor y es mala si lleva a estados de utilidad negativa mayor. Es decir,

adoptamos un punto de vista totalmente práctico donde la utilidad máxima posible será la satisfacción de la motivación del comportamiento. En cada caso particular, la utilidad de las acciones aplicadas y, por tanto, la motivación depende de la tarea a realizar [Mascaro, 01].

Hasta ahora, en la mayor parte de los sistemas computacionales, las motivaciones eran externas, es decir, un programador establecía qué motivaba la acción del sistema de una forma directa. Estos objetivos externos implican una rigidez muy clara a la hora de alcanzar los fines, llevando al final a comportamientos muy estereotipados y a soluciones poco originales para los problemas. Desde luego, lo que no proporciona este tipo de soluciones es una autonomía plena del agente.

En general consideramos que, para que un agente pueda actuar de forma autónoma y sobrevivir en entornos cambiantes, ha de contener sus motivaciones dentro de sí mismo. En el caso de los seres vivos, la propia selección y evolución natural ha eliminado aquellos miembros que no tenían las motivaciones necesarias para adaptarse a los entornos y circunstancias que iban surgiendo. Las motivaciones se traducen en impulsos para realizar ciertas acciones. Estos impulsos ponen en funcionamiento ciertos programas, que pueden ser genéticos o aprendidos, y dirigen la acción del individuo en la medida de su fuerza. Es decir, focalizan la atención del individuo en aquellos aspectos de su entorno que son relevantes para llevar a cabo la acción o acciones asociadas al impulso y que conducen a satisfacer la motivación de dicho impulso.

Tal y como se estableció en el apartado anterior, la motivación que guía el comportamiento de un agente está formada, en general, por distintas motivaciones que pueden entrar en conflicto en un momento dado de modo que, para que el agente pueda operar, tiene que existir un mecanismo para resolver dichos conflictos. Este es un aspecto que no abordaremos en el presente trabajo, pero la existencia de un vector de motivaciones debe ser enfocada mediante una escala de valores que decida cuál de dichas motivaciones tiene mayor relevancia en un cierto instante.

En general, en el MDB asumimos que las motivaciones son siempre fijas y podemos establecer que el vector de motivaciones en un momento dado se corresponde con una función que depende de las sensaciones subjetivas del agente, es decir, de aquello que le produce placer o satisfacción. Esta función, puede ser una relación simple entre motivación y satisfacción como la que usamos en este trabajo, u otra más compleja que, por ejemplo, podría llevar al agente a desear no comer en contra del impulso natural de supervivencia.

Existe, por tanto, una infinidad de matices que considerar en el terreno de la motivación ya que nos estamos adentrando en temas abiertos hoy en día en el campo de la psicología y que exceden el marco del presente trabajo. De hecho, en los ejemplos que se presentarán en el apartado 7, utilizaremos motivaciones simples que consisten en la minimización de una distancia o en la maximización de los premios recibidos. Enfocamos así el mecanismo de modo que el agente tiende a satisfacer motivaciones básicas o primarias.

Una vez presentados los fundamentos matemáticos y los elementos básicos del modelo cognitivo, nos planteamos a continuación la interconexión de dichos elementos para dotar al modelo de un modo de funcionamiento que cumpla los objetivos marcados.

5.4 Construcción de un Mecanismo Cognitivo

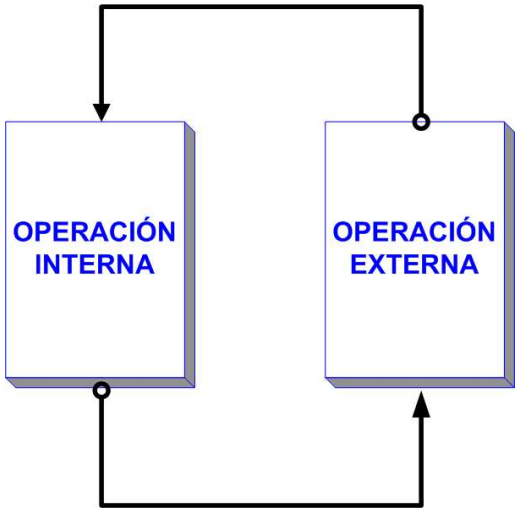
El objetivo de este apartado consiste en establecer con claridad cuáles son los bloques básicos de cualquier mecanismo que se base en un modelo cognitivo como el que acabamos de exponer. Para ello, comenzaremos con estructuras simples, es decir, bloques muy generales, para llegar a un esquema final de mecanismo más concreto y complejo.

Como definimos en el apartado anterior, nuestro agente va a tener tres tipos de representaciones mentales: una representación de posibles estrategias, un conjunto de modelos del mundo y otro de modelos internos. Al interactuar con el entorno, percibe las consecuencias de las estrategias que escoge en forma de nuevos valores en los sensores que le dirán si ha acertado o no al realizar dichas acciones. Esta realimentación le permite perfeccionar los modelos de mundo y los modelos internos, de modo que las representaciones mentales se ajustan cada vez más a la realidad. Recurriendo a las teorías biopsicológicas introducidas en el apartado 4.4, vamos a establecer una jerarquía de dos niveles formada por conjuntos de procesos paralelos *inconscientes* que llevan, tras un tiempo de procesado, a una solución *consciente* por selección.

El problema reside en cómo combinar estrategias, modelos, sensores, actuadores, motivaciones y entorno para construir un conjunto cerrado que permita al agente llevar a cabo una cierta tarea con éxito. Esta combinación implica un modelo de interacción entre todos estos elementos que consiga un funcionamiento fluido en tiempo real, y que a la vez verifique los requisitos que se han planteado para que exista inteligencia, es decir, posibilidad de aprendizaje, creación de soluciones originales, etc.

De estas interacciones, la del agente como sistema basado en un modelo cognitivo (modelos, estrategias, sensores y actuadores) con el entorno es muy clara. Percibe el entorno por medio de los sensores y actúa sobre él por medio de los actuadores y, en principio, los modelos y las estrategias están aisladas de una interacción directa con el entorno.

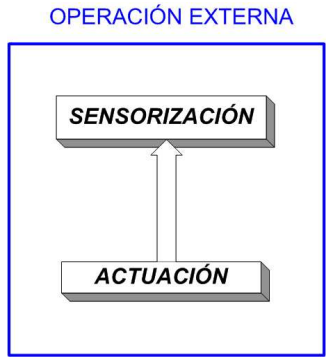
Por tanto, una primera representación esquemática de los bloques fundamentales que forman el mecanismo podría ser la siguiente:



donde simplemente hemos dividido el funcionamiento del mecanismo en operación interna y externa. De la aplicación de una estrategia en el entorno, el agente obtiene información real a través de sus sensores. Esta información es la única de la que dispone para obtener los modelos de mundo y modelos internos que utiliza en la operación interna, con lo que la relación entre ambos bloques es simple y se reduce a un bucle de realimentación.

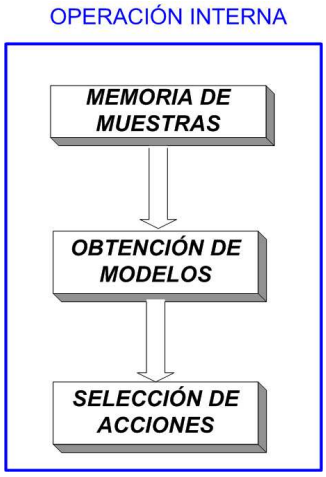
Es evidente que la imagen que cualquier agente pueda tener internamente de su entorno está en función de los sensores externos que posea y, de igual forma, el modelo de percepción interna va a venir determinado por los sensores internos de que disponga. Por ejemplo, si un cierto agente tiene un sensor binario de percepción interna HAMBRE-NO HAMBRE, un sensor binario externo de distancia a la comida CERCA-LEJOS y su única motivación es SOBREVIVIR, sus acciones de manera natural deben ir encaminadas a conseguir estar CERCA de la comida, que la percepción interna sea NO HAMBRE de modo que el grado de satisfacción de la motivación, o sea, el estado interno, sea máximo. De este modo, los modelos de mundo sólo pueden contemplar la información que el agente tenga sobre el entorno y sobre sí mismo, es decir, la información que proporcionen los sensores y en el formato que éstos la proporcionen. Por otra parte, las estrategias se van a limitar a las acciones que se puedan realizar sobre el entorno, que están en función de los actuadores que el agente posea. Estos dos puntos son muy importantes, ya que debe quedar claro que la "vida" de cualquier agente está limitada por sus sensores y actuadores, y éstos van a determinar en gran medida al agente especialmente su facilidad de adaptación al entorno.

Cada uno de estos dos primeros bloques, operación externa y operación interna, puede ser dividido de forma más concreta. Por una parte, la operación externa se puede ver como la combinación de un proceso de actuación, es decir, de aplicación de las acciones seleccionadas al mundo real a través de los actuadores del agente, y de un proceso de sensorización posterior para conocer las consecuencias de dichas acciones. Este bloque quedaría, por tanto dividido en dos:



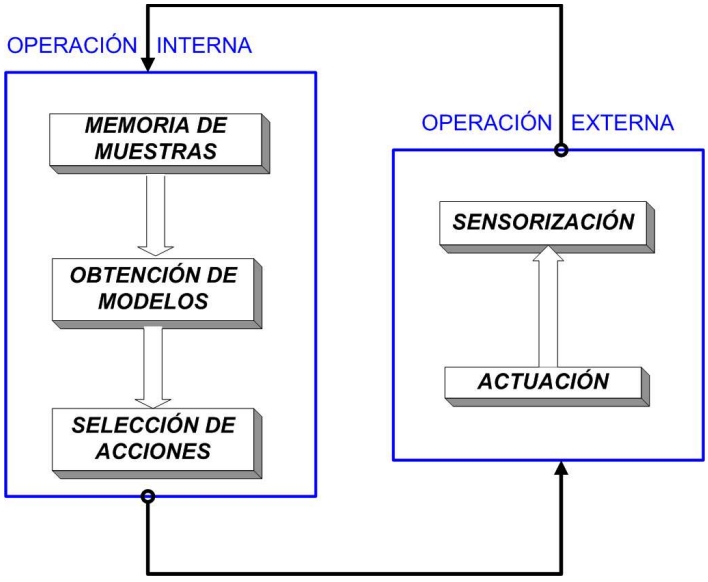
Por otra parte, la operación interna requiere de un proceso de búsqueda de extremos en el espacio de modelos para poder llevar a cabo la búsqueda en el espacio de acciones. Ambos son procesos típicos en las distintas arquitecturas de agentes presentadas en el apartado 4.2.2, ya que representan un proceso de selección de acciones basado en representaciones internas. En este sentido, el Mecanismo Cognitivo que proponemos es deliberativo aunque no en el sentido clásico, ya que dichos modelos son dinámicos como veremos más adelante. Por tanto, el bloque de operación interna puede ser

dividido en dos, uno de obtención de modelos y otro de obtención de acciones. Como hemos explicado en el apartado anterior, los modelos son seleccionados en un proceso de maximización a partir de muestras de la realidad. Esto implica que, internamente, el agente debe poseer dichas muestras, de modo que se hace necesaria la utilización de una memoria que las almacene. Esto nos conduce a un nuevo esquema para el bloque de operación interna:



donde la obtención de los modelos se realiza a partir de los datos contenidos en la memoria y la búsqueda de la mejor acción se lleva a cabo tras la de los modelos. La conexión entre la operación externa después de la sensorización y la operación interna se lleva a cabo en el bloque de memoria, donde se almacenan las muestras obtenidas de la realidad, es decir, los pares acción-percepción.

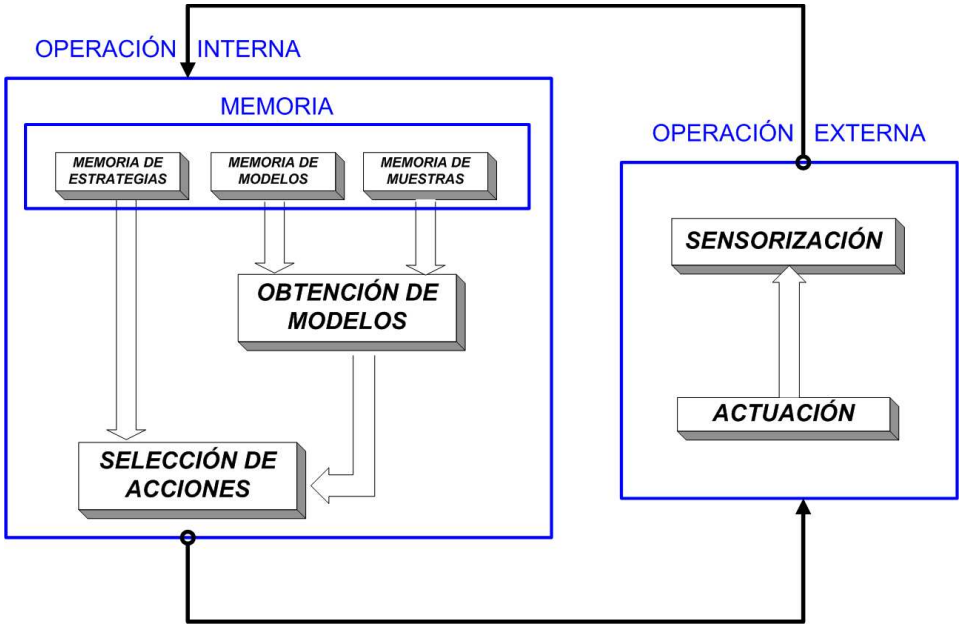
Si combinamos las nuevas estructuras de operación interna y externa en un mismo esquema, obtenemos:



Estos bloques conforman la estructura básica de cualquier Mecanismo Cognitivo basado en un modelo similar al que hemos introducido en el apartado 5.2.

Existe otro elemento que debe ser añadido al esquema anterior si pretendemos dotar al mecanismo de eficiencia práctica real. Hasta ahora no hemos hecho ninguna suposición en torno al proceso de búsqueda de los modelos, ya que corresponde a la implementación práctica que es el tema del siguiente apartado. Pero lo que sí podemos establecer sin pérdida de generalidad, es que los modelos no tienen por qué ser los mismos a lo largo del tiempo de vida del agente. En este sentido, se hace necesaria una nueva memoria que almacene los modelos que han sido obtenidos con anterioridad, de cara a optimizar la búsqueda de los mismos en situaciones futuras. Es decir, si el proceso de maximización de las funciones reales parte de unas condiciones iniciales que lo aceleran, estaremos optimizando el funcionamiento del mecanismo. Asimismo, las acciones (estrategias en general) obtenidas a lo largo del tiempo pueden ser reutilizadas en el futuro optimizando de nuevo el proceso de búsqueda.

Estas memorias (de modelos y estrategias) pueden ser englobadas junto con la memoria de muestras en un único bloque de memorias, de modo que el esquema final que proponemos para el Mecanismo Cognitivo es el siguiente:



Con estos elementos, tenemos las piezas básicas para el desarrollo de un Mecanismo Cognitivo genérico. Hemos realizado el planteamiento teórico intentando establecer los fundamentos matemáticos y computacionales en que se basa un mecanismo que proporcione a un agente la posibilidad de seleccionar las acciones que debe aplicar para llevar a cabo una cierta tarea. Pero un mecanismo de este tipo requiere abordar otros aspectos como son el aprendizaje, la adaptabilidad, las motivaciones del comportamiento, etc que, por ser más concretos y específicos, no se reflejan en el esquema anterior. Realmente son estos aspectos los que proporcionan al mecanismo un funcionamiento dotado de inteligencia y requieren un estudio a fondo.

En el siguiente apartado abordaremos la construcción computacional de este esquema. Para ello entraremos en el estudio de las distintas técnicas para la búsqueda de los modelos y de las acciones además de analizar cómo incorporar aprendizaje al esquema básico.

Implementación del MDB

6 Implementación del MDB

Este es el apartado más extenso de esta tesis y aglutina gran parte de los elementos innovadores que se presentan. Es por ello el apartado que más labor de investigación ha requerido y donde hemos centrado nuestros mayores esfuerzos.

Ha sido estructurado en 3 partes fundamentales, que son las siguientes:

1 **Diagrama de bloques del MDB:** donde planteamos la estructura del mecanismo a partir del esquema básico del apartado anterior realizando una breve introducción de cada elemento.

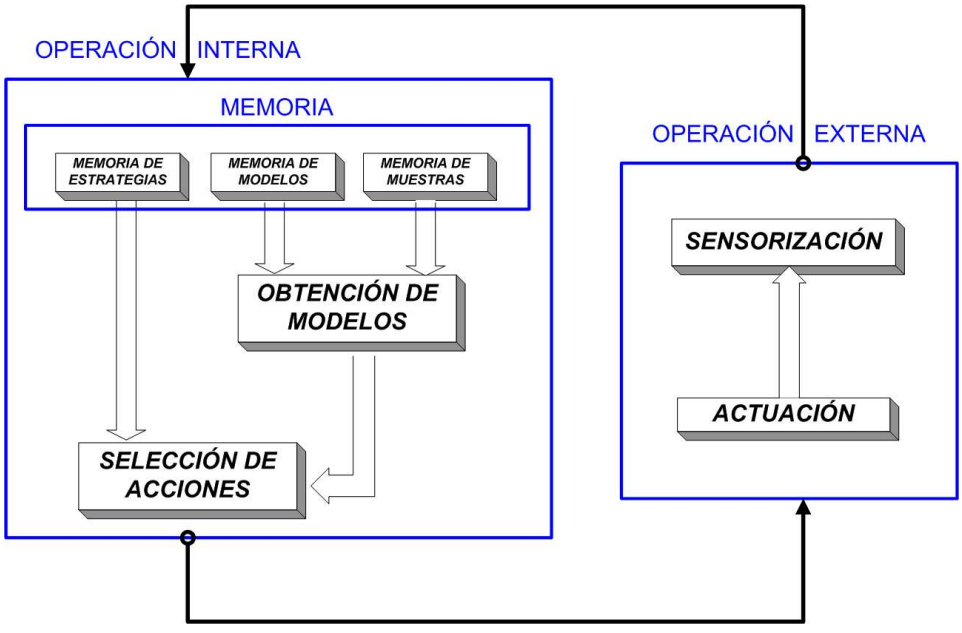
2 **Análisis en profundidad de los elementos básicos:** donde estudiaremos las dos partes fundamentales del mecanismo, la obtención de modelos y acciones y la gestión de las memorias.

3 **Mejoras sobre el esquema básico:** donde se plantea una metodología que permita la obtención y combinación automática de los modelos aprendidos, generalizando así los procesos de búsqueda y aprendizaje que tienen lugar en el mecanismo.

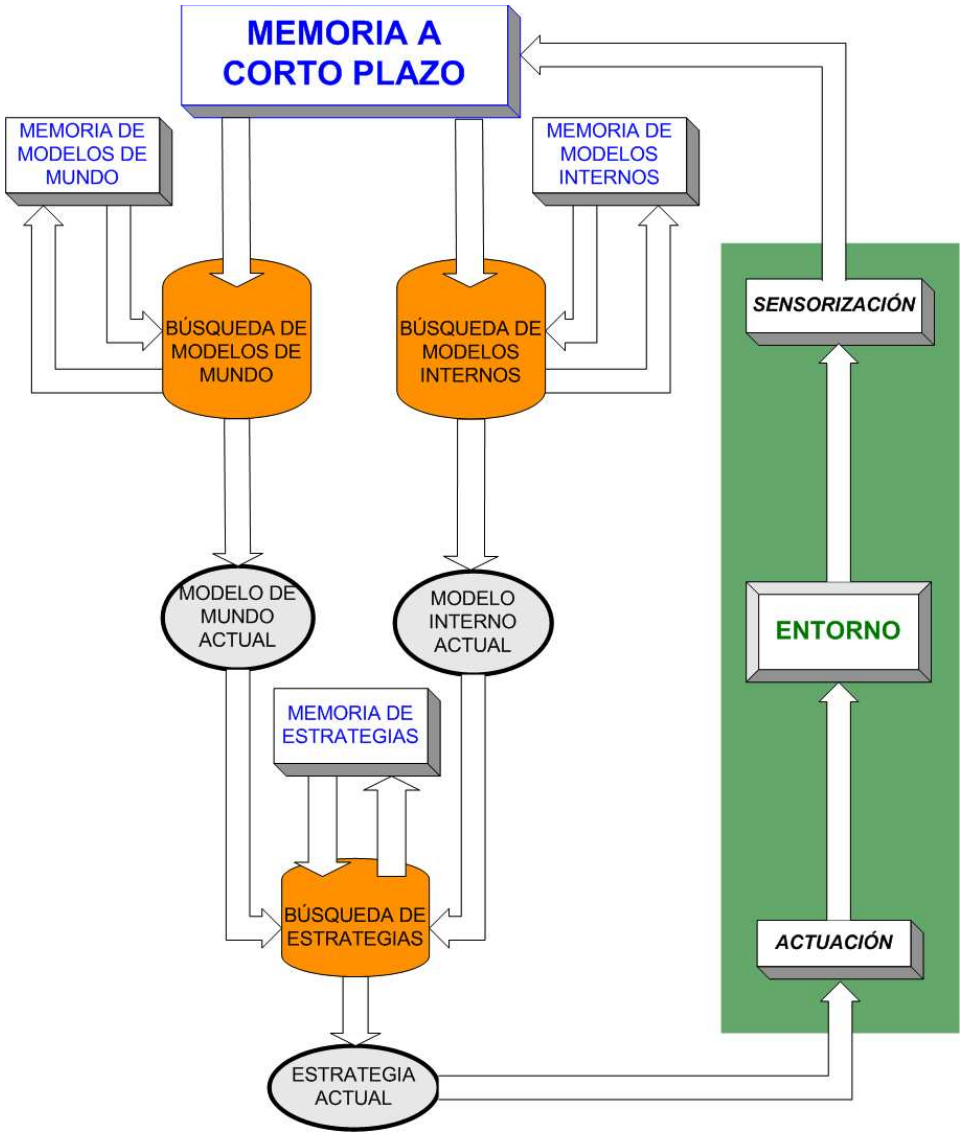
Por tanto, comenzamos el estudio de la implementación real del MDB planteando el diagrama de funcionamiento del mismo.

6.1 Diagrama de bloques del MDB

A partir del esquema final que hemos establecido en el apartado anterior para cualquier Mecanismo Cognitivo que utilice un modelo como el presentado en el apartado 5.2, hemos creado un diagrama de bloques básico que resume el funcionamiento del MDB. Por tanto, la estructura debe respetar el esquema final del apartado anterior, es decir:



Teniendo esto en cuenta, hemos desarrollado el siguiente diagrama de funcionamiento básico para el MDB:



La operación que se representa en este diagrama puede ser analizada comenzando por su punto final, es decir, por la selección de la acción que debe ser aplicada, ya que éste es el objetivo del Mecanismo Cognitivo. Por tanto, si partimos del bloque *Estrategia Actual*, que representa la secuencia de acciones que ha sido seleccionada para ser aplicada en un instante dado, el ciclo básico de funcionamiento es el siguiente:

1. La estrategia seleccionada es aplicada al *Entorno* a través de los actuadores del agente, representados en el bloque *Actuación*, obteniendo una *Sensorización* como consecuencia.

2. Los datos de actuación realizada y sensorización obtenida proporcionan un par acción-percepción que es almacenado en la *Memoria a Corto Plazo (MCP).*
3. Con los pares contenidos en esta memoria se llevan a cabo los procesos de *Búsqueda del Modelo de Mundo* y *Búsqueda del Modelo Interno* tratando de maximizar la adaptación a las funciones reales representadas por los pares acción-percepción. La *Memoria de Modelos de Mundo* y *la Memoria de Modelos Internos* almacenan modelos que han resultado satisfactorios en cada caso para optimizar futuras búsquedas, y representan la Memoria a Largo Plazo (MLP).
4. Los mejores modelos obtenidos en los procesos de *Búsqueda del Modelo de Mundo* y *Búsqueda del Modelo Interno*, representados por los bloques *Modelo de Mundo Actual* y *Modelo Interno Actual*, se utilizan en la *Búsqueda de Estrategias* que, de nuevo, tiene asociada una *Memoria de Estrategias* para optimizar futuras búsquedas.
5. La mejor de las estrategias, *Estrategia Actual*, es aplicada al *Entorno* en una etapa de *Actuación* obteniendo una nueva *Sensorización.*

Cada uno de estos ciclos básicos comienza y finaliza en el bloque *Estrategia Actual* y de ahora en adelante lo denominaremos *iteración*. Por tanto, el paso de las iteraciones hace que cada vez sea mayor el número de pares acción-percepción que se pueden utilizar en la búsqueda de modelos y, en consecuencia, dichos modelos serán cada vez mejores. Esta mejoría implica que las acciones seleccionadas a partir de estos modelos sean cada vez más satisfactorias.

A continuación haremos una breve descripción de cada de los bloques básicos antes de abordar su estudio en profundidad:

6.1.1 Entorno

Este bloque representa el mundo real, es decir, el entorno donde el agente se desenvuelve. El MDB ha sido diseñado para ser aplicado a agentes que habitan en entornos dinámicos complejos, es decir, entornos reales. Al trabajar desde el principio en las condiciones más adversas posibles, el mecanismo resultará robusto y fácilmente aplicable a entornos simples.

La operación externa de todo Mecanismo Cognitivo tiene lugar en el entorno, del que dependen tanto la actuación como la sensorización. En la bibliografía del campo también se denota al entorno como ambiente, sobre todo al referirnos a agentes situados en el mundo real. Tal y como hemos introducido en el apartado 4.3, cualquier entorno real posee cinco características que lo hace especialmente complejo: es inaccesible, no determinista, no episódico, dinámico y continuo.

6.1.2 Sensorización

Tras aplicar una estrategia en el entorno, el agente debe sensar las consecuencias de dicha estrategia. Esto implica una nueva percepción externa que localiza al agente en su entorno y una nueva percepción de estado interno. De los sensores se obtienen los valores que permiten el posicionamiento consciente en el entorno, es decir, constituyen la conexión de un ser con su mundo.

En robótica autónoma, tal y como explicamos en el apartado 4.3, se suele trabajar con sensores muy simples y poco precisos como, por ejemplo, sensores infrarrojos, sensores de luz, de contacto, etc. Introducir sensorización más potente como la visión

artificial, es fundamental para dar un salto de calidad en lo que se puede hacer actualmente en robótica aproximando a nuestra forma de ver el mundo la del robot con el que debemos trabajar.

6.1.3 Memoria a Corto Plazo

Tras la ejecución de una estrategia en el entorno y de sensar sus consecuencias, guardamos el par acción-percepción resultante en una memoria para ser utilizado en los procesos de búsqueda de modelos. Esta memoria no puede ser infinita por limitaciones computacionales obvias, de modo que almacenará una parte de dichos pares acción-percepción. Es una tarea de diseño que analizaremos con detalle en el apartado 6.2.2 el decidir qué pares se deben almacenar. Es decir, se hace necesario el desarrollo de una estrategia de reemplazo para esta memoria.

Este tipo de memorias son ampliamente utilizadas en la bibliografía del campo y se denominan memorias a corto plazo (MCP), reflejando el hecho de que suelen almacenar datos de reciente adquisición.

6.1.4 Búsqueda de modelos de mundo, modelos internos y estrategias

Como establecimos en el apartado 5.2, estos dos primeros bloques representan el proceso de búsqueda de modelos lo más parecidos posible a las funciones que representan al mundo real y al estado interno del agente. La búsqueda se debe realizar sin conocer analíticamente las funciones a aproximar, sino a través de muestras de las mismas, es decir, a través de los pares acción-percepción. Una vez obtenidos los mejores modelos posibles en cada iteración (que no serán los óptimos hasta que se haya recabado suficiente información) se realiza la búsqueda de la estrategia sobre ellos.

Existen distintas técnicas que pueden ser aplicadas para llevar a cabo esta búsqueda. Las más representativas serán analizadas en el apartado 6.2.1.

6.1.5 Memorias de modelos de mundo, modelos internos y estrategias

Este tipo de memorias almacenan modelos y estrategias que han resultado satisfactorios en una cierta iteración de cara a optimizar los recursos del agente, ya que en futuros procesos de búsqueda se puede acelerar la obtención de los resultados. Se les denomina memorias a largo plazo (MLP) en referencia a que la información que almacenan no suele ser reciente y se mantiene durante largos periodos de tiempo.

Este es un tema complejo que requiere de un análisis en profundidad y al que dedicaremos el apartado 6.2.3.

6.1.6 Actuación

Esta es la última etapa del mecanismo y en ella se ejecuta en el entorno la estrategia seleccionada internamente. El tiempo de ejecución de una acción o estrategia (secuencia de acciones) depende de la naturaleza de los actuadores, de modo que la aplicación puede ser instantánea o bien puede llevar un cierto tiempo. Por ejemplo, que un brazo se mueva es una acción inmediata mientras que un aumento de temperatura por medio de un calefactor requiere un cierto tiempo antes de poder analizar sus consecuencias.

Lógicamente, la ejecución de una estrategia durará más cuanto mayor sea el número de acciones que incluye.

Siguiendo el flujo de ejecución del mecanismo a partir del diagrama de bloques, la siguiente etapa es de nuevo el entorno, es decir, la aplicación de la estrategia sobre los actuadores se manifiesta en el entorno del agente. Aquí termina la iteración actual y comienza la siguiente (n instantes de tiempo después para una estrategia de longitud n). Después tenemos una etapa de sensorización que da lugar a un nuevo par acción-percepción y, consecuentemente, a una nueva entrada en la Memoria a Corto Plazo.

Hasta aquí el planteamiento inicial del diagrama de bloques básico del MDB. A continuación pasamos a estudiar con detalle los principales procesos que tienen lugar en el mecanismo: la búsqueda de modelos y acciones y la gestión de las memorias.

6.2 Análisis de bloques básicos

Los procesos de búsqueda de modelos y acciones conforman la base de todo Mecanismo Cognitivo. Es en estos procesos donde se pueden incluir las capacidades de más alto nivel, como el aprendizaje y la adaptabilidad, y en ellos nos centramos a continuación. Más tarde analizaremos con detalle las memorias que son necesarias en el mecanismo y cómo interactúan con los procesos de búsqueda.

6.2.1 Búsqueda de modelos y acciones. Aprendizaje

Lo primero que debemos analizar es qué son realmente los modelos y las acciones, es decir, cuál es su sustrato computacional y su base matemática.

Sustrato de los modelos

Un modelo es una función no lineal definida en un espacio n-dimensional que aproxima y trata de predecir características reales. En el caso de los modelos de mundo, estamos suponiendo que el entorno es modelable, al menos por partes, y en el caso de los modelos internos suponemos que el nivel de satisfacción del agente también es predecible. Como sustrato para dichos modelos, por tanto, requerimos un aproximador matemático no lineal que pueda ser actualizado y modificado en función de la cantidad de muestras de que se disponga.

Teniendo esto en cuenta, para los modelos podemos usar muy distintas representaciones matemáticas, de las que mencionaremos cuatro por su especial relevancia: las funciones polinómicas (por ser la representación más simple y directa que existe), las reglas (por ser la representación clásicamente más utilizada), la lógica borrosa (por ser la representación basada en lógica más extendida actualmente en el campo de la Inteligencia Artificial) y las redes neuronales artificiales (por ser especialmente adecuadas a nuestros problemas):

- **Funciones polinómicas:** una función polinómica n-dimensional constituye la representación matemática más directa que podemos utilizar, ya que las funciones que aproximan pueden ser actualizadas mediante técnicas de interpolación o ajuste que actúen sobre los coeficientes polinómicos [Kincaid, 94]. Constituyen una representación simple y computacionalmente eficiente para problemas con bajo número de variables.

- **Reglas:** en lugar de utilizar una función matemática exacta, en el campo de la Inteligencia Artificial [Rich, 94] se ha tendido ha resolver los problemas desde otro punto de vista más práctico. Para ello se utilizan reglas que relacionan las entradas y las salidas en forma de condición-acción, es decir, reglas del estilo:

  si *condición* ***entonces*** *acción.*

  En la Inteligencia Artificial Clásica se asume que cualquier proceso computacional capaz de resolver instancias de un problema no algorítmico necesariamente debe dar la apariencia de un comportamiento inteligente. Dentro de la Inteligencia Artificial, básicamente hay dos corrientes científicas: la simbólica y la subsimbólica.

  Los sistemas basados en reglas aplicados en Inteligencia Artificial se denominan Sistemas Expertos [Giarrantano, 94] y en su propia definición reside el hecho de que requieren gran participación del conocimiento del diseñador, con lo que las capacidades autónomas quedan reducidas. Para los constructores de sistemas expertos, es fundamental la representación del conocimiento humano. En un sistema experto hay dos tipos de conocimiento: conocimiento acerca del problema particular o *declarativo* y conocimiento acerca de cómo obtener más conocimiento a partir del que ya tenemos o *procedural*. Para el conocimiento declarativo existen técnicas como los frames que fueron los padres de lo que hoy conocemos como Programación Orientada a Objetos. El conocimiento procedural también es llamado mecanismo de inferencia y requiere además de un método de búsqueda que permita tomar decisiones, como por ejemplo, seleccionar la regla a aplicar del conjunto total de posibles reglas. Cualquier modelo de un Mecanismo Cognitivo puede ser codificado a base de reglas, pero los problemas aparecen a la hora de actualizar dichas reglas de forma automática mediante nuevas muestras.

- **Lógica borrosa:** la lógica borrosa se define como una lógica multivaluada que se ha extendido para manejar conceptos de verdad parcial, es decir, trabaja con valores intermedios entre totalmente falso y totalmente cierto. Esta lógica nació en los años 60 en la Universidad de California en Berkeley, introducida por Lofti A. Zadeh [Zadeh, 65] como la lógica del razonamiento aproximado, y en ese sentido podría considerarse una extensión de la lógica multivaluada. La lógica borrosa está relacionada y fundamentada en la teoría de los Conjuntos Difusos. Según esta teoría, el grado de pertenencia de un elemento a un conjunto va a venir determinado por una función de pertenencia, que puede tomar todos los valores reales comprendidos en el intervalo [0,1]. De manera intuitiva se tiene el concepto de conjunto borroso como una colección bien definida de elementos, en la que es posible determinar para un objeto cualquiera, en un universo dado, si acaso éste pertenece o no al conjunto. La decisión, naturalmente, es ”sí pertenece" o bien “no pertenece". De esta forma, los modelos basados en este tipo de lógica asocian a cada entrada o conjunto de entradas una salida en función de una serie de reglas borrosas. Los sistemas que utilizan este tipo de lógica son muy adecuados en campos como la robótica autónoma y la Inteligencia Artificial por las altas tolerancias que soportan y por su flexibilidad. Por otro lado, la principal desventaja que presentan es que las reglas y los conjuntos borrosos se definen generalmente “a mano” para cada problema. Este tipo de lógica es ampliamente utilizada en la actualidad en muy distintos campos científicos y su auge es continuo [Kaufmann, 91], [Kosko, 00].

- **Redes Neuronales Artificiales:** son sistemas para el tratamiento de la información cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano, la neurona. Desde el punto de vista de aplicación matemática, aproximan cualquier función no lineal n-dimensional a base de combinaciones de funciones no lineales ponderadas por una serie de parámetros. Fue en 1943 cuando Warren McCulloch y Walter Pitts propusieron el clásico modelo de neurona en el que se basan las redes neuronales actuales. La redes neuronales artificiales conforman un modelo de procesado distribuido muy potente con características que las hacen muy adecuadas al tipo de problemas con entornos reales que se utilizan en la robótica autónoma: tolerancia a fallos, tolerancia a ruido, posibilidad de utilizar algoritmos de aprendizaje clásicos y posibilidad de combinar evolución con aprendizaje conexionista. Los principales problemas que presentan son la dificultad a la hora de seguir su funcionamiento interno y que deben ser construidas "ad hoc" para cada problema en cuanto a su tamaño y parámetros de aprendizaje. Son numerosos los grupos de investigación que utilizan y han utilizado las redes neuronales como base de sus trabajos, y aún continúan siendo un tema en auge. Una excelente revisión y compilación de los trabajos más relevantes en el campo de las redes neuronales y la computación neuronal en general es la realizada por Fiesler en [Fiesler, 97].

Como primera observación, debemos hacer notar que de estas posibles representaciones, las funciones polinómicas y las reglas son las menos adecuadas para el MDB dada su limitada capacidad de actualización. En este sentido, tanto la lógica borrosa como las redes son más apropiadas para el mecanismo, pero existe otro condicionante muy importante a la hora de utilizar una representación u otra: el algoritmo de búsqueda que utilicemos sobre dicha representación. Por este motivo, volveremos sobre los posibles sustratos para los modelos tras analizar las distintas técnicas para la obtención de los mismos.

Sustrato de estrategias

Las estrategias son una serie de comandos a los actuadores, con lo que pueden ser simples listas de acciones que interpreta el actuador u otra estructura de cómputo más compleja. Quizás el requisito fundamental es que las estrategias estén constituidas de forma que puedan encadenarse, de modo que se facilite la generación de nuevas estrategias a partir de las antiguas.

Las acciones que constituyen las estrategias no tienen por qué ser una codificación directa de los comandos a los actuadores, podemos trabajar con estrategias más simples donde un comando a un actuador para el MDB se convierte en una serie de acciones reales en el agente. Tendríamos así un actuador virtual, que es un concepto análogo al sensor virtual que utilizaremos en el ejemplo del apartado 7.1 y que se analiza con detalle en el Apéndice A.

En resumen, las estrategias se pueden representar de forma general como vectores de datos cuyo dominio depende de los actuadores del agente, como cualquier otra variable de tipo numérico.

Abordaremos a continuación la parte central de este apartado, esto es, el análisis de las técnicas de búsqueda de modelos que serán aplicadas al mecanismo.

Técnicas de búsqueda y optimización

La búsqueda de los modelos en el MDB está condicionada por una serie de limitaciones y requisitos:

- La función a optimizar o maximizar se conoce de forma discreta, a base de muestras de la misma.
- El MDB será aplicado en un agente real y en un entorno real, de modo que las muestras se obtienen en tiempo real, nunca son conocidas a priori.
- Las muestras que se conocen pueden ser particulares de una zona o más generales, y el tener una u otra situación depende del tipo de tarea a realizar y de la exploración que se está realizando del entorno.
- Los entornos reales son dinámicos, de modo que los procesos de búsqueda deben ser capaces de adaptar las funciones que van obteniendo.

Teniendo esto en cuenta, las primeras técnicas que podemos comentar son las basadas en *métodos analíticos*, de modo que si existe la función solución, es de una sola variable, y se puede derivar dos veces en todo su rango, se pueden hallar todos sus máximos, sean locales o globales. Sin embargo, la mayoría de las veces no se conoce la forma de dicha función, y si se conoce, no tiene porqué ser diferenciable ni siquiera una vez. Incluso el tratamiento analítico para funciones de más de 1 variable es complicado. De entre estos métodos, al tener muestras de la función a aproximar, podemos utilizar técnicas de interpolación polinómica o, de cara a reducir el orden del polinomio, de ajuste, como los *splines*. Aunque estas técnicas han sido utilizadas en espacios n-dimensionales como el que nos ocupa (ya que los modelos pueden tener un número ilimitado de entradas y salidas), sus resultados para espacios grandes son pobres y la mayor parte de los trabajos se concentran en 2 y 3 dimensiones como los de [Foley, 86], [Alfeld, 89] o [Mann, 99]. El principal problema de este tipo de técnicas reside en la explosión combinatoria que se produce al aumentar el número de muestras y la dimensión del espacio, además de sus limitaciones adaptativas.

Por otro lado, existen métodos *exhaustivos* que recorren todo el espacio de búsqueda quedándose con la mejor solución y métodos *aleatorios* que van muestreando el espacio de búsqueda acotando las zonas que no han sido exploradas, escogen la mejor solución, y, además, proporcionan el intervalo de confianza de la solución encontrada. Tanto unos como los otros adolecen de nuevo de problemas de explosión combinatoria en funciones de varias variables.

Otro tipo de algoritmos son los basados en *descenso* y *ascenso de gradiente*, donde se va evaluando la función en uno o varios puntos, pasando de un punto a otro en el cual el valor de la evaluación es superior. La búsqueda termina cuando se ha encontrado el punto con un valor máximo. Estos algoritmos toman muchas formas diferentes, según el número de dimensiones del problema solución, el valor del incremento y la dirección en la cual se tiene que dar. Por ejemplo, en algunos casos se utiliza el llamado Método Montecarlo, en el cual se escoge la nueva solución de forma aleatoria. Debemos mencionar, en particular, el algoritmo de descenso de gradiente aplicado a redes neuronales artificiales, denominado algoritmo de retropropagación, y que ha sido extensamente utilizado. Dicho algoritmo fue presentado en 1949 por Donald Hebb [Hebb, 49] y en 1957, Frank Rosenblatt presentó el Perceptron [Rosenblatt, 57], una red neuronal ampliamente utilizada cuya regla de aprendizaje era una modificación de la propuesta por Hebb.

El principal problema de este tipo de algoritmos de descenso y ascenso de gradiente, es que se quedan en el pico más cercano a la solución inicial y no son válidos para problemas multimodales, en los cuales la función de coste tiene varios óptimos posibles [Sutton, 86]. Además, son técnicas que requieren un conocimiento previo del espacio de búsqueda y presentan grandes problemas al ser aplicados a entornos reales, ya que no es posible determinar a priori el grado de conocimiento del espacio que se va a tener [Yao, 97]. En este tipo de problemas, por ejemplo, con un algoritmo de retropropagación se obtendrían soluciones muy particulares. Estos métodos requieren un conocimiento previo para poder "arrancar", y en un caso real esto no es siempre posible. Y por último, la adaptabilidad de las soluciones obtenidas en entornos dinámicos es muy costosa, ya que depende de cuán parecida sea la nueva muestra a las existentes.

En los últimos años, las técnicas de búsqueda y optimización matemática más utilizadas en problemas de Inteligencia Artificial se han basado en la introducción de *heurísticas*. Las heurísticas son criterios, métodos o principios para decidir cuál de varias alternativas de acción promete ser la más efectiva para cumplir con una meta y representan un compromiso entre simplicidad y poder discriminatorio. El auge que experimentan los procedimientos heurísticos [Laguna, 03] se debe sin duda a la necesidad de disponer de herramientas que permitan ofrecer soluciones rápidas a problemas reales. Es importante destacar el hecho de que los algoritmos heurísticos, por sí solos, no garantizan que la solución encontrada sea óptima, aunque su propósito es encontrar una solución cercana al óptimo en un tiempo razonable.

Dentro de las técnicas heurísticas podemos encontrar diversos métodos tales como: métodos constructivos, de descomposición, de reducción, de manipulación del modelo y de búsqueda local. Tradicionalmente, para resolver un problema dado se diseñaba un algoritmo específico que pertenecía a algunos de los métodos enumerados. Hoy día, el interés primordial de los investigadores del área es el de diseñar métodos generales que sirvan para resolver clases o categorías de problemas. Dado que estos métodos generales sirven para construir o guiar el diseño de métodos que resuelvan problemas específicos se les ha dado el nombre de *metaheurísticos*.

Los cuatro procedimientos metaheurísticos más utilizados y reconocidos son: Recocido Simulado, Búsqueda Tabú, GRASP y Algoritmos Evolutivos. En [Adenso, 96] se lleva a cabo un análisis en profundidad de estas técnicas, de las que a continuación realizamos una breve reseña:

- **Recocido Simulado (simulated annealing):** esta técnica se basa en una analogía con el comportamiento de un sistema físico al someterlo a un baño de agua caliente. El Simulated Annealing (SA) ha sido probado con éxito en numerosos problemas de optimización, mostrando gran capacidad para evitar quedar atrapado en óptimos locales. Fue propuesto originalmente por Kirpatrick, Gelatt y Vecchi en 1983 [Kirpatrick, 83], y es un procedimiento basado en búsqueda local en donde todo movimiento de mejora es aceptado y se permiten movimientos de no mejora de acuerdo con unas probabilidades. Dichas probabilidades están basadas en la analogía con el proceso físico de enfriamiento y se obtienen como función de la temperatura del sistema. De cara a profundizar en esta técnica destacamos el trabajo de Johnson y Aragon [Johnson, 89], [Aragon, 91] donde se analizan en profundidad las distintas aproximaciones a la utilización del SA.
- **Búsqueda Tabú:** es una estrategia para resolver problemas de optimización combinatoria y se suele plantear como la minimización de una función objetivo *c(x)*

sujeta a ciertas restricciones [Glover, 93]. En esencia es un metaheurístico que puede ser utilizado para guiar cualquier procedimiento de búsqueda local en la búsqueda agresiva (estrategia que trata de evitar que la búsqueda quede "atrapada" en una solución local) del óptimo del problema. A tal efecto, la búsqueda tabú utiliza el concepto de memoria y lo implementa mediante estructuras simples con el objetivo de dirigir la búsqueda teniendo en cuenta la historia de ésta. Es decir, el procedimiento trata de extraer información de lo sucedido y actuar en consecuencia. En este sentido puede decirse que hay un cierto aprendizaje y que la búsqueda es inteligente.

- **Algoritmos evolutivos:** los algoritmos evolutivos establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en un vector a modo de cromosoma. Fueron introducidos por Holland [Holland, 70] y su principal diferencia a nivel matemático con el resto de técnicas reside en que en vez de tener una sola solución que se va alterando hasta obtener el óptimo, se persigue el óptimo cambiando varias soluciones de forma conjunta facilitando el proceso de escapar de los mínimos locales. En un algoritmo evolutivo, tras parametrizar el problema en una serie de variables, éstas se codifican en un cromosoma. Todos los operadores utilizados se aplicarán sobre estos cromosomas, o sobre poblaciones de ellos. Las soluciones codificadas en un cromosoma compiten para ver cuál constituye la mejor solución (aunque no necesariamente la mejor de todas las soluciones posibles). El ambiente, constituido por el resto de soluciones, ejercerá una presión selectiva sobre la población, de forma que sólo los mejor adaptados (aquellos que resuelvan mejor el problema) sobrevivan o leguen su material genético a las siguientes generaciones, igual que en la evolución natural de las especies. La diversidad genética se introduce mediante mutaciones y reproducción sexual. En el siguiente apartado volveremos a analizar con más detalle los algoritmos evolutivos.
- **GRASP:** los métodos GRASP fueron desarrollados al final de la década de los 80 con el objetivo inicial de resolver problemas de cubrimientos de conjuntos [Feo, 89]. El término GRASP (Greedy Randomized Adaptive Search Procedures) fue introducido por Feo y Resende [Resende, 95] como una nueva técnica metaheurística de propósito general, y es un procedimiento iterativo en donde cada paso consiste en una fase de construcción y una de mejora. En la fase de construcción se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial. Esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local. La mejor de todas las soluciones examinadas se guarda como resultado final. Este tipo de algoritmo es el más reciente de todos los mencionados, pero sus perspectivas en cuanto a fiabilidad y rapidez hacen que cada vez sea más utilizado por los investigadores [Martí, 01].

Cualquiera de estas técnicas metaheurísticas son aplicables al mecanismo a la hora de encontrar los modelos porque cumplen los requisitos introducidos al comienzo de este apartado. Como ya hemos dicho al analizar el sustrato de los modelos, la elección de una técnica u otra depende del tipo de representación utilizada para dichos modelos. A continuación pasamos a estudiar los procesos de búsqueda desde el punto de vista del aprendizaje, algo que nos ayudará a la hora de decidirnos por una u otra de estas técnicas de búsqueda.

Aprendizaje

Los procesos de búsqueda de modelos se llevan a cabo a partir de muestras o puntos conocidos de las funciones reales que dichos modelos deben aproximar. Estas muestras se obtienen de la interacción del agente con el entorno en el que habita tras la realización de una acción o secuencia de acciones. En el caso de un mecanismo congnitivo, esta búsqueda no sigue el patrón común de los procesos matemáticos de optimización, ya que las muestras que definen el espacio de soluciones son conocidas en tiempo de ejecución, es decir, no se pueden utilizar a priori. Esto implica que los modelos no pueden ser obtenidos mediante un procedimiento instantáneo, sino que se requiere un proceso de *aprendizaje* temporal. Así, los modelos son perfeccionados y actualizados como resultado de la interacción entre el agente y el mundo y de la observación por el agente de sus propios procesos de toma de decisiones.

Como se establece en [Russell, 96], los procesos de aprendizaje se componen conceptualmente de un elemento de desempeño, que es el responsable de escoger las acciones y de un elemento de aprendizaje, que modifica el elemento de desempeño. El elemento de aprendizaje ha sido representado de muy diversas maneras en el campo de la Inteligencia Artificial: descripciones deterministas como los polinomios, oraciones propositivas y oraciones lógicas de primer orden, descripciones probabilísticas como las redes de creencia, etc. Los algoritmos de aprendizaje en cada caso son diferentes, pero la idea de fondo es la misma en todos los casos y se basa en el *aprendizaje por inducción*, es decir, a base de pares de ejemplo entrada/salida, también denominado aprendizaje a partir de la observación.

Como hemos venido explicando, los modelos en un Mecanismo Cognitivo relacionan información sensorial, tanto interna como externa, y acciones en función del tiempo. La pregunta que debemos plantearnos es: ¿cómo pueden los modelos llegar a formarse y modificarse? o, dicho en los términos anteriores, ¿cómo se llegan a aprender estos modelos?. La respuesta más simple consiste en utilizar una señal de error que es retroalimentada. Si las salidas correctas son conocidas por el agente, este tipo de aprendizaje se denomina *supervisado* y requiere de un maestro que informa sobre las salidas. Por otro lado, si no existe ninguna indicación sobre cuáles son las salidas correctas, se le conoce como aprendizaje *no supervisado* y presenta la gran ventaja de que sus patrones de aprendizaje se toman directamente de la secuencia temporal de datos que le son presentados a los sensores sin necesidad alguna de un entrenador externo.

Entre estos dos tipos de aprendizaje, podemos situar el aprendizaje *por refuerzo* [Sutton, 98], que se puede considerar un aprendizaje supervisado donde la señal de retroalimentación no es la salida directamente, sino una señal informativa de lo que se debe hacer para mejorar las soluciones. Esta técnica se basa en el uso de una señal de refuerzo que debe ser maximizada, de modo que no se proporcionan al aprendiz las acciones que debe tomar, sino que las va escogiendo por prueba y error. Estas acciones afectan a la señal de refuerzo actual y a las futuras. Existe un compromiso entre el uso de las acciones que ha aprendido y han funcionado y la exploración de nuevas acciones para maximizar el refuerzo. Los principales elementos que se utilizan en el aprendizaje por refuerzo son:

- **Política** (π): define cómo se comporta el sistema en un instante dado. Una política relaciona, a veces estocásticamente, los estados con las acciones.

- **Función de recompensa** (R): define la meta. Relaciona cada estado-acción a un número (recompensa), indicando lo deseable del estado.
- **Función de valor** (V): indica lo que es bueno a largo plazo. Es la recompensa total que un agente puede esperar acumular empezando en ese estado (predicciones de recompensas). Se buscan acciones que den los valores más altos, no la recompensa mayor. Las recompensas están dadas por el entorno, pero los valores se deben de estimar en base a las observaciones.
- **Modelo del ambiente** (opcional): imita el comportamiento del entorno. Se puede usar para hacer planificación al considerar posibles situaciones futuras basadas en el modelo.

El aprendizaje por refuerzo es una de las áreas más activas en la investigación del aprendizaje aplicado a agentes artificiales como lo demuestran trabajos recientes como los de [Rennie, 99] aplicado a búsquedas en internet, [Smart, 2000] a tareas de control dinámico, [McGovern, 01] que acelera el proceso de aprendizaje a base de fijar subobjetivos, etc.

Para el MDB, en el proceso de aprendizaje de los modelos y en una primera aproximación, usaremos directamente la diferencia entre lo que predicen dichos modelos en un cierto instante y lo que se percibe realmente en ese instante, respecto tanto al mundo exterior como al interior. Esta señal de error representa la función de recompensa y establece una indicación de lo bueno o malo que es el modelo que proporciona la salida. En este sentido, estamos utilizando un aprendizaje supervisado por refuerzo ya que indicamos cuál es la función de error deseada aunque no en el sentido tradicional (no utilizamos política, función de recompensa, etc de forma explícita). Volveremos a incidir sobre este punto en siguientes apartados.

Además de incluir capacidades de aprendizaje en los modelos, las acciones han de poderse probar en dichos modelos, y estos, a su vez, han de ser capaces de darnos una predicción de los efectos de dichas acciones sobre el entorno, el agente y sus motivaciones. Debemos utilizar, por tanto, una técnica de aprendizaje que cumpla todos estos requisitos y, además, incluya las conclusiones que se extraen de las teorías biológicas introducidas en el apartado 4.4. Esta tarea se lleva a cabo a continuación.

Darwinismo en el MDB

La forma en que hemos incluido las teorías darwinistas del apartado 4.4 en un modelo computacional para el MDB ha sido a través de *técnicas evolutivas*. Como ha sido analizado en el apartado anterior, estas técnicas están basadas en la generación de una serie de representaciones o codificaciones, en principio aleatorias, de las soluciones a un problema. Dichas soluciones se evalúan por medio de una función de calidad que las ordena atendiendo a su capacidad para resolver con acierto dicho problema. Del mismo modo que en un proceso selectivo natural, las mejores soluciones permanecen y variaciones suyas dan lugar a otras, mientras que las peores desaparecen.

Además, las técnicas evolutivas son procesos basados en poblaciones de soluciones donde no hay una única solución globalmente buena como en el caso de un algoritmo de optimización puro, sino que la población almacena mucha información útil de modo que, por ejemplo, tal y como expone Xin Yao en [Yao, 97], la combinación de distintos individuos en un proceso de coevolución puede dar lugar a soluciones aún mejores. Este es el concepto de aprendizaje que pretendemos que posea el MDB, ya que la adaptación

a los cambios en entornos reales será más eficiente si existe una población de posibles soluciones que si existe una única.

Pasamos, por tanto, a estudiar con detalle este tipo de técnicas previo a explicar en profundidad su inclusión en el MDB.

Técnicas evolutivas

Estas técnicas y algoritmos se basan en una analogía entre el proceso evolutivo en la naturaleza y la búsqueda de una solución en un espacio determinado. Su funcionamiento general es el siguiente:

1. Partimos de una población inicial de individuos donde cada uno de ellos representa una posible solución al problema, un punto en el espacio de soluciones, y viene caracterizado por un cromosoma, es decir, por un modo particular de codificar dicha solución.
2. Esta población evoluciona con el tiempo de modo que los individuos se reproducen y sufren mutaciones en los genes de sus cromosomas introduciendo así nuevas soluciones posibles. Las posibilidades que tienen de pasar los individuos originales o sus descendientes a la siguiente generación son directamente proporcionales a lo cerca que estén de la solución del problema.

La forma en la que los individuos se reproducen, cómo tienen lugar las mutaciones, la frecuencia de ambas formas de reproducción y sobre qué individuos se producen, dan lugar a los diversos tipos de algoritmos evolutivos. Todos ellos se han englobado en una disciplina denominada computación evolutiva, de la que destacamos las revisiones realizadas en [Fogel, 98] y [Bäck, 97].

La función que evalúa a los individuos de la población, es decir, a cada una de las soluciones se denomina función de calidad. Así, a cada individuo se le calcula su valor de calidad, que es el que determina siempre los individuos que se van a reproducir y aquellos que se van a eliminar. Los valores de calidad establecen la métrica según la cual se van a comparar los individuos. El hecho de que podamos utilizar cualquier tipo de criterio para otorgar calidad a un individuo, hace que los algoritmos evolutivos sean muy potentes en aquellos casos donde no se conocen los detalles del problema a tratar con exactitud, pero sí se conoce el efecto o consecuencia buscado. El cálculo de la calidad puede incluir cualquier además más de un objetivo, como veremos más adelante al hablar de problemas multiobjetivo.

Bajo la denominación de algoritmos evolutivos se engloban distintas variantes como los algoritmos genéticos, las estrategias de evolución, la programación evolutiva y los algoritmos macroevolutivos. A continuación haremos una breve revisión de cada una de estas técnicas por separado.

Algoritmos genéticos

Originalmente, en los algoritmos genéticos propuestos por Holland [Holland, 70], los cromosomas estaban codificados en binario. Actualmente existen otras posibles codificaciones, por ejemplo, en el MDB utilizaremos mayormente una codificación donde cada gen es un número real. Los genes de los individuos de la población inicial suelen ser tomados aleatoriamente, aunque también es posible intentar distribuirlos de forma que queden equidistantes en el espacio de soluciones. Una vez creada esta

población inicial, el proceso evolutivo se puede implementar en un bucle con los siguientes 5 pasos:

1. Se calcula la calidad de los individuos de la población inicial en función de lo satisfactorios que resulten en el problema de optimización.
2. Se seleccionan los individuos a reproducirse siguiendo algún criterio relacionado con la calidad.
3. Se obtiene una población de descendientes mediante el cruce y mutación.
4. Se evalúan los individuos de la nueva población mediante el cálculo de la función de calidad.
5. Se seleccionan los individuos de la población de los padres a ser reemplazados y se funden ambas poblaciones.

El método aplicado originalmente para la selección de individuos ha sido la *selección proporcional*, donde cada individuo tiene una probabilidad de ser seleccionado para reproducirse igual a su calidad dividido por la suma de las calidades de todos los individuos. Existen otros dos tipos de selección que arreglan los problemas de presión selectiva de esta primera técnica. Una es la *selección por posición*, en la que los individuos se ordenan por calidad y la probabilidad de selección es función de su posición en esa lista ordenada. El otro tipo de selección es la *selección por torneo*, donde se escoge aleatoriamente un grupo de *n* individuos (donde *n* es el tamaño de la ventana de selección) y dentro de este grupo se selecciona el que tiene mayor calidad para la reproducción.

El *cruce* es uno de los dos operadores de reproducción básicos en los algoritmos genéticos y su aplicación consiste en seleccionar dos individuos, un punto de cruce, y crear dos individuos nuevos de tal forma que cada uno tenga los primeros *n* genes de un padre y los siguientes *m-n* genes del otro (siendo *m* la longitud total del cromosoma en genes). Existen otras variantes de cruce, usando por ejemplo dos o tres puntos de cruce en lugar de uno. La idea del operador cruce se basa en que, dados dos individuos con buena calidad, el individuo resultante se quede con los genes que provocaban una buena calidad en ambos padres, dando lugar a un individuo todavía mejor. Esto evidentemente no se produce siempre, sino con una cierta probabilidad, pero cuanto mayor es la calidad de un individuo más veces se reproduce y, por tanto, mayor es la probabilidad de que la parte de su genoma causante de su buena calidad se mezcle con la parte correspondiente de otro individuo.

El otro operador básico a la hora de generar diversidad en la población de soluciones es la *mutación*, que consiste en cambiar el valor de un gen por otro aleatorio respetando la codificación original. Por ejemplo, si los genes son números reales, es frecuente que la mutación consista en sumar o restar una cantidad al valor del gen. La mutación en los algoritmos genéticos es un medio de introducir variedad en la población y por eso la probabilidad de mutación suele ser baja.

Los algoritmos genéticos tienen una serie de parámetros que se tienen que fijar para cada ejecución, como por ejemplo:

- *Tamaño de la población:* debe de ser suficiente para garantizar la diversidad de las soluciones, y, además, tiene que crecer más o menos con el tamaño del cromosoma, aunque no existe una regla fija en este sentido.

- *Criterio de parada:* lo más habitual es que el criterio de parada sea la convergencia del algoritmo genético o un número prefijado de generaciones.

Un texto fundamental tanto desde el punto de vista introductorio como a la hora de profundizar en los distintos aspectos de los algoritmos genéticos es el escrito por David Goldberg [Goldberg, 89].

Estrategias evolutivas

Las estrategias evolutivas se diferencian principalmente de los algoritmos genéticos en que el operador que guía el proceso evolutivo es la mutación y no el cruce.

Existen, básicamente, dos tipos de estrategias de evolución:

- Las *(m+n)ES* donde *m* individuos generan *n* hijos y se seleccionan para la siguiente generación los *m* mejores individuos del total *m+n*.
- Las *(m,n)ES* donde *m* individuos generan *n* hijos y se seleccionan para la siguiente generación los *m* mejores individuos de los *n* hijos.

En ambos casos, los *n* hijos se generan mutando todos los genes de los cromosomas, donde dicha mutación consiste en sumar un valor a cada gen. El total de los valores sumados a los genes suele seguir una distribución de media cero. El valor máximo a sumar a los genes viene fijado por una regla o puede incluirse dentro del cromosoma como un gen más. Igualmente, dicho valor puede ser el mismo para todos los genes del cromosoma o puede haber varios valores para distintos grupos de genes.

Han surgido variantes de las estrategias de evolución que aplican el operador de cruce entre dos individuos antes de aplicar el operador mutación, con lo que la diferencia entre los algoritmos genéticos y las estrategias de evolución disminuye.

Por último, con objeto de relacionar dos de las metaheurísticas presentadas en el apartado dedicado a las técnicas de búsqueda de modelos y acciones, diremos que las estrategias evolutivas puras que no utilizan operador de cruce son equivalentes a múltiples algoritmos (una población de ellos) de recocido simulado en paralelo.

Programación evolutiva

La programación evolutiva fue planteada desde un principio como una alternativa a la Inteligencia Artificial clásica con el objetivo de obtener comportamientos inteligentes de forma más automática.

Es similar a las estrategias de evolución, pero se centra más en que lo realmente importante es el comportamiento y no la codificación del individuo en el cromosoma. Cada individuo posee un cromosoma que constituye un programa en un lenguaje de alto nivel que puede ser desde un controlador hasta un programa que describa la construcción de una red neuronal. La mutación es más compleja que en los casos anteriores, y consiste en cambiar una sentencia o grupo de sentencias de ese lenguaje de alto nivel por otra sentencia o grupo de sentencias, comprobando que no se generan individuos imposibles, incorrectos sintáctica o semánticamente.

Existe una variante denominada programación genética en la que los cromosomas son programas en un lenguaje de más bajo nivel y donde también existe el operador cruce.

El mayor inconveniente de la programación evolutiva radica en que en cada problema se deben fijar las operaciones permitidas sobre los cromosomas y establecer una codificación que sea flexible y, al mismo tiempo, permita obtener resultados en un tiempo razonable. Como consecuencia, el lenguaje utilizado suele ser demasiado específico y no permite su generalización a otros problemas diferentes.

Un autor destacado en este campo es John Koza, que realiza una extensa revisión del campo en [Koza, 92], [Koza, 94], [Koza, 99], [Koza, 03].

Algoritmos macroevolutivos

Este tipo de algoritmo es el de más reciente aparición y fue desarrollado por Jesus Marin y Ricard Solé [Solé, 97] y formalizado en 1999 en el artículo [Marín, 99].

La diferencia principal con el resto de algoritmos evolutivos es que en éstos la evolución tiene lugar a nivel de población mientras que en los macroevolutivos (MA) la evolución se produce a más alto nivel. Como en los algoritmos genéticos, los MA usan una población constante de individuos que evolucionan en el tiempo mediante actualizaciones sucesivas de los operadores. Ahora los individuos se denominan *especies* y los operadores básicos que se utilizan son *selección* y *colonización*. La idea fundamental es que el sistema escogerá qué individuos deben ser eliminados (selección) así como garantizará la exploración del espacio de soluciones introduciendo individuos nuevos (colonización).

Se define el estado de la especie i en la generación t como:

- 1 si el estado es “vivo”
- 0 si el estado es “extinguido”

La relación entre especies se representa mediante una matriz de conexión *W* donde cada elemento mide la influencia de una especie sobre la otra con un valor continuo en el intervalo (-1, 1). Estos valores W se calculan mediante una función que depende de la calidad de cada especie respecto a la del resto de la población.

Cada generación consiste de los siguientes 3 pasos fundamentales:

1. *Cálculo de la calidad de cada especie:* el concepto de función de calidad y su cálculo es análogo al resto de técnicas evolutivas.
2. *Extinción:* la relación entre cada especie y el resto de la población determina su coeficiente de supervivencia *h* que se define como:

   $$h_i(t) = \sum W_{ij}(t)$$

   donde t es el número de generación. El estado de cada especie se actualiza usando:

   $$S_i(t+1) = 1 \text{ (vivo) si } h_i(t) \geq 0$$

   $$S_i(t+1) = 0 \text{ (extinguido) en otro caso}$$

   en la primera generación, se escoge aleatoriamente una conexión $W_{ij}(t)$ para cada especie.

   En este paso se aplica, por tanto, el operador de selección que marca las especies que van a ser eliminadas.

3. *Diversificación:* de las especies supervivientes se escoge aleatoriamente las que reemplazan a las que se extinguen. Pero este reemplazo no es tal porque la especie nueva se basa en la que se extingue modificada para parecerse a una de las que sobrevive (es decir, la especie superviviente *A* "atrae" a la especie extinguida *B*). De esta forma, estaremos aplicando el operador colonización.

Cualquiera de estas 4 técnicas evolutivas pueden ser aplicadas en la búsqueda del modelo interno I, del modelo de mundo *W* e incluso de la acción *A*. Tal vez, la programación evolutiva sea la técnica que menos se adapta al problema siendo más adecuada a la hora de generar programas de control y no una simple búsqueda de funciones. De las otras tres, hemos utilizado inicialmente los algoritmos genéticos y los macroevolutivos con resultados muy similares y satisfactorios aplicándolos a la búsqueda de modelos y, de cara a mantener la misma filosofía, también de las acciones. Pero la realidad es que, a la hora de aplicar directamente estos algoritmos en problemas complejos con el MDB, encontramos ciertas limitaciones que implicaron el desarrollo de un algoritmo propio, denominado Algoritmo Genético basado en Genes Promotores (PBGA), y que analizaremos con detalle más adelante.

A modo de resumen, enumeramos a continuación los principales motivos que hacen que los algoritmos evolutivos sean especialmente adecuados a la hora de llevar a cabo el aprendizaje de los modelos en el MDB:

1. Presentan un alto grado de similitud con los procesos selectivos naturales que son la base de las teorías del apartado 4.4, donde las mejores soluciones permanecen y variaciones suyas dan lugar a nuevas soluciones, mientras que las peores desaparecen.
2. Se han mostrado como técnicas muy potentes a la hora de realizar búsquedas en espacios n-dimensionales al escapar con facilidad de los mínimos locales.
3. Permiten un aprendizaje a partir de muestras de modo similar al aprendizaje por refuerzo, en el sentido de que se incluye la función de recompensa a través de la función de calidad aunque no se utilizan conceptos como función de valor, políticas, etc.
4. El aprendizaje se produce en una población de individuos, de modo que el algoritmo lleva un conjunto de soluciones posibles. Cada vez que aparece una nueva muestra, el algoritmo actualiza y "refina" la solución, pero al tener un conjunto de soluciones posibles, este proceso se acelera ya que se aumenta la información disponible. Por tanto, los algoritmos evolutivos nos proporcionan una alta capacidad adaptativa en los modelos, lo que los hace indicados para tratar con problemas dinámicos.
5. El número de generaciones de evolución nos permite controlar directamente el grado de aprendizaje, de modo que si deseamos un aprendizaje suave para evitar que los modelos se queden en soluciones particulares, únicamente debemos utilizar un número de generaciones bajo para cada iteración del mecanismo.

Una vez fijado el algoritmo de búsqueda, debemos establecer sobre qué sustrato se realiza dicha búsqueda, es decir, qué representación usaremos para los modelos. Para ello, la representación escogida debe ser fácilmente evolucionable, es decir, su interrelación con los algoritmos evolutivos debe ser adecuada.

Como primera aproximación al MDB, hemos utilizado una *red neuronal artificial* (en principio de tipo perceptrón multicapa, pero no se ha impuesto ningún tipo de

restricción en este sentido). Tal y como se presentó en el apartado dedicado al sustrato de los modelos, no es más que un aproximador universal de funciones no lineales con parámetros ajustables y capaz de proporcionar una función continua a partir de muestras. En principio no existe ninguna razón para no poder utilizar cualquier otro tipo de estructura como las mencionadas en dicho apartado, pero las redes poseen una capacidad adaptativa superior al resto de representaciones que es fundamental de cara a la aplicación real del MDB. Además, son estructuras que requieren bajo nivel de conocimiento del problema por parte del diseñador (únicamente número de entradas y salidas), evitando así las etapas previas que se requieren a la hora de fijar, por ejemplo, las reglas borrosas.

Las teorías de conocimiento y aprendizaje que hemos considerado como base para el MDB, hablan de selección de circuitos neuronales, tanto en la formación del cerebro como en sus primeros aprendizajes y en la selección de salidas de circuitos en el funcionamiento adulto. Siguiendo esta analogía, los cromosomas que codifican los modelos en las técnicas evolutivas bien pueden corresponder a redes neuronales artificiales, sobre todo por su capacidad de aprendizaje local ya que se refuerzan aquellos circuitos neuronales que resultan buenos a base de modificar los pesos. Por tanto, de cara a dotar de capacidad de aprendizaje al MDB, hemos utilizado técnicas evolutivas aplicadas sobre redes neuronales artificiales que representan los modelos. Este tipo de técnicas se engloban dentro del campo del Aprendizaje Evolutivo, que ha tenido un gran auge desde principios de los años 90 y que pasamos a revisar brevemente.

Aprendizaje evolutivo

El algoritmo de retropropagación clásico para redes neuronales perceptrón presenta, como hemos dicho en el apartado dedicado a las técnicas de búsqueda de modelos y acciones, una serie de problemas a la hora de ser aplicado a entornos dinámicos, o lo que es lo mismo, a problemas reales. Además, es un algoritmo que proporciona una única solución, una única red, de modo que resuelve un problema de optimización, pero no de aprendizaje tal y como expone Xin Yao en [Yao, 97]. En este sentido, los investigadores que trabajan con procesos que requieren aprendizaje, han tratado de obtener las redes neuronales mediante algoritmos evolutivos. Como hemos visto, este tipo de algoritmos están basados en poblaciones de soluciones, de modo que no hay una única solución globalmente buena como en el caso de un algoritmo de optimización puro, sino que la coexisten varias que pueden resultar más adecuadas en un instante dado y menos en otro. Aquí reside la propiedad más interesante de los algoritmos basados en poblaciones, y aquí reside el concepto de aprendizaje.

En este campo, desde principios de los años 90 han aparecido trabajos [Weiss, 90] tratando de sentar las bases de este tipo de aprendizaje, denominado también *neuroevolución*. La mayor parte de los investigadores se han centrado en encontrar distintas técnicas para modificar los pesos que modulan las conexiones sinápticas entre las neuronas y para modificar la propia arquitectura de la red.

En cuanto a los pesos, inicialmente, éstos se representaban mediante cadenas en binario [Caudell, 89], de acuerdo con las primeras versiones de los algoritmos genéticos. Este tipo de codificación tiene una problema fundamental, y es que, si el número de bits utilizado para representar cada peso es demasiado pequeño, la evolución se ralentiza y es posible que no se pueda alcanzar la solución buscada por falta de

precisión. Por otro lado, si el número de bits es demasiado grande, el tiempo de cómputo se dispara.

Un modo más natural de representar los pesos sinápticos es utilizando directamente números reales por peso [Fogel, 90]. Los operadores genéticos de cruce y mutación originales [Goldberg, 89] no son aplicables a números reales, por lo que se ha hecho necesario el desarrollo de operadores capaces de introducir variedad en las poblaciones formadas por cromosomas de números reales. En este sentido, destaca el artículo de Montana y Davis [Montana, 89], donde se define un gran número de operadores tanto de cruce como de mutación para números reales incluyendo algunas heurísticas sobre el entrenamiento de redes neuronales.

Siguiendo con las modificaciones a los pesos sinápticos, la eficiencia del aprendizaje evolutivo puede ser mejorada si se combina con una técnica de búsqueda local, en lo que se denomina *entrenamiento evolutivo híbrido*. En este tipo de técnicas, se utiliza un algoritmo evolutivo tradicional y a la hora de aplicar el operador de mutación, los pesos son retocados mediante un algoritmo local como el de retropropagación [Kitano, 90], u otro más potente como recocido simulado [Yao, 91].

Por otro lado, cada individuo del proceso evolutivo representa, normalmente, una red, es decir, una solución independiente. Por tanto, en cada cromosoma se codifican, además de los pesos, los distintos parámetros de la red que permitan aplicar dichos pesos a una cierta topología. Es decir, se hace necesaria la definición de la arquitectura de la red, ya que son múltiples los distintos tipos de redes existentes.

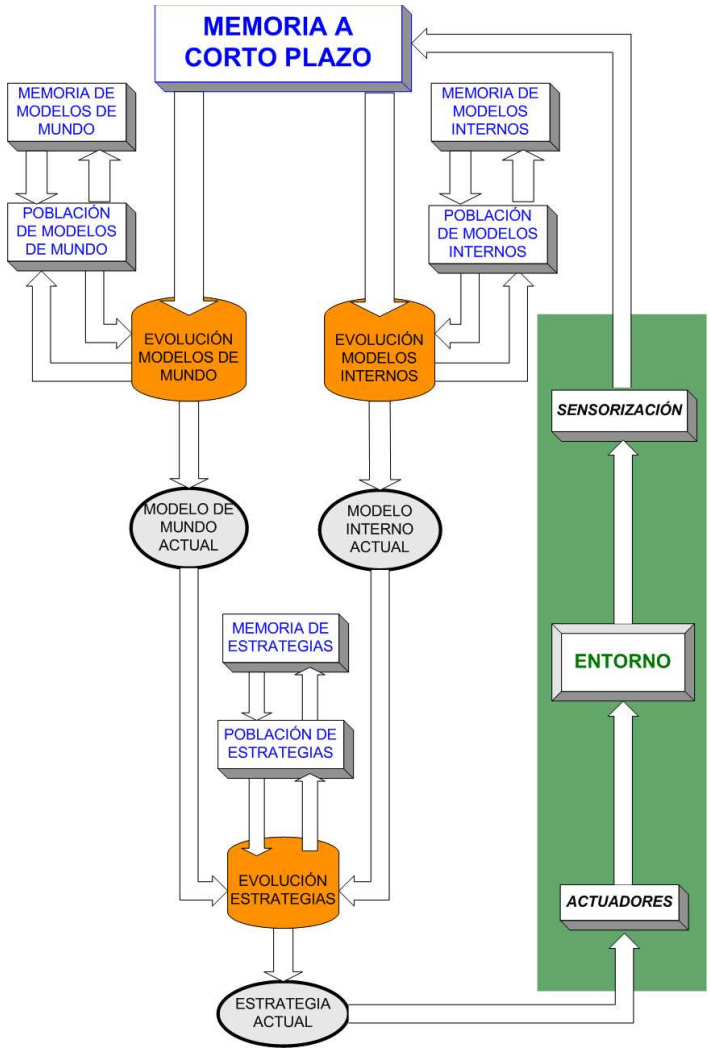
A la hora de codificar las arquitecturas, los investigadores han optado por dos aproximaciones diferentes: una codificación directa y otra indirecta. En la directa, cada conexión dentro de la red viene especificada totalmente por su representación binaria [Wilson, 90], de modo que son necesarias unas reglas simples para guiar la decodificación a la hora de ejecutar dicha red. En la codificación indirecta, los cromosomas incluyen únicamente los parámetros más importantes para la decodificación de acuerdo a un patrón. Estas aproximaciones son más compactas y más próximas a los procesos naturales, ya que requieren una transformación genotipo-fenotipo al igual que, por ejemplo, en los cromosomas biológicos reales. Así, únicamente son necesarios un número reducido de parámetros y un patrón que indique cómo se combinan los distintos pesos en la estructura final de la red.

Una excelente recopilación y resumen de los distintos algoritmos desarrollados en el campo del aprendizaje evolutivo, es la realizada por Xin Yao en [Yao, 93].

Volveremos a incidir sobre las características de estos algoritmos al presentar el PBGA, un algoritmo genético que evoluciona redes neuronales y que hemos desarrollado para ser aplicado en problemas reales. Pasamos ahora a analizar las implicaciones de la elección de un algoritmo con aprendizaje evolutivo sobre el funcionamiento del MDB.

Aplicación al MDB

El diagrama de bloques planteado para el MDB en el apartado 6.1, se puede modificar de modo que incluya los algoritmos evolutivos como procesos de búsqueda concretos para los modelos y las acciones (estrategias en general). Así, planteamos un nuevo diagrama de bloques que esquematiza el funcionamiento del MDB, y el es siguiente:



En la operación del MDB, los modelos de mundo se evalúan a partir de lo buenos que han sido al predecir las percepciones sensoriales tras la aplicación de una estrategia. Como acabamos de comentar, esto se puede llevar a cabo de forma simple estableciendo algún tipo de función de error entre las percepciones previstas y las obtenidas para un modelo dado, que podría ser algo tan simple como la diferencia entre ellas. Así, la propia interacción con el entorno irá evaluando el conjunto de modelos posibles existentes. Es importante observar que con este mecanismo hay un procesamiento paralelo masivo de muchas combinaciones de modelos y un proceso de selección que lleva al modelo actual, o sea, al modelo del cual el agente es consciente y que utilizará para la evaluación de las estrategias. Del mismo modo tiene lugar la evolución de los modelos internos utilizando una función de calidad que tiene en cuenta la diferencia entre el estado interno real y el que obtiene el modelo.

En el caso de las estrategias el mecanismo es similar, sólo que ahora se evalúan aplicándolas en los mejores modelos tras la evolución y el criterio de calidad de cada estrategia es el grado de satisfacción predicho de las motivaciones. La estrategia escogida se aplica al mundo real a través de los actuadores y esto proporciona una nueva relación entre la sensorización y el estado interno en t y t+n, es decir, un nuevo par acción-percepción que servirá para perfeccionar los modelos en una nueva iteración del mecanismo. Este ciclo básico se repite indefinidamente y establece la forma de realizar un procesado paralelo que lleva a soluciones secuenciales, con muy pocas restricciones respecto al tipo de soluciones que pueden resultar, y proporcionando la oportunidad de soluciones nuevas no programadas y de capacidades de predicción y planificación.

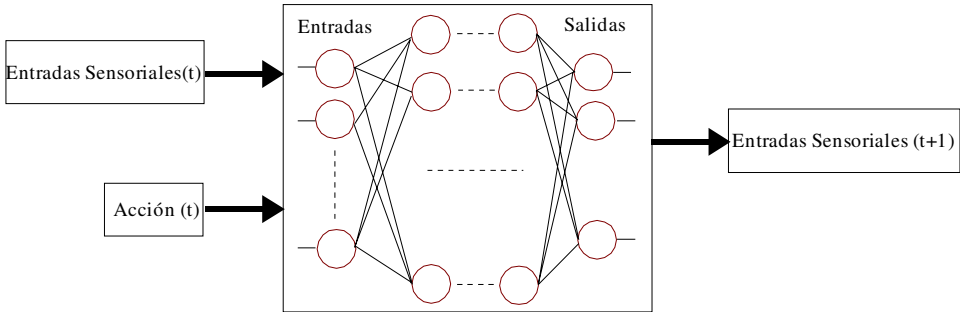
Pasamos a comentar con más detalle las principales características de la inclusión de estos nuevos bloques asociados a los algoritmos evolutivos.

Modelos de mundo

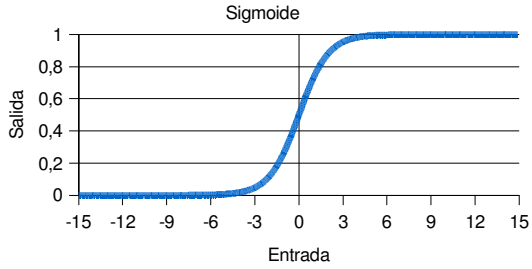
El modelo de mundo que escoge el agente como apropiado a la hora de buscar una cierta estrategia, se obtiene mediante un proceso evolutivo que usa para calcular la función de calidad los pares acción-percepción que se guardan en la Memoria a Corto Plazo.

El mecanismo comienza con una población aleatoria de modelos de mundo que se van perfeccionando a medida que el agente interacciona con el entorno. La evolución de los modelos de mundo no debe implicar muchas generaciones entre interacciones con el entorno (iteraciones del MDB), porque estaríamos consiguiendo una población buena para una cierta situación (ideal para escoger la estrategia actual) pero no en general, es decir, estaríamos premiando la optimización sobre el aprendizaje. En una iteración del mecanismo dada, la Memoria a Corto Plazo guarda información parcial del mundo debido a que hemos limitado su capacidad y no nos interesa que el mecanismo aprenda el contenido de dicha memoria en ese caso o momento particular. Por este motivo fundamental, debemos limitar las generaciones de evolución entre iteraciones. Además, una excesiva evolución implica que la población pueda converger, y su variedad resulta tan reducida que en la siguiente iteración del mecanismo (otra situación que puede requerir otra estrategia) sería muy complicado llegar a una solución apropiada ya que los procesos de cruce no aportarían nada nuevo, y estaríamos trabajando solo con mutación, es decir, con una búsqueda casi aleatoria. Volveremos a analizar este hecho en el apartado 6.2.2 donde se estudia a fondo la Memoria a Corto Plazo.

Independientemente del tipo de algoritmo evolutivo que utilicemos, los individuos pueden codificar directamente los modelos de mundo o, de un modo más general, pueden estar formados por una estructura que, dadas unas entradas (entradas sensoriales en t y acción), nos proporcione las salidas deseadas (entradas sensoriales en t+1). Hemos optado, inicialmente, porque cada modelo de mundo esté representado por una red neuronal de tipo perceptrón multicapa sin perjuicio de que pueda ser sustituido por otro tipo de representación. Como consecuencia, dichos modelos de mundo se pueden ver de forma esquemática:



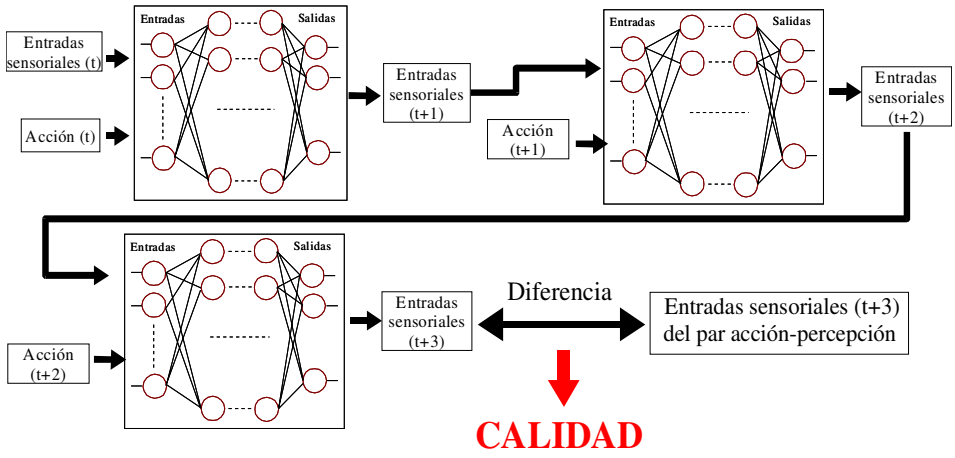
Utilizaremos como función de activación de cada neurona, una sigmoide simple $y = \frac{1}{(1+e^{-(x+B)})}$ como la de la figura:



En cuanto a los parámetros de las redes, vamos a evolucionar únicamente los pesos sinápticos y el parámetro B (bias) de cada función de activación como la anterior.

Recientemente hemos aplicado otro tipo de red neuronal como sustrato de los modelos [López Peña, 03], en este caso una red con retardos sinápticos [Duro, 99] obteniendo resultados muy prometedores en un problema de modelado de funciones en tiempo real a partir de muestras.

Como hemos venido diciendo en los apartados dedicados al aprendizaje, la función de calidad puede ser directamente la diferencia entre lo que predice la red y el valor correspondiente a las entradas sensoriales en t+n que se almacena en los pares acción-percepción, ponderado para todos los pares presentes en la Memoria a Corto Plazo. Al trabajar con estrategias de longitud *n*, la calidad de un cierto modelo de mundo se conoce tras aplicar las *n* acciones en dicho modelo como se muestra en el siguiente esquema para una estrategia de longitud 3:



La diferencia, medida mediante algún tipo de métrica adecuada, entre la *entrada sensorial en t+3* que predice el modelo con lo que guarda el par acción-percepción nos proporciona el valor de calidad que utilizaremos para la estrategia.

Por último, no debemos olvidar que la Memoria de Modelos de Mundo que aparece en el diagrama de bloques corresponde a una Memoria a Largo Plazo en la que se almacenan los modelos que han resultado satisfactorios.

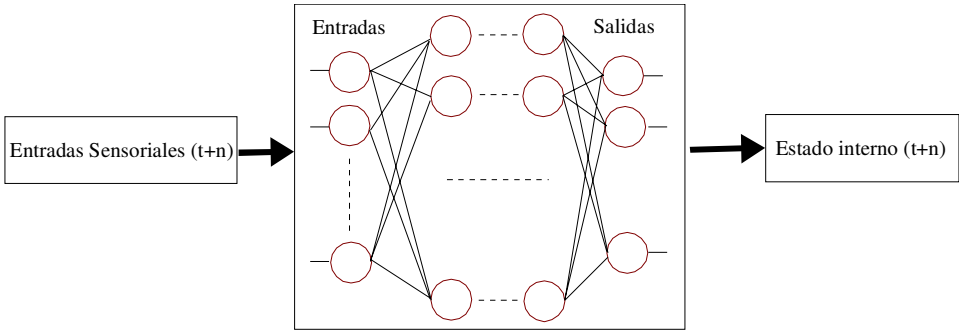
Población de modelos de mundo

En relación directa con el apartado anterior se encuentra la memoria que guarda la población de modelos de mundo. Al comienzo del mecanismo, esta población es aleatoria y eso implica que los primeros modelos dan resultados poco satisfactorios. De una iteración del mecanismo a la siguiente la población evolucionada *se mantiene*, por lo que con el paso del tiempo toda la población tiende mejorar. Como hemos dicho, con la técnica evolutiva no buscamos un superindividuo, sino una población globalmente buena.

Volvemos así a recalcar la idea de que el proceso de obtención de los modelos es un proceso de aprendizaje sobre las poblaciones de modo que, para que lo aprendido no se pierda, dichas poblaciones se deben mantener.

Modelos internos

Al igual que en el caso de los modelos de mundo, los modelos internos son aprendidos mediante un algoritmo evolutivo tomando como patrón la función real que modela el grado de satisfacción de la motivación. De nuevo, la población inicial de modelos internos es aleatoria y dichos modelos se pueden codificar directamente mediante reglas o pueden estar formados por redes neuronales u otra estructura similar. En el caso de que se representen mediante redes, su estructura esquemática sería la que se muestra en la figura siguiente, donde no incluimos las entradas sensoriales internas de acuerdo con lo establecido en el esquema funcional del apartado 5.2:



Teniendo en cuenta que el estado interno es máximo cuando la satisfacción de la motivación es máxima, es trivial establecer una función tal que, tras la aplicación de una estrategia en el entorno, nos proporcione dicho estado interno real de forma directa. En la evolución de los modelos internos, para cada par acción-percepción, dadas unas entradas sensoriales en t+n el estado interno predicho por el modelo interno se compara con el estado interno real que almacena dicho par. La diferencia entre ellos constituye el valor de calidad de cada individuo de la población de modelos internos. De nuevo, esta calidad debe ser ponderada entre todos los casos existentes en la Memoria a Corto Plazo.

La evolución de los modelos internos entre iteraciones no debe ser excesiva tal y como comentamos antes para los modelos de mundo para evitar que generalice situaciones particulares. Pero debemos tener en cuenta que el modelo interno es menos propenso a caer en un mínimo local ya que supondremos que la motivación no cambia tanto como las consecuencias de las acciones en el entorno, sobre todo en un entorno dinámico. Es decir, los modelos de mundo pueden cambiar más fácilmente que los modelos internos en la "vida" de un agente.

Población de modelos internos

Al igual que con los modelos de mundo, la memoria de modelos internos guarda la población que se utiliza en el proceso evolutivo. Al comienzo del mecanismo, esta población es aleatoria y los primeros modelos son poco adecuados por lo que, en estas primeras iteraciones, hay que tener en cuenta que al error en la salida del modelo de mundo se le une el error a la salida del modelo interno. Esto implica que la predicción en el estado interno es prácticamente aleatoria, sin ninguna relación con la deseada en estas primeras iteraciones. De nuevo, las poblaciones de modelos internos se mantienen entre iteraciones del MDB.

Estrategias

La obtención de las estrategias sigue un proceso evolutivo análogo al de los modelos de mundo e internos, aunque no tendría por qué ser así. Los modelos deben ser aprendidos, mientras que las estrategias, en principio, deben ser simplemente optimizadas sobre dichos modelos. Es decir, para la obtención de la mejor estrategia se utilizará, en la mayor parte de los casos, un algoritmo evolutivo por generalidad y porque resuelven problemas de optimización de manera totalmente satisfactoria, pero sin entrar en las consideraciones de aprendizaje realizadas para los modelos.

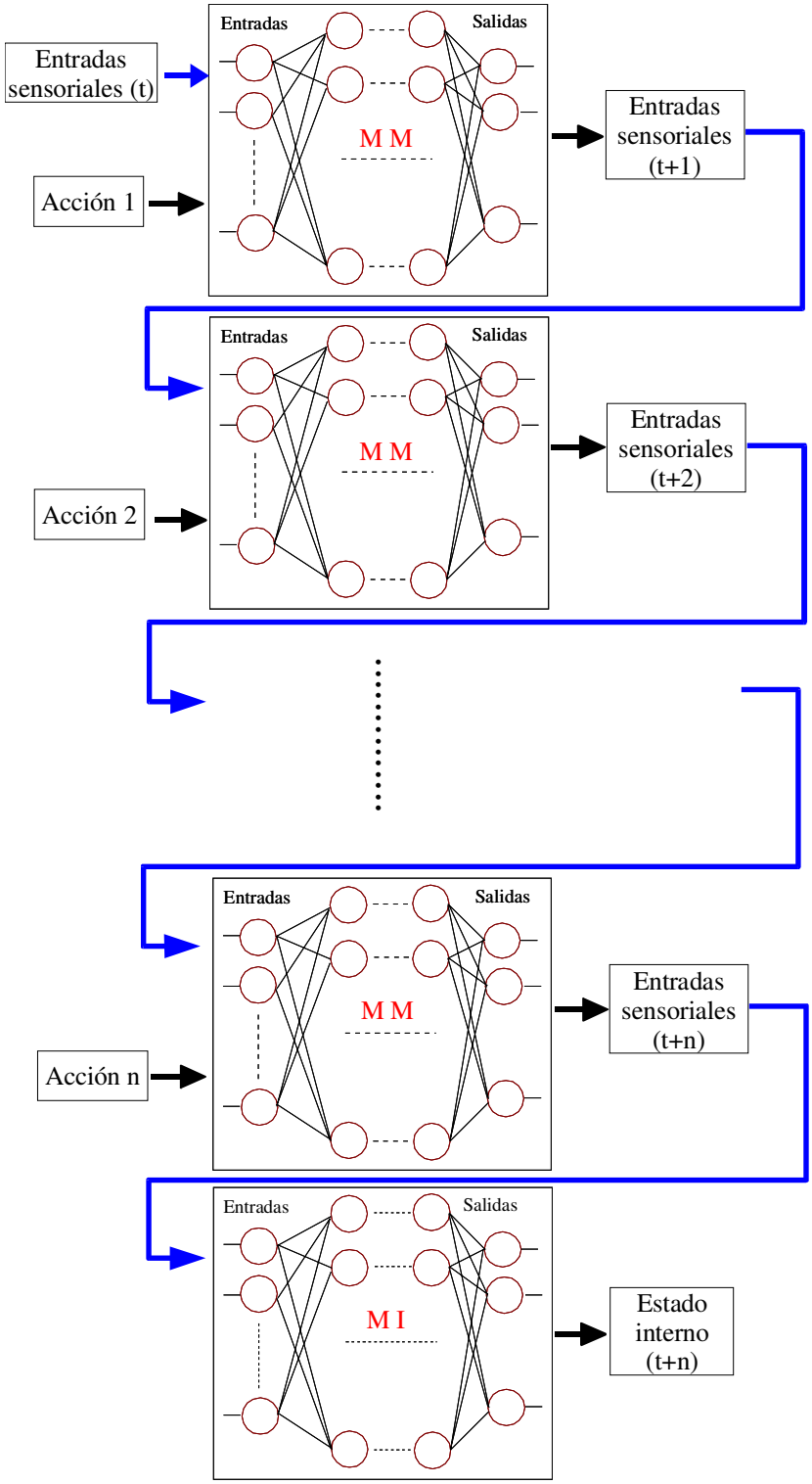
El concepto de estrategia implica que el mecanismo es capaz de prever las consecuencias de sus acciones a largo plazo, de forma que puede escoger una secuencia de ellas. En la primera iteración del mecanismo, se parte de una población aleatoria de estrategias. En este caso, el número de generaciones de evolución puede ser mayor que en los modelos porque en cada iteración buscamos la mejor estrategia independientemente de la población de partida. De hecho, los mejores resultados fueron obtenidos partiendo de una población de estrategias aleatoria en cada iteración y dejando la evolución hasta que dicha estrategia fuese óptima. Esto es debido a que la evolución de las estrategias, al menos en los casos que hemos tratado, es un proceso muy rápido y buscamos la mejor solución posible en cada instante. Al contrario de lo que ocurre con los modelos, ahora sí nos interesa la mejor estrategia posible para las condiciones actuales de cada iteración. Cuando el tamaño o la complejidad de las estrategias sea un límite al tiempo de procesado, la evolución deberá ser corta como en el caso de los modelos para que la memoria de estrategias no contenga únicamente soluciones parciales.

A la hora de calcular la calidad de una cierta estrategia se realiza el siguiente ciclo, que podemos analizar con más claridad en el esquema de la página siguiente:

1. El último par acción-percepción nos proporciona las entradas sensoriales al modelo de mundo actual elegido tras la evolución de los modelos de mundo.
2. De la estrategia que queremos calcular la calidad, tomamos el primer conjunto de acciones a los actuadores y completamos las entradas al modelo de mundo.
3. Del modelo de mundo actual obtenemos como salidas la sensorización predicha en t+1.
4. Estas salidas se ponen como entradas de nuevo al modelo de mundo actual junto con el segundo conjunto de acciones a los actuadores obteniendo así la sensorización predicha en t+2.
5. Este proceso se repite tantas veces como conjuntos de acciones contenga la estrategia (n acciones en el esquema).
6. La última salida proporcionada por el modelo de mundo actual se usa como entrada del modelo interno actual. La salida del modelo interno nos da el estado interno predicho en t+n que, de acuerdo con el grado de satisfacción de la motivación, se usa como función de calidad para la estrategia.

Hemos optado por el uso de estrategias porque permiten la obtención de comportamientos complejos como, por ejemplo, la aplicación de acciones que impliquen poca satisfacción en un periodo determinado que posteriormente se compensa con acciones satisfactorias, de modo que se justifica una pérdida de eficiencia intermedia con el logro final de una eficiencia global mayor. Además, mientras el agente aprende los modelos, es necesaria la mayor cantidad de información posible, y ésta se consigue aplicando muchas acciones y sensando muchas veces.

La calidad de cada estrategia podría ser compleja y no sólo tener en cuenta que el estado interno sea el buscado, sino también el coste asociado al logro de dicho estado interno (longitud de la estrategia).


Entradas sensoriales (t)
Acción 1
Entradas
Salidas
M M
Entradas sensoriales (t+1)
Acción 2
Entradas sensoriales (t+2)
Acción n
Entradas sensoriales (t+n)
M I
Estado interno (t+n)

Población de estrategias

En principio, la memoria de estrategias sirve como población inicial en la evolución de las estrategias y se puede conservar de una iteración del mecanismo a la siguiente. Pero, como acabamos de comentar, esto no tiene por qué ser así de rígido ya que una estrategia útil en un cierto instante de tiempo (en un cierto contexto) no tiene por qué ser buena en el siguiente. Parece más adecuado que la evolución de las estrategias no se detenga hasta encontrar la óptima partiendo si fuese necesario de una población nueva cada vez, y conservar en otra Memoria a Largo Plazo aquellas estrategias que resultaron buenas.

En los primeros ejemplos de aplicación del MDB que se presentan en el apartado 7.1, utilizamos un algoritmo genético simple para la evolución de los modelos y las redes neuronales que representan los individuos son de tamaño fijo. En este punto del trabajo, es posible la lectura de dichos ejemplos ya que fueron realizados de cara a probar el funcionamiento del MDB como concepto, es decir, del esquema funcional presentado en el apartado 5.2. Estas primeras experiencias resultaron muy satisfactorias y, a la vez, mostraron las carencias de estas representaciones simples a la hora de tratar con problemas reales. Por este motivo decidimos desarrollar un nuevo tipo de algoritmo evolutivo basado, de nuevo, en un concepto biológico: el gen promotor. Esto ha dado lugar al algoritmo PBGA que pasamos a analizar en detalle.

Algoritmo genético basado en genes promotores (PBGA)

En las primeras pruebas que realizamos con el MDB y que se muestran en el apartado 7.1, los procesos evolutivos involucrados en la obtención de los modelos se llevaron a cabo mediante algoritmos genéticos básicos donde los individuos de la población eran redes neuronales perceptrón multicapa de tamaño fijo. Así obtuvimos modelos en tiempo real a partir de muestras discretas de una forma eficiente y precisa, aunque siempre trabajando con entornos simples. Este tipo de entorno resultó adecuado para sentar las bases del MDB y realizar las primeras pruebas sobre el mismo, pero con la filosofía de que el mecanismo se pueda aplicar a problemas reales con agentes reales, se hace necesaria la utilización de entornos más complejos.

Cuando comenzamos a aplicar el MDB en casos reales, nos encontramos con el problema fundamental de que los *modelos de mundo* pueden llegar a ser muy difíciles de obtener porque las redes que los representan tiene un número elevado de entradas y salidas y, además, porque las funciones a aproximar son complejas. Como consecuencia, se hace necesaria una etapa previa que ralentiza el estudio, donde se prueban diferentes tamaños en las redes que forman las poblaciones de modelos de mundo hasta que se encuentra la solución más adecuada. En muchos de estos casos, esta solución es muy pobre y los modelos de mundo predicen el entorno con un error cercano a lo inaceptable. Este problema no nos ha ocurrido, al nivel que nos movemos en la actualidad, con los modelos internos porque su codificación es más subjetiva y el número de variables de entrada y salida es más fácil de manipular.

Existen una serie de simplificaciones que pueden ser adoptadas para tratar con entornos dinámicos complejos sin perder relevancia en el planteamiento del problema, como por ejemplo, la inclusión de sensores y actuadores virtuales que reduzcan el número de variables de entrada y salida necesarias, además de realizar un procesado de la información que elimine parte del ruido inherente a los datos reales. En ciertos casos, ni siquiera con estas simplificaciones es viable la obtención de un modelo de mundo, o directamente no son aplicables. Llegados a este punto, la forma de abordar el modelado de entornos dinámicos complejos pasa por realizar modificaciones y mejoras en los algoritmos de aprendizaje, y en la reutilización de la información aprendida. De hecho, sería de gran utilidad poder obtener los modelos globales del entorno como combinación de submodelos más simples, ya que éstos serán más fáciles de aprender, serán reutilizables y su adaptación a entornos dinámicos resultará más eficiente. De esta forma, debemos mantener de manera constante poblaciones de posibles submodelos en la evolución y escogerlos o combinarlos en función de los cambios externos.

La principal causa por la que los algoritmos genéticos y evolutivos estándar fallan cuando se aplican a este tipo de problemas dinámicos, es que tienden a converger a poblaciones homogéneas. En entornos estáticos esto no es un problema si la convergencia de toda la población se alcanza tras lograr el óptimo. Desafortunadamente, no hay forma de garantizarlo y dicha convergencia se puede ver muy disminuida si la diversidad se estanca antes de alcanzar el óptimo global, ya que la única forma de seguir una función de calidad cambiante es mediante el operador de mutación, que se traduce en una búsqueda prácticamente aleatoria y muy lenta.

Para solventar este problema, los investigadores han considerado tres aproximaciones distintas. Dos de ellas se basan en acciones exógenas sobre la población y la tercera actúa sobre la transcripción genotipo-fenotipo. En el primer grupo, algunos autores han adoptado soluciones consistentes en la introducción de individuos en la

población cuando se considera necesario para generar diversidad. Estos individuos se pueden generar de forma aleatoria tal y como ha hecho Grefenstette en [Grefenstette, 92], o mediante algún criterio que implique la utilización de una segunda memoria que guarde individuos relevantes que se inyectan en la población, solución adoptada en el trabajo de Hartono y Hashimoto [Hartono, 01]. Otros investigadores han considerado mecanismos de selección más restrictivos para forzar un determinado nivel de diversidad genética en las poblaciones, utilizando conceptos como similitud entre individuos según una determinada métrica. Un ejemplo de este tipo de aproximación ha sido desarrollado por Kubalik en [Kubalik, 02]. En ambos casos, se requiere la definición de una medida de distancia consistente en la población.

Un modo más biológico de afrontar el problema requiere trabajar con individuos en representación genotípica y realizar la transformación al fenotipo a la hora de aplicarlos. En los últimos años, se ha incrementado el número de estudios que utilizan este tipo de técnica, como son los de Ryan en [Ryan, 98], Peow en [Peow, 95], Kargupta en [Kargupta, 99] o Vassilev en [Vassilev, 00]. En esta línea, existen dos tipos de aproximaciones a la hora de expresar los genes: representaciones diploide y mecanismos basados en genes promotores. Los genotipos diploides están formados por una estructura de cromosoma doble donde cada hebra contiene información para las mismas funciones. Cada vez que se construye un fenotipo a partir del genotipo, uno de los dos posibles alelos de cada gen se selecciona mediante un criterio de dominancia que puede cambiar con el tiempo. Como no se expresan todos los genes que forman el cromosoma y como la calidad de un individuo se determina por su fenotipo, los genes recesivos se protegen de la presión selectiva proporcionando así una memoria en la propia codificación del genotipo. Este tipo de técnicas fueron introducidas en la computación evolutiva por Goldberg [Goldberg, 87] que afirmó que una representación diploide combinada con un mapa de dominancia puede superar a un algoritmo genético estándar en problemas dinámicos. Podemos encontrar distintos estudios y aplicaciones de este mecanismo en la bibliografía, como por ejemplo [Ryan, 98] y [Peow, 95].

En nuestro caso, con el objetivo de automatizar y simplificar el proceso de obtención de los modelos de mundo en entornos dinámicos complejos, hemos desarrollado un algoritmo genético que, basado en transformaciones genotipo-fenotipo, utiliza el concepto biológico de gen promotor. Por este motivo ha sido bautizado como *Algoritmo Genético basado en Genes Promotores (PBGA)* y ha sido diseñado con los siguientes objetivos y características básicas:

1. Debe mantener la estructura básica de los algoritmos genéticos clásicos por su gran capacidad de generalización y búsqueda en el espacio de funciones.
2. Debe manejar de forma eficiente redes neuronales, por ser generalizadores matemáticos universales muy adecuados a nuestro problema.
3. Debe ser capaz de dividir la función compleja a modelar en partes más simples obteniendo el número de redes adecuado de forma automática. Esta característica es fundamental para poder abordar el funcionamiento en entornos reales.
4. Debe ser capaz de manejar estos modelos simples que evolucionan en la misma población y que pueden ser muy diferentes (en el caso de las redes, éstas pueden tener distinto número de entradas y salidas) de modo que la evolución no se estanque por falta de diversidad.

5. Debe trabajar con individuos en representación genotípica que son transformados a representación fenotípica al ser aplicados. Como hemos comentado previamente, esta es una solución muy eficiente a la hora de evitar el estancamiento en la evolución al tratar con problemas dinámicos.
6. Debe minimizar la fase de pruebas para encontrar el tamaño de las redes necesarias en la evolución de los modelos obteniendo dicho tamaño de forma automática. La transformación no directa genotipo-fenotipo favorece la utilización de una representación compleja que se simplifica al ser aplicada.

En los siguientes apartados, explicaremos en profundidad el algoritmo propuesto, pero antes de nada pasamos a realizar una breve introducción biológica sobre los genes promotores y su relación con el PBGA.

Genes promotores

En las células simples, la codificación del ADN para sintetizar una proteína es continua. En células más complejas, dicha codificación es discontinua y secuencias de ADN codificado, denominadas exones, se encuentran intercaladas con secuencias más largas de ADN no codificado. Estas últimas secuencias se denominan intrones y son tan comunes que los investigadores se niegan a aceptar, como parece hoy en día, que no cumplan ninguna función específica.

Los aminoácidos que constituyen una proteína se deben ir añadiendo según una secuencia correcta, existiendo un triplete de nucleótidos consecutivos (codón) que especifica un aminoácido. Por tanto, un codón es un grupo de tres bases (A, T, C ó G) que codifica un solo aminoácido y que establece dónde se decodifica una proteína. El codón de inicio está formado por las letras ATG, de modo que, cuando la maquinaria de la célula ve este primer ATG, sabe que las instrucciones para construir una proteína comienzan en ese punto. El codón de fin le indica a la maquinaria de la célula que ha alcanzado el final de la proteína y que debe parar de trasladar el código.

A partir del anterior proceso se puede definir como *gen* un conjunto de nucleótidos de una molécula de ADN que sirve como molde para la producción de una proteína o una familia de proteínas. Los *genes promotores* son importantes estructuras reguladoras que controlan el inicio y el nivel de transcripción de un gen. Se localizan antes del gen y controlan si debe estar habilitado o no, y durante cuánto tiempo. Estas estructuras son muy interesantes desde el punto de vista del diseño del algoritmo porque nos dan idea de cómo se puede controlar la transcripción genotipo-fenotipo. El algoritmo PBGA desarrollado, utiliza el concepto de gen promotor como principio estructural y de funcionamiento.

Esquema básico del PBGA

Comenzamos la presentación del algoritmo por el sustrato del mismo, y es que cada individuo de la población del PBGA es una red neuronal (de cualquier tipo). En un algoritmo genético clásico dentro del aprendizaje evolutivo, el cromosoma de este tipo de individuo está formado, en el caso más simple, por los pesos sinápticos de las distintas capas. Por ejemplo, si suponemos una red con la estructura que se muestra en el esquema de la figura 1, un posible cromosoma en un algoritmo clásico sería:

$$[W_{13}\ W_{23}\ W_{14}\ W_{24}\ W_{15}\ W_{25}\ W_{36}\ W_{46}\ W_{56}]$$

de acuerdo con la nomenclatura para los nodos y los pesos sinápticos que aparece en el esquema de la red. En este cromosoma aparece un peso tras otro con un orden conocido y fijo, necesario a la hora de aplicar los operadores de reproducción. Además de los pesos sinápticos, se pueden incluir en el cromosoma distintos parámetros de las funciones de activación presentes en las neuronas (tal y como hemos explicado anteriormente) pero, por simplicidad, en el ejemplo han sido omitidos. En este caso, la representación genotípica se corresponde directamente con el fenotipo.

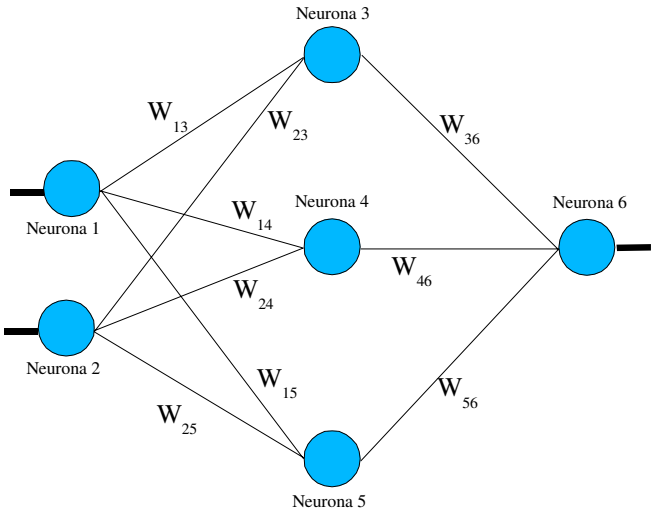


Figura 1. Representación esquemática de una red neuronal genérica

En el PBGA, proponemos la inclusión de un parámetro asociado a cada neurona y que posee únicamente dos estados: activo o inactivo. Es, por tanto, un parámetro binario de valor 1 ó 0 que está presente en la representación genotípica del cromosoma y que indica si la neurona a la que está asociado estará o no presente al realizar la transcripción al fenotipo. Esta es, de forma discretizada, la función biológica de los genes promotores y, por tanto, utilizaremos la nomenclatura de gen promotor para referirnos al valor asociado a este parámetro regulador. Así, el esquema anterior de la red neuronal se modifica con la inclusión de genes promotores en las neuronas, quedando tal y como se muestra en la figura 2 únicamente si todos los genes promotores están activos. En cuanto a la codificación genotípica del cromosoma, hemos incluido el valor de cada gen promotor previamente a los pesos a los que afecta del modo siguiente:

$$[\mathbf{1}\ \mathbf{1}\ \mathbf{1}\ W_{13}\ W_{23}\ \mathbf{1}\ W_{14}\ W_{24}\ \mathbf{1}\ W_{15}\ W_{25}\ \mathbf{1}\ W_{36}\ W_{46}\ W_{56}]$$

de tal forma que los dos primeros unos representan los genes promotores asociados a la neuronas 1 y 2 (neuronas de entrada). El tercer uno representa el gen promotor de la neurona 3, con los pesos W_{13} y W_{23} asociados. Esto quiere decir que, en el fenotipo de la red (representada en la figura 2), esta neurona y estos dos pesos estarán presentes. El cuarto uno tiene asociados los pesos W_{14} y W_{24}, y así sucesivamente hasta el último uno que representa la neurona de salida y tiene tres pesos asociados.

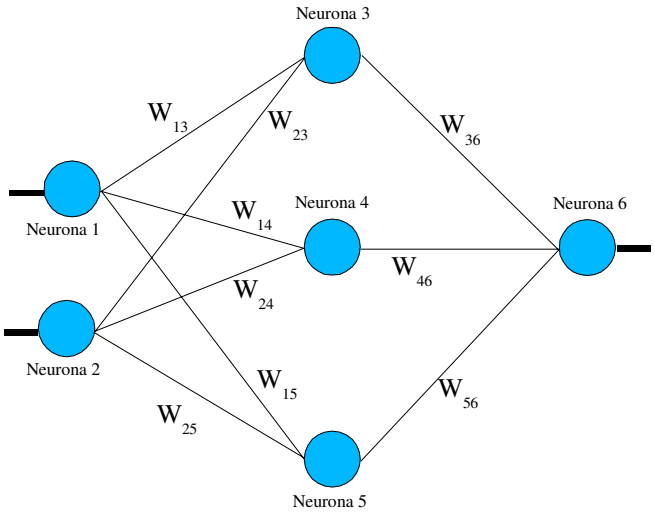
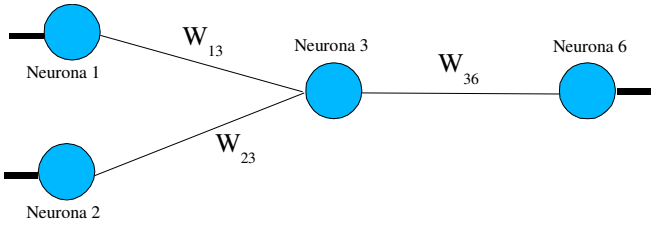


Figura 2. Fenotipo de una red neuronal con todos los genes promotores activos.

Para otro individuo, el genotipo podría ser:

$$[\mathbf{1}\ \mathbf{1}\ \mathbf{1}\ W_{13}\ W_{23}\ \mathbf{0}\ W_{14}\ W_{24}\ \mathbf{0}\ W_{15}\ W_{25}\ \mathbf{1}\ W_{36}\ W_{46}\ W_{56}]$$

donde la diferencia reside en que aparecen dos genes promotores a cero asociados a las neuronas 4 y 5, por lo que los pesos asociados W_{14} W_{24} W_{15} W_{25} W_{46} W_{56} no aparecen en el fenotipo, que sería simplemente:



En resumen, el PBGA utiliza como cromosoma para el individuo una representación genotípica compleja que incluye los pesos y los genes promotores asociados a cada neurona, por lo que estos genes promotores también entran en la evolución como un gen más del cromosoma. A la hora de calcular la calidad de la red, es decir, a la hora de ser aplicada en la práctica, generamos su fenotipo utilizando para ello sólo las conexiones asociadas a neuronas con genes promotores activos, de valor 1.

Por tanto, la inclusión de los genes promotores en la codificación del cromosoma del algoritmo genético, nos permite utilizar una transformación no directa genotipo-fenotipo como habíamos planteado al inicio del apartado. Además, nos permite obtener de forma automática la topología de la red en cuanto al número de neuronas en cada capa y, por tanto, en cuanto al número de capas ocultas. Esto no es nuevo en el campo del aprendizaje evolutivo, y distintos autores [Yao, 93] han desarrollado algoritmos evolutivos que trabajan con redes de longitud variable. En la práctica totalidad de los casos, el fundamento de estos algoritmos consiste en añadir o eliminar neuronas y sus

correspondientes conexiones sinápticas en la propia representación genotípica. No dejan de ser, por tanto, algoritmos genéticos clásicos donde simplemente el cromosoma es más complejo y, por tanto, las operaciones de generación de diversidad (cruce y mutación) son también más complejas.

Cualquier algoritmo de los comentados, obtiene el tamaño de la red neuronal que es solución al problema de forma automática. La novedad que aporta el PBGA reside fundamentalmente en dos aspectos:

1. El cromosoma utiliza la representación genotípica y su tamaño no cambia. De este modo, neuronas con su gen promotor inactivo (a cero) también entran en el proceso evolutivo, por lo que a sus pesos asociados se les aplican los operadores de selección y reproducción. Esto no ocurre en los algoritmos clásicos que trabajan con redes de longitud variable donde el tamaño del cromosoma también es variable. En el PBGA, la activación o desactivación de un gen promotor tiene el mismo efecto final que en los algoritmos clásicos, el crear o destruir una neurona, pero el impacto sobre el cambio en los pesos sinápticos es menor porque nunca desaparecen ni se crean. Como explicaremos más adelante en este apartado, estas neuronas inactivas también entran en los procesos de reproducción.
2. Dado que las neuronas de entrada y salida también tienen asociado un gen promotor, pueden estar o no presentes en el fenotipo. Esto implica que el algoritmo obtiene de forma automática las entradas que son relevantes para el cálculo de las salidas. En un algoritmo clásico, las variables de entrada y salida son siempre fijas y simplemente se trabaja con las neuronas y capas intermedias.

Esta segunda característica del PBGA fue tenida en cuenta como requerimiento de diseño y la hemos comentado en la introducción. Deseamos que, de forma automática, el algoritmo divida una función compleja en funciones más simples si la evolución lo considera necesario. Al utilizar los genes promotores en las neuronas de entrada y salida, estamos otorgando al PBGA un grado de libertad enorme porque puede prescindir de ciertas entradas en el cálculo de ciertas salidas e incluso prescindir de las propias salidas.

De cualquier modo, las salidas de las redes utilizadas en este tipo de problemas de aprendizaje evolutivo están fijadas y son siempre necesarias. Es decir, no tiene sentido que las redes no proporcionen salidas que no se utilizan, ya que en ese caso no existirían. Por tanto, el PBGA tiene libertad para obtener redes que fenotípicamente no posean todas las salidas activas, pero imponemos que cada red tenga, al menos, una neurona de salida activa y que la solución final que se presente cubra todas las salidas del problema. Se nos abre así un nuevo concepto de solución como combinación de varias soluciones individuales, que están especializadas en distintas características de la función global a obtener.

La formación de redes que realizan funciones diferentes no tiene por qué darse siempre, ya que existen soluciones más fáciles de obtener con una única red que tenga todas las salidas activas. El PBGA no impone ninguna restricción en este sentido, y es la propia evolución (la dificultad del problema) la que determina si se debe realizar la especialización por salida.

El principal problema con que nos enfrentamos al plantear un algoritmo de este tipo, es que en la población existen redes muy diferentes evolucionando juntas. Por una parte buscamos que se formen automáticamente estas redes especializadas para dividir la

tarea compleja de obtener una función, pero por otra parte el algoritmo evolutivo clásico no está preparado para trabajar simultáneamente con soluciones distintas a problemas diferentes. Por ejemplo, si no se modifica el operador de selección, la reproducción que le sigue estará mezclando individuos que corresponden a soluciones diferentes, de modo que los nuevos individuos que se forman podrían resultar inadecuados.

En resumen, el PBGA es un algoritmo evolutivo que utiliza una transformación genotipo-fenotipo para las redes neuronales representadas en los cromosomas. Cada unidad funcional, esto es, cada neurona con los pesos que le llegan, de las redes está controlada por un parámetro binario denominado gen promotor, que especifica si dicha neurona estará o no presente en el fenotipo. Este tipo de representación permite obtener la topología de las redes de forma automática y permite, además, la formación de grupos de redes especializados en distintas partes de la solución global que evolucionan juntos en la misma población. Para que la evolución de estos individuos se realice de forma óptima, es necesaria una revisión de los operadores básicos de los algoritmos evolutivos: selección, cruce y mutación.

Selección

El PBGA debe proporcionar una única solución como cualquier algoritmo de búsqueda de un óptimo matemático, aunque ésta sea compleja. Es decir, las n salidas de las redes que se obtienen en el proceso evolutivo deben estar cubiertas en el sentido de que debe haber al menos una red que contenga cada salida activa. Pero no se impone ningún tipo de restricción en cuanto al número de redes necesarias y, así, debe ser igual de probable la formación de redes de una única salida activa como la formación de redes con n salidas activas. La proporción de redes de cada tipo vendrá impuesto por la propia evolución, es decir, por la función de calidad.

El operador de selección clásico, explicado en el apartado dedicado a los algoritmos genéticos, se basa en la calidad de cada individuo, de modo que tiende a favorecer la reproducción de aquellos miembros de la población que tienen mejor calidad siguiendo un criterio de búsqueda rápida del óptimo. Este tipo de operador no es aplicable en el PBGA, porque ahora en la selección se debe tener en cuenta que los individuos son diferentes, y el uso de la calidad como criterio no tiene en cuenta criterios funcionales o estructurales de los individuos. Se hace necesario, por tanto, un nuevo operador de selección que permita la coexistencia de individuos diferentes topológica y funcionalmente en la misma población, de tal forma que el operador de reproducción se aplique a individuos del mismo tipo de cara a que el proceso evolutivo converja. Las principales características de funcionamiento de este nuevo operador son las siguientes:

1. De una población de P individuos creamos tantas subpoblaciones como variables de salida tienen las redes a obtener.
2. Cada subpoblación asociada a una salida determinada está formada por todos los individuos que tienen dicha salida activa (y que además pueden tener otras).
3. En cada una de estas subpoblaciones se aplica el operador de selección por torneo clásico y los operadores de reproducción que serán presentados en el siguiente apartado.

Por tanto, hemos dividido el proceso de selección en dos partes, en una primera se generan subpoblaciones con individuos funcionalmente iguales en cuanto a la salida que contemplan y en otra se aplica el proceso de selección de los mejores individuos para la

reproducción de forma clásica. Así, los nuevos individuos que se forman descienden de individuos análogos y la convergencia del proceso evolutivo se mantiene. En la figura 3 mostramos un esquema del proceso de selección presentado aplicado a un problema con dos variables de salida, en el que podemos observar cómo se forman las subpoblaciones y cómo se reconstruye la población original tras la reproducción. Las poblaciones representadas están formadas por individuos con cromosomas que incluyen genes promotores además de pesos sinápticos, como hemos introducido anteriormente. Para simplificar el esquema, de dichos cromosomas sólo mostramos los genes promotores asociados a las neuronas de salida, de tal modo que un individuo representado por [...W_i.....**1**W_i....**0**] tiene la primera neurona de salida activa y la segunda inactiva. De esta forma, en la población inicial existen individuos de todo tipo: con la primera salida activa únicamente, con las segunda salida activa o con ambas. En general, en un problema con *n* variables de salida existen 2^n-1 tipos de redes diferentes por salida.

De esta población heterogénea inicial se crean dos subpoblaciones que, como vemos en la figura 3, contienen individuos similares en cuanto a las neuronas de salida que están activas. Así, la primera subpoblación que se crea contiene individuos con la primera salida activa como requisito de pertenencia. Esta subpoblación se transforma en otra tras la aplicación de los operadores de cruce y mutación. Lo mismo ocurre para la segunda subpoblación, donde todos los individuos tienen la segunda salida activa. La última etapa que se muestra en la figura corresponde a la creación de una nueva población de tamaño inicial *P* creada escogiendo los mejores individuos de cada subpoblación de forma equitativa.

Este esquema de selección requiere que la calidad de cada individuo se calcule por salida, de modo que un individuo lleva asociados tantos valores de calidad como variables de salida tiene el problema. Esta es una de la nuevas características del PBGA, ya que introduce la posibilidad de que un individuo sea mejor para una salida que para otras, o bien que sea adecuado para varias (cuantas más neuronas de salida tenga activas). Esta calidad por salida, se utiliza en el proceso de selección por torneo que tiene lugar en cada subpoblación. Es decir, dentro de cada subpoblación, el algoritmo genético tiene un comportamiento totalmente clásico y cada individuo tiene un valor de calidad asociado. Cuando se actualiza este valor de calidad, se compara el resultado proporcionado por cada individuo con el valor correspondiente para cada salida por separado y nunca se calcula una calidad global de la red.

El principal problema que presenta este tipo de selección es que no es estadísticamente neutra y aparece una tendencia hacia la existencia de redes con mayor número de salidas activas ya que éstas participan en más subpoblaciones. A continuación analizamos y proponemos una solución para este problema.

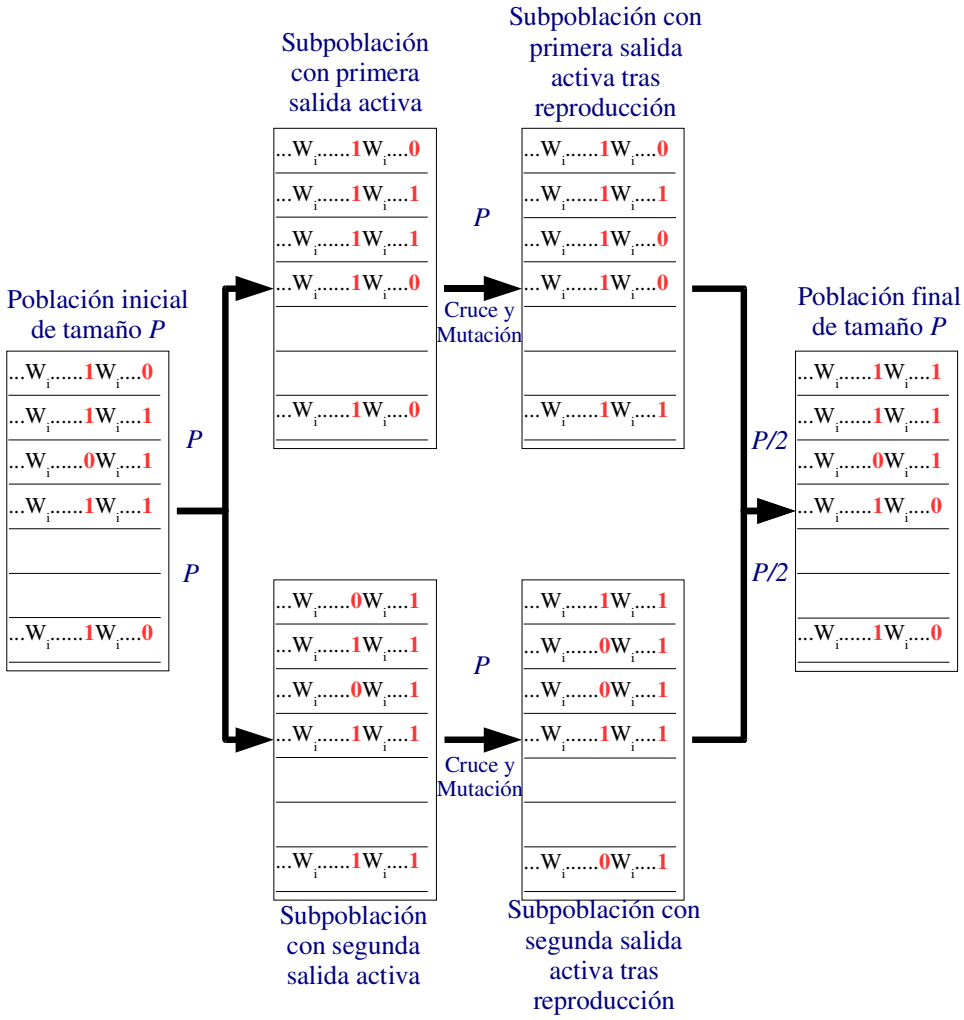


Figura 3. Esquema del proceso de selección utilizado en el PBGA para redes de dos salidas

Selección estadísticamente neutra

La probabilidad que tiene un cierto individuo de ser seleccionado en el proceso de selección clásico por torneo, depende del tamaño de la ventana de selección, de la presión selectiva y de la calidad del individuo. Esto es lo que ocurre en cada una de las subpoblaciones con las que trabaja el PBGA. Pero los individuos con mayor número de salidas activas, tienen mayor probabilidad de entrar en más de dichas subpoblaciones, ya que la selección en esta primera fase sólo depende del número de salidas activas.

Por ejemplo, si continuamos con el caso de redes de dos salidas tenemos que, al crear la subpoblación con la primera salida activa, todos los individuos con las dos salidas activas pueden ser seleccionados. Pero también podrán ser seleccionados en la otra subpoblación dado que tiene la segunda salida activa. Es obvio que, si no controlamos

las proporciones de redes de cada tipo, cuanto mayor sea el número de salidas activas, mayor es la probabilidad de estar presente en la siguiente generación. Esto introduce un sesgo no deseado en el proceso de selección ajeno a la función de calidad, es decir, ajeno a la complejidad del aprendizaje de la función.

La solución más eficiente a este problema consiste en que las subpoblaciones contengan mayor número de individuos cuanto menor sea el número de salidas activas, de modo que la proporción de individuos de cada tipo en todas las subpoblaciones sea el mismo. Así, para el caso de dos salidas y una población de tamaño P, en cada subpoblación debe haber $2P/3$ individuos de una única salida activa y $P/3$ individuos con las dos salidas activa. Si en la población original no hay tal número de individuos de un cierto tipo, deberán ser replicados hasta llegar al número adecuado. Para problemas con un número n de variables de salida, la proporción de individuos con una única salida activa debe ser de n veces la proporción de individuos con todas las salidas activas, de $n-1$ veces la de individuos con todas las salidas activas menos una, y así para todos los tipos de individuos.

En la práctica, generamos poblaciones auxiliares que contienen todos los individuos que haya en la población con todas sus salidas activas, y rellenamos dicha población auxiliar con el resto de individuos manteniendo las proporciones a partir del número de redes con todas activas. Esto implica que se pueden formar subpoblaciones de tamaño mayor al de la población original, pero será de forma temporal ya que las poblaciones de descendientes que se generan recuperan el tamaño original.

En resumen, hemos tenido que modificar el proceso de selección para que sea estadísticamente neutro, a base de mantener las proporciones globales de individuos diferentes en el conjunto de todas las subpoblaciones.

Volviendo al esquema de selección presentado, éste es idéntico al utilizado en el algoritmo VEGA [Schaffer, 85] desarrollado para problemas de optimización multiobjetivo. En este tipo de problemas, existen distintos objetivos en la solución que se deben satisfacer al mismo tiempo, por lo que en la población pueden coexistir distintas soluciones que combinadas cumplen el objetivo final [Coello, 02]. El principal inconveniente de VEGA en problemas multiobjetivo es que lleva a la formación de grupos especializados de individuos, que se han denominado *especies*. En la optimización multiobjetivo, las soluciones deben ser óptimas en el sentido de cubrir el *frente de Pareto*, es decir, deben ser soluciones globalmente buenas. Por este motivo, la formación de especies no es deseable. Pero en el PBGA, la situación se invierte y deseamos la formación de estos grupos de individuos independientes, ya que no buscamos soluciones en el *frente de Pareto*. La descomposición en distintas funciones que pretendemos llevar a cabo en el PBGA debe ser automática, es decir, el diseñador desconoce si dicha descomposición es posible y, de serlo, cuál es la forma óptima. Por tanto, nuestro problema no puede ser formulado como un problema de multiobjetivo porque desconocemos dichos objetivos múltiples por separado, es decir, no conocemos el *frente de Pareto*. Por este motivo, el esquema de selección presentado parece adecuado a nuestros intereses.

Todo el desarrollo llevado a cabo para este operador de selección, viene motivado por el deseo de simplificar los modelos de mundo al trabajar en entornos complejos a base de dividirlos en otros más simples. Es decir, estamos creando sistemas modulares, algo que es bastante común en la bibliografía del aprendizaje evolutivo [Darwen, 97] para reducir la complejidad. Como acabamos de comentar, en el PBGA los módulos se

forman con la aparición de especies de individuos en la población. Hasta el momento, la formación de especies se realiza por salida activa, pero hemos desarrollado una metodología más general para que no tenga por qué ser así y que ha derivado en una versión más general del PBGA, el GPBGA, que será explicado con detalle en el apartado 6.3. Su principal característica es la formación automática (en función de un parámetro regulador denominado afinidad) de grupos de individuos que colaboran para alcanzar la solución. Es decir, hemos creado un sistema modular automático para la obtención de soluciones complejas a base de combinar soluciones más simples. Por tanto, que la formación de especies dependa del número de salidas activas es un caso particular del esquema de selección por grupos que hemos desarrollado y será utilizado en el apartado actual.

A continuación, pasamos a estudiar con detalle los operadores de reproducción del PBGA, que deben tener en cuenta la existencia de los genes promotores en los cromosomas.

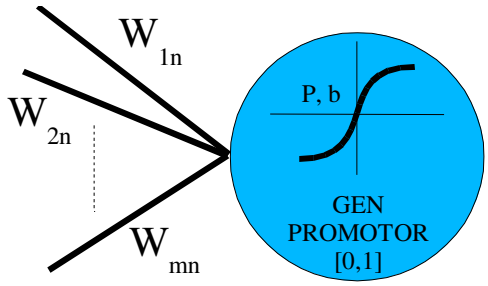
Reproducción

En cada una de las subpoblaciones generadas tras la aplicación del operador de selección, el PBGA mantiene individuos similares funcionalmente hablando y, de acuerdo con el apartado anterior, similares estructuralmente. Estos individuos son, por tanto, susceptibles de dar lugar a nuevos individuos que mejoren los resultados como en cualquier algoritmo evolutivo, es decir, son aptos para la aplicación de los operadores de reproducción. Al igual que en el algoritmo genético aplicado en las primeras versiones del MDB, estos operadores son el cruce y la mutación de individuos.

Cruce

La presencia de los genes promotores en los cromosomas, dificulta la acción del operador de cruce porque deben ser tratados de forma muy diferente a los pesos sinápticos y al resto de parámetros de dicho cromosoma. Y es que, por ejemplo, el cruce simple por uno o dos puntos de una de estas redes, no conserva información topológica sobre el orden de las neuronas que están activas o inactivas. Es decir, en el PBGA existe una fuerte dependencia de la posición de una neurona y de su actividad, y una vez que se alcanza una configuración de neuronas activas e inactivas para realizar una función, el mezclar redes con diferentes topologías no conduce a la convergencia.

Por este motivo, hemos decidido utilizar cada neurona de cada individuo como una unidad funcional con cierto grado de independencia. Estas unidades funcionales están compuestas por el gen promotor de la neurona, por los parámetros de la función de activación la neurona y por los pesos sinápticos que llegan a la neurona, como se muestra en el siguiente esquema:



Por tanto, el cruce de dos individuos seleccionados por torneo en cada una de las subpoblaciones, se produce neurona por neurona y de forma ordenada. Así, solo se puede cruzar la primera neurona de la primera capa oculta del primer individuo con la primera neurona de la primera capa oculta del segundo individuo y así sucesivamente. Pero este tipo de cruce provoca inestabilidad en los descendientes derivados de un cruce excesivo, y es que la información relevante de los padres no se transmite nunca sin modificar. Para contrarrestar este exceso de cruce, utilizamos una probabilidad de cruce por capa, que permite que todas las neuronas de una misma capa de uno de los padres pasen al hijo sin modificar. De este modo se consigue que información relevante permanezca sin cambio. Además de la inclusión de la probabilidad de cruce por capa, este operador tiene una serie de características particulares que son las siguientes:

- Utiliza reproducción panmítica, es decir, de dos individuos padre se forma un único descendiente con objeto de dar más relevancia a la función que llevan a cabo los genes promotores en los padres y que se hereda.
- Si ambos padres tienen el gen promotor de una cierta neurona activo, el hijo también lo tendrá activo. Si ambos padres lo tienen inactivo, el hijo también lo tendrá inactivo. En caso de que un padre lo tenga activo y el otro no, existe un 50% de probabilidad de que el hijo lo tenga activo o inactivo.
- Utilizamos una probabilidad de cruce general como en un algoritmo clásico que permite que un individuo seleccionado pase a la siguiente generación sin modificar. Al utilizar reproducción panmítica, sólo uno de lo padres puede pasar, y asignamos a ambos la misma probabilidad de hacerlo.
- Utilizamos, además, una probabilidad de cruce por capa que permite que todas las neuronas de una capa sean heredadas.
- Cuando no hay cruce por capa, lo hay por neurona, y en este caso los pesos y parámetros de la función de activación de los dos padres se cruzan. De nuevo, con objeto de conservar información útil sin modificar, existe un 25% de probabilidad de que los pesos y parámetros del hijo sean idénticos a uno de los padres, un 25% de que sean idénticos a los del otro padre y un 50% de probabilidad de que sean mezcla de ambos padres. Para ello, generamos un punto de cruce y unimos ambos cromosomas por dicho punto de forma usual.

El operador de cruce desarrollado funciona en primer lugar a nivel de individuo, ya que la probabilidad de cruce permite la conservación de individuos intactos. En segundo lugar funciona a nivel de capa, permitiendo que los descendientes guarden mucha información de los padres sin cambio. En tercer lugar funciona a nivel de unidad funcional (de neurona). En esta etapa, las neuronas de la capa de entrada simplemente cruzan sus genes promotores decidiendo si los correspondientes del descendiente estarán o no activos de acuerdo con la regla establecida anteriormente. Las neuronas del resto de capas incluida la de salida, cruzan sus genes promotores de igual modo y además cruzan los parámetros y pesos.

La evolución a nivel de neurona no es nueva en el campo del aprendizaje evolutivo ni en el aprendizaje por refuerzo y, por ejemplo en [Moriarty, 95] es aplicado en el algoritmo SANE. Estos autores utilizan dos poblaciones en evolución constante, una formada por neuronas, que especifican con qué otras neuronas se conectan y con qué valores sinápticos, y la otra por “proyectos” de redes, que especifican cuales de las anteriores neuronas se utilizan para construir la red final. Utilizan por tanto, un

algoritmo que explota la búsqueda a nivel neuronal en combinación con otra búsqueda a nivel topológico, obteniendo así un grado de libertad más que en el PBGA.

Una característica muy importante del operador de cruce, es que no hace distinción de un peso o parámetro que pertenece a una neurona activa o inactiva, es decir, la información que no se muestra también evoluciona. Al no participar en el fenotipo, esta información "oculta" no contribuye a la calidad del individuo y, por lo tanto, a la selección o no del mismo.

A nivel de activación, hemos querido que el hijo herede la funcionalidad de los padres si ésta es coincidente. Pero el cruce paramétrico tiene lugar independientemente del valor de la activación, de modo que neuronas inactivas también intercambian información, incluso con neuronas activas. Con esta propiedad, buscamos que una red pueda guardar información a nivel de gen, es decir, que puedan existir neuronas inactivas que cuando se activan den lugar a una nueva función de salida. Esta característica será muy útil como veremos más adelante, ya que nos va a permitir que, tras cambios en las funciones a obtener (en un entorno dinámico), las redes de la población sean capaces de mantener información anteriormente aprendida en neuronas inactivas. Por tanto, nos interesa que esa información que está presente en el cromosoma aunque no se muestre, también entre en los procesos de cruce para mantener así una especie de memoria genética. Además, una neurona puede pasar de estar inactiva a activa con facilidad por lo que las parámetros asociados a ella deben guardar cierta funcionalidad.

Tras aplicar a todos los individuos de una subpoblación el operador de cruce explicado, pasamos a aplicar el operador de mutación como en la mayoría de los algoritmos genéticos puros. De nuevo habrá que tener en cuenta las especiales características y funciones de los genes promotores a la hora de mutar.

Mutación

Al igual que en el algoritmo genético utilizado en las primeras versiones del MDB, el operador de mutación que aplicamos sobre los pesos y parámetros de la función de activación, se basa en añadir al valor del gen un número aleatorio entre -1 y 1 elevado al cubo, de modo que se tiende a explorar con mayor probabilidad la zona cercana al gen. Este operador no puede ser aplicado sobre los genes promotores, que son valores binarios. Además, la probabilidad de mutación usual en un algoritmo genético (en torno a un 1%) puede resultar muy grande al aplicarla sobre los genes promotores, ya que los cambios en éstos tienen consecuencias notables sobre la red obtenida.

Concretando, el operador de mutación desarrollado para el PBGA tiene las siguientes características fundamentales:

- Realizamos una mutación de tipo cubo para los pesos y parámetros de la función de activación de las neuronas.
- Los genes promotores se representan mediante valores binarios, por lo que la mutación simplemente cambia el estado: de 0 a 1 ó de 1 a 0.
- Utilizamos una probabilidad de mutación paramétrica que controla la mutación de los pesos y parámetros de las neuronas. Esta probabilidad es la misma que se utiliza en un algoritmo genético clásico.

- Además utilizamos una probabilidad de mutación estructural que controla la mutación de los genes promotores. Esta probabilidad debe ser menor que la anterior ya que provoca cambios muy importantes en la funcionalidad de los individuos. Por ejemplo, la activación de una cierta neurona implica que todas las conexiones asociadas también se activen, mientras que una mutación en un peso o parámetro tiene menos repercusión en la función final que se obtiene.

De nuevo, este operador se aplica sobre los parámetros asociados a neuronas que pueden estar inactivas. De hecho, no existe ninguna distinción en cuanto a la activación de la neurona y su probabilidad de mutación paramétrica. Al igual que en el caso del operador de cruce, esto nos permite que todos los genes sean fuente de información genética. Al aplicar el operador de cruce, los parámetros de una neurona inactiva pueden pasar a formar parte de una neurona activa, por lo que nos interesa que hayan intervenido en el proceso evolutivo y puedan aportar información útil.

Existen en la bibliografía de los algoritmos evolutivos distintos autores que han diseñado algoritmos para evolucionar redes neuronales basándose en parámetros que controlen la activación de distintas partes de dichas redes. Por ejemplo, el Dr. Dasgupta de la Universidad de Memphis, desarrolló en 1993 [Dasgupta, 93] un algoritmo genético que utiliza un fundamento similar al PBGA, y que denominó *algoritmo genético estructurado* (sGA). En este algoritmo, cada cromosoma viene representado por un conjunto de subcadenas de *genes de bajo nivel* controladas por *genes de alto nivel* que hacen la función de un activar o desactivar dichas subcadenas (que equivalen a las neuronas, con sus parámetros y pesos sinápticos, en el PBGA).

Existen, al igual que en el PBGA, genes activos y genes pasivos que permanecen siempre en el cromosoma manteniendo así múltiples soluciones en un mismo individuo. Durante el proceso de reproducción, los genes de bajo nivel son modificados por los operadores genéticos de forma usual. En la decodificación al genotipo, únicamente los genes que se encuentran activos en el fenotipo contribuyen a la calidad del algoritmo genético.

Las principales diferencias entre el sGA y el PBGA residen en el tipo de representación utilizada para los genes, ya que mientras que en el PBGA los individuos son redes neuronales y, por tanto, los genes representan los parámetros de dichas redes, en el sGA los genes son valores en binario.

Por tanto, estos dos algoritmos son conceptualmente similares en sus fundamentos teóricos, tratando de simular las características de los cromosomas biológicos reales donde existen genes pasivos y genes activos. El sGA utiliza un operador de selección clásico, por lo que no es posible la formación automática de soluciones implementada en el PBGA.

Es algo común en el campo de la programación genética, que a la hora de evolucionar redes neuronales, se utilice una estructura basada en nodos y terminales, de modo que se lleva a cabo una evolución de la topología de las redes y de los parámetros por separado obteniendo así redes de longitud variable. Por ejemplo, en [Pujol, 98] los autores desarrollan un algoritmo basado en Programación Genética Paralela Distribuida (PDGP) para la obtención automática de redes recurrentes donde, de modo similar al PBGA, existe un parámetro que controla la activación de cada nodo.

Tras haber presentado las principales características estructurales y funcionales del PBGA, pasamos a analizar con detalle su comportamiento práctico. Pero antes de nada,

debemos realizar una serie de matizaciones sobre la aplicación real del PBGA, es decir, sobre cómo ha sido implementado.

Aplicación práctica del PBGA

Aunque el concepto teórico del PBGA no tiene limitaciones en cuanto al tamaño de las redes que se pueden obtener, de cara a realizar una implementación práctica en este experimento, se han tomado una serie de decisiones de diseño que afectan al tamaño máximo y mínimo permitidos y a qué neuronas de entrada y salida pueden estar activas o inactivas. Así, las principales características de la implementación práctica del PBGA son las siguientes:

- El número mínimo de capas ocultas que puede tener cualquier red obtenida es de dos.
- En cada capa oculta hay un límite en cuanto al número máximo de neuronas que existen y, por tanto, que pueden estar activas y pasar al fenotipo. Este límite depende de la complejidad de la función a aproximar.
- En cada capa oculta debe haber al menos una neurona activa.
- El número de neuronas de entrada y salida viene impuesto por el problema.
- Las neuronas de entrada están siempre activas, de modo que todas las variables de entrada son conocidas por todos los individuos de la población.
- Las neuronas de salida pueden estar o no activas, pero como mínimo una debe estarlo.
- Utilizamos, en estos experimentos, redes neuronales artificiales de tipo perceptrón multicapa.

Con estas restricciones, la red más pequeña que se puede obtener tendrá el número de entradas del problemas en concreto, una primera capa oculta con una única neurona, una segunda capa oculta con una única neurona y una sola neurona de salida. Por otra parte, la red más grande que se puede obtener tendrá el número de neuronas de entrada del problema, la primera y segunda capa oculta con tantas neuronas como hayamos impuesto como límite superior y las neuronas de salida del problema.

El hecho de utilizar siempre dos capas ocultas, permite obtener funciones no lineales más complejas cuanto mayor sea el número de neuronas activas y, por otro lado, permite obtener funciones muy simples con una única neurona en cada capa. De modo que, en el PBGA, la única variable en cuanto al tamaño de las redes que debemos fijar es el número máximo de neuronas en las capas intermedias. Este número no es conocido con exactitud al tratar de resolver un determinado problema, pero sí podemos conocer una cota superior (impuesta, por ejemplo, mediante consideraciones de tiempo de cómputo) y dejar que el algoritmo obtenga el número de neuronas que necesita para aproximar la función. Es decir, el tamaño máximo de las redes es un problema cuando restringe grados de libertad al PBGA, de otra forma el algoritmo obtendrá siempre redes menores.

Las pruebas realizadas permitiendo que los genes promotores de las neuronas de entrada estén activos o inactivos como en el resto de neuronas no fueron satisfactorias. El principal problema reside en que el número de grados de libertad se vuelve excesivo y el algoritmo tiende a obtener redes con el menor número de neuronas de entrada

activas posible. Esto implica que se utilizan menos variables de entrada de las necesarias para una determinada salida y la función objetivo se predice con gran error. En las primeras iteraciones, el algoritmo tiende a obtener redes simples que, aunque proporcionan un error grande, condicionan la evolución porque se desfavorece la aparición de redes más complejas (con mayor número de entradas) ya que su ajuste es más costoso. Como solución a este problema, hemos decidido que todas las redes tengan las variables de entrada siempre presentes, de modo que se facilita la generación de funciones independientes al tener toda la información de entrada disponible. El PBGA no pierde así generalidad, porque el objetivo principal de dividir de forma automática funciones complejas en partes más simples no se ve afectado por el hecho de que todas las neuronas de entrada estén activas. Lo realmente importante a la hora de crear funciones independientes es el número de neuronas de salida activas, y este grado de libertad se conserva. Si una determinada red de la población se especializa en una parte de la función, tendrá únicamente ciertas neuronas de salida activas y si no requiere alguna variable de entrada para dicha función, simplemente adaptará sus pesos sinápticos a cero. Esto es usual en un algoritmo de aprendizaje evolutivo.

Expuestas estas condiciones iniciales sobre la implementación práctica del PBGA, pasamos a estudiar en los siguientes apartados, la información que almacenan los genes inactivos presentes en los cromosomas y cómo podemos utilizar dicha información en problemas de optimización reales. Además aplicaremos el PBGA a la obtención de un modelo de mundo a partir de datos obtenidos con robots reales.

Información genética derivada del uso de genes promotores

Tanto el operador de cruce como el operador de mutación presentados con anterioridad, favorecen la permanencia de información relevante en todos los parámetros de las redes aunque pertenezcan a neuronas inactivas. Como hemos explicado, en el PBGA no existe diferencia entre los parámetros de una neurona u otra, con objeto de que en el momento en que se necesiten estos parámetros, contengan información relevante, es decir, que hayan participado en el proceso evolutivo.

Para comprobar esta afirmación, compararemos el PBGA con un algoritmo genético que utiliza cromosomas de longitud variable (VLCGA) de forma convencional. En este tipo de algoritmo, la transformación de genotipo a fenotipo es directa, y siempre se trabaja con el fenotipo del individuo. El operador de cruce nunca puede dar lugar a individuos con neuronas nuevas, ya que simplemente mezcla información de los padres, pero no la crea. El operador de mutación es el responsable de la aparición y desaparición de neuronas y, como sabemos, de modificar los parámetros de las existentes. De hecho, la principal diferencia entre este algoritmo y el PBGA reside precisamente en la creación de neuronas nuevas, ya que implica la creación de conexiones nuevas y el establecimiento de los pesos y parámetros mediante algún criterio. Es usual en este tipo de algoritmo que los parámetros de la nuevas neuronas se creen de forma aleatoria o bien sean combinación de los parámetros de las neuronas vecinas. Nosotros hemos optado por la primera opción de modo que, cada vez que el operador de mutación introduce una nueva neurona, los pesos y parámetros asociados se crean aleatoriamente.

Utilizamos, por tanto, el PBGA con el objetivo de modelar una función teórica simple durante un número fijo de generaciones de evolución. Tras este periodo, cambiamos la función por otra mucho más compleja y volvemos a evolucionar durante el mismo número fijo de generaciones. Cuando cambiamos la función a modelar

utilizamos la población existente de la evolución a la función anterior, de modo que va a estar muy especializada. Esta misma metodología será aplicada al algoritmo clásico VLCGA, buscando que los resultados obtenidos por una y otra aproximación puedan ser comparables. No debemos olvidar la razón que nos ha llevado al desarrollo de un nuevo algoritmo, esto es, la complejidad del aprendizaje de ciertos modelos de mundo en el MDB. Por este motivo, las pruebas de este apartado serán realizadas con funciones que simulan las características de los modelos de mundo. Estas funciones son las siguientes:

$$\text{Función suma 3D:} \quad z = x + y$$

$$\text{Función senoidal 3D:} \quad z = x \cdot sen(x) + y \cdot sen(y)$$

Para comparar ambos algoritmos en igualdad de condiciones, realizamos un muestreo de ambas funciones de tal forma que los puntos que utiliza la evolución con el PBGA son exactamente los mismos que utiliza el VLCGA. Como vemos, las funciones constan de 2 variables independientes y una dependiente, de modo que las redes a obtener poseen 2 neuronas de entrada y una de salida. En este apartado, nos interesa estudiar el PBGA como generador de funciones simples, por lo que utilizamos funciones de una única salida. Los genes promotores que pueden cambiar, por tanto, son los asociados a las neuronas de las dos capas ocultas, mientras que el número de neuronas de entrada y salida está fijado.

Por ahora nos centramos en el estudio del PBGA como algoritmo, no en su interacción y cooperación con otras partes del MDB. Por tanto, nos interesa estudiar el PBGA como aproximador de una cierta función objetivo. Como hemos introducido al hablar del aprendizaje en el mecanismo, esta función vendrá dada por muestras que son almacenadas en la Memoria a Corto Plazo y que hemos de gestionar adecuadamente. Esta tarea es totalmente independiente del algoritmo utilizado para aproximar la función, por lo que no hemos utilizado ningún tipo de estrategia de reemplazo es este estudio. Lo que hemos hecho ha sido llenar la Memoria a Corto Plazo (MCP) con un número de muestras fijo (42 exactamente) de las dos funciones anteriores y mantenerlas durante todo el proceso evolutivo sin reemplazo. Desde el punto de vista de las muestras almacenadas en la MCP, éstas contienen para un mismo valor de las variables de entrada x e y, dos valores de la variable de salida z, pero sólo se utiliza una de cada vez. Esta estrategia es más eficiente que utilizar dos memorias distintas o que rellenar la MCP cada vez que cambiamos la función objetivo. El único problema que plantea es que el muestreo de las variables de entrada debe ser idéntico para ambas funciones. Este planteamiento no permite un aprendizaje de las funciones en todo el dominio porque las muestras utilizadas no son suficientes para dicha tarea, pero sí nos sirve para probar el PBGA con dos conjuntos de muestras diferentes, uno más simple y otro más complejo.

En la figura 4 mostramos la evolución del error cuadrático medio entre la predicción que realiza la mejor red de la población y el valor teórico real para todas las muestras de la MCP. Como ya hemos explicado, tras un número fijo de generaciones utilizando como objetivo la función suma 3D, pasamos a intentar aproximar la senoidal 3D. Cuanto mayor sea este número de generaciones, mayor será la especialización de las poblaciones y más costoso será el aprendizaje de la nueva función. Además, es de esperar que, tras haber evolucionado usando la función suma 3D, el cambio hacia la senoidal 3D sea todavía más difícil porque el aprendizaje de la función más simple se logra antes, con lo que la población habrá convergido más y, por lo tanto, casi todos los individuos serán iguales. Las gráficas de la figura 4 se obtuvieron usando 20 generaciones de evolución entre cambios de la función objetivo y además corresponden

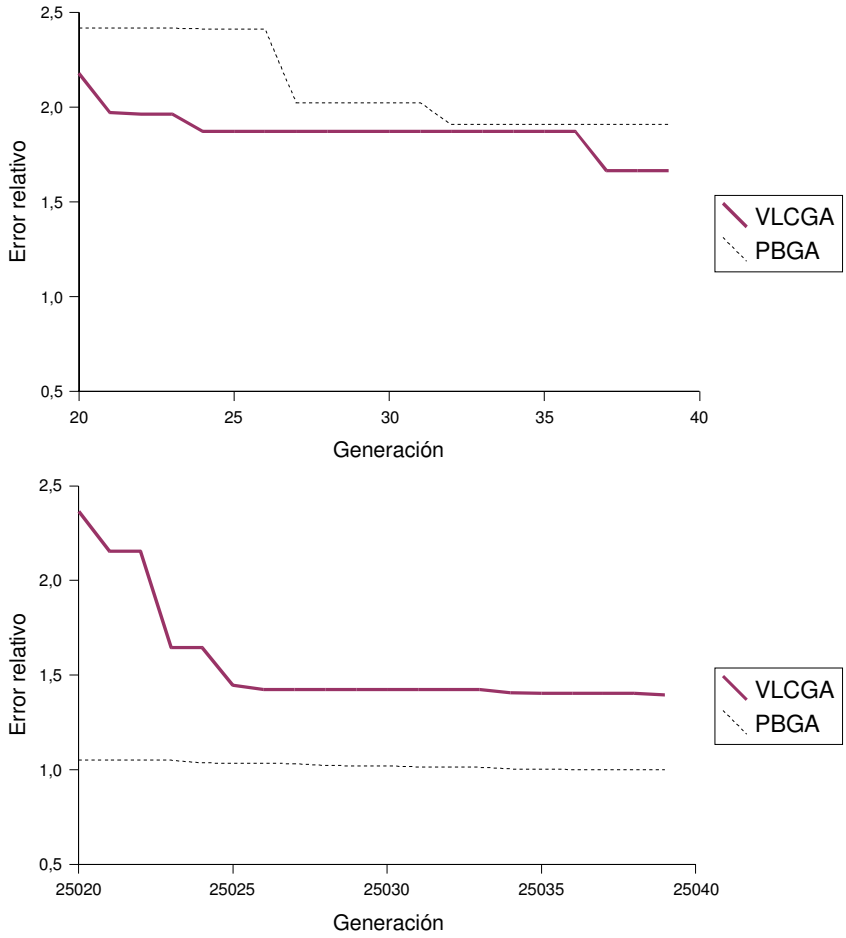


Figura 4. Evolución del error cuadrático medio (normalizado al menor valor) entre la predicción que realiza la mejor red de la población y el valor real para todas las muestras de la MCP durante 20 generaciones de evolución. Ambas gráficas corresponden al modelado de la función senoidal 3D. La gráfica superior fue obtenida en las primeras etapas de la prueba (generaciones 20 a 40) mientras que la figura inferior se obtuvo 25000 generaciones más tarde.

al error cometido con la función senoidal 3D, es decir, comienzan tras un periodo de aprendizaje con la función suma 3D que es un caso bastante desfavorable. No estamos interesados por ahora en la precisión de las redes obtenidas con el PBGA, por lo que en la figura 4 hemos normalizado todos los resultados al mejor valor de error obtenido, que puede dar lugar a un modelado pobre. De hecho, para la función suma 3D, el modelado resultó muy bueno, mientras que para la función senoidal 3D, el número de muestras utilizado y el carácter totalmente estático de las mismas no fue suficiente para un modelado satisfactorio. Lo que sí obtuvimos fue una clara mejora del error en ambos casos, que es lo que resulta relevante para el actual estudio.

La gráfica superior de la figura 4 corresponde al intervalo comprendido entre las generaciones de evolución 20 y 40, que es la primera vez que se usa la función senoidal 3D como objetivo. Es decir, la evolución comienza con la función suma 3D como modelo a obtener, en la generación 20 se cambia por la senoidal 3D y estos son los

datos que mostramos en la gráfica. Como vemos, el nivel de error obtenido es similar en ambos casos y grande si lo comparamos con el mejor que llegamos a obtener de valor 1. El PBGA requiere más de 30 generaciones para alcanzar un nivel de error estable en torno a 1.6, mientras que el VLCGA no deja de evolucionar en todo el intervalo alcanzando incluso niveles de error más bajos que el PBGA.

En la gráfica inferior se observa una tendencia muy diferente y, aunque el nivel de error alcanzado por ambos algoritmos es menor, en el caso del PBGA la mejora es muy notable. Esta gráfica se obtuvo con los datos de error correspondientes al intervalo comprendido entre 25020 y 25040, es decir, 25000 generaciones de evolución después. Tras este periodo tan grande de tiempo de aprendizaje, el VLCGA presenta un comportamiento muy similar a las primeras generaciones y, aunque existe una cierta mejora en el error, éste cambia de nuevo durante todo el intervalo. En cambio, el PBGA llega a un valor de error mucho menor desde el principio y prácticamente no evoluciona. Este resultado confirma en principio nuestras suposiciones sobre la información que es almacenada en los genes que no se muestran, ya que no es posible que tras 20 generaciones de evolución con la función suma 3D, el algoritmo alcance tan pronto la convergencia si no es porque la configuración de neuronas adecuada ya está presente en el genotipo.

De modo que, los individuos del PBGA acaban por incluir dos configuraciones topológicas de neuronas en sus cromosomas y, ante un cambio de función, simplemente conmutan entre una y otra. Las neuronas que pueden ser útiles para aproximar la función suma 3D, no tienen por qué serlo para aproximar la senoidal 3D, pero el cambio de una red a otra no es tal, si no que simplemente es necesario que se activen neuronas que permanecían con su gen promotor deshabilitado. Los operadores de cruce y mutación que estamos usando favorecen que los parámetros de las distintas neuronas no desaparezcan, y sigan presentes en la población tras 20 generaciones de evolución.

En el caso del VLCGA, cada vez que se cambia la función objetivo las redes de la población deben cambiar y adaptarse a las nuevas condiciones. Para ello, se deben crear o eliminar neuronas y las correspondientes conexiones neuronales cada vez. Es decir, las funciones deben ser reaprendidas siempre que se cambian, como confirman los resultados de evolución del error de la figura 4. En ambas gráficas, el VLCGA empieza desde un valor de error similar debido a que la población está muy convergida hacia la función suma 3D, y este algoritmo no tiene forma de retener información genética. Al partir de ese valor de error y de la población anterior, no alcanzará nunca resultados mejores que los mostrados en la gráfica inferior. En el PBGA, una vez aprendidas ambas configuraciones, la población contiene información genética de ambas funciones y al cambiar de una a la otra, no hay prácticamente transición ya que en el primer proceso de reproducción aparecen individuos buenos con facilidad.

Tras lo explicado hasta este punto, nos podemos preguntar qué ocurre si utilizamos un número más grande de generaciones entre cambios de la función objetivo. La evolución se alarga en el tiempo, por lo que las poblaciones estarán todavía más especializadas a las funciones de cada intervalo. ¿Será el PBGA capaz ahora de mantener información sobre las dos configuraciones?. La respuesta está en la figura 5, que es análoga a la 4 anterior pero usando 100 generaciones de evolución con cada función, y nos muestra cómo el PBGA aún es capaz de mantener información genética relevante.

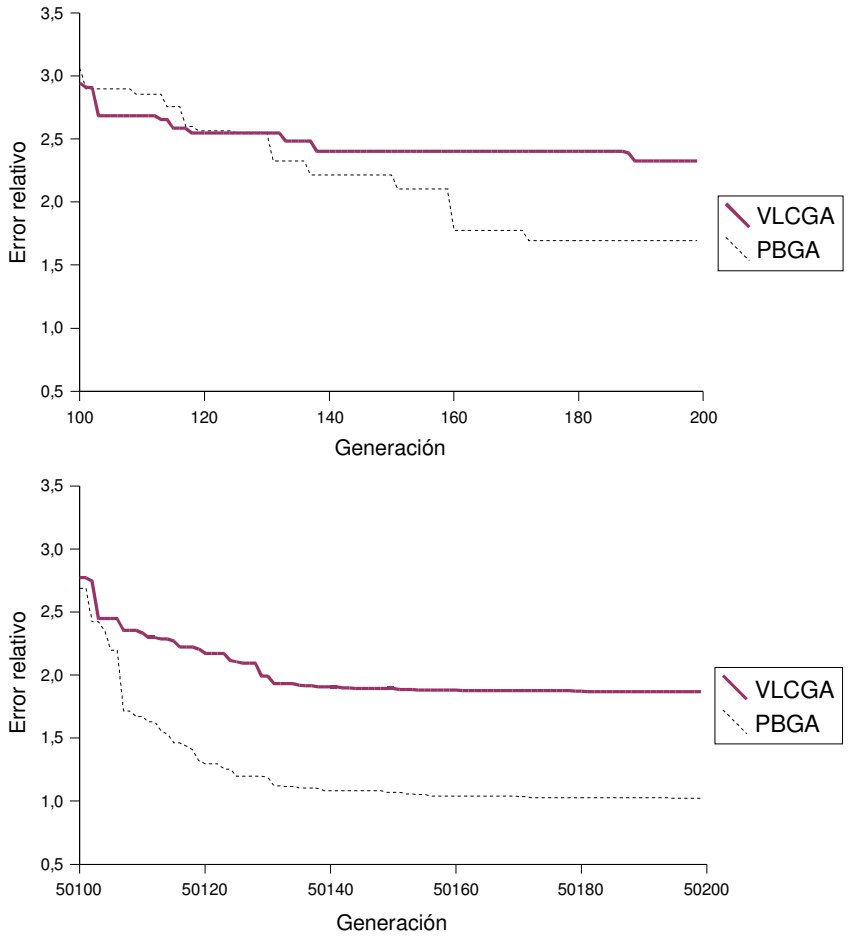


Figura 5. Evolución del error cuadrático medio (normalizado al menor valor) entre la predicción que realiza la mejor red de la población y el valor real para todas las muestras de la MCP durante 100 generaciones de evolución. Ambas gráficas corresponden al modelado de la función senoidal 3D. La gráfica superior fue obtenida en las primeras etapas de la prueba (generaciones 100 a 200) mientras que la figura inferior se obtuvo 50000 generaciones más tarde.

En la gráfica superior, en la primeras generaciones que utilizan la función senoidal 3D mencionada, de nuevo el comportamiento es muy similar en ambos algoritmos, y la mejora del error se prolonga durante las 100 generaciones sin estabilizarse. En la gráfica inferior, obtenida tras un periodo muy grande de aprendizaje (50000 generaciones), el PBGA se estabiliza antes que el VLCGA aunque, como era de esperar, los resultados se igualan. Ahora, ambos algoritmos deben aprender desde situaciones muy similares: poblaciones demasiado especializadas. Es decir, a medida que el número de generaciones entre cambios de la función objetivo aumenta, los tiempos de evolución (tiempo de reaprendizaje) tienden a ser más parecidos.

En la evolución representada en la figura 4 y que utiliza 20 generaciones por cambio de función objetivo, se puede plantear la cuestión de si realmente el PBGA conmuta entre dos configuraciones o bien si simplemente es muy poco tiempo de evolución y las redes que eran buenas para la función usada 20 generaciones antes, aún siguen en la

población. Con objeto de analizar lo que ocurre en realidad en el interior de las poblaciones de redes, utilizamos la evolución anterior con 100 generaciones por cambio de la función objetivo, de modo que es muy poco probable que aún permanezcan redes buenas para la función anterior. En la gráfica superior de la figura 6 mostramos el número medio de neuronas activas para todas las redes de la población durante 1500 generaciones para el PBGA. Esto es, mostramos el número de neuronas de la configuración fenotípica. Al evolucionar usando la función suma 3D, el tamaño medio de las redes de la población es de 5 neuronas y usando la función senoidal 3D oscila entre 8 y 10. Esto nos indica, en primer lugar, que existe una transformación global de toda la población y, en segundo lugar, que realmente el PBGA requiere aprender dos redes muy diferentes para llevar a cabo los dos modelados.

Vemos, además, que el número de generaciones requerido para cambiar de una configuración a otra es muy pequeño (en torno a 10), lo que indica que aún en el caso de 100 generaciones, existe información remanente en los genes no mostrados de los

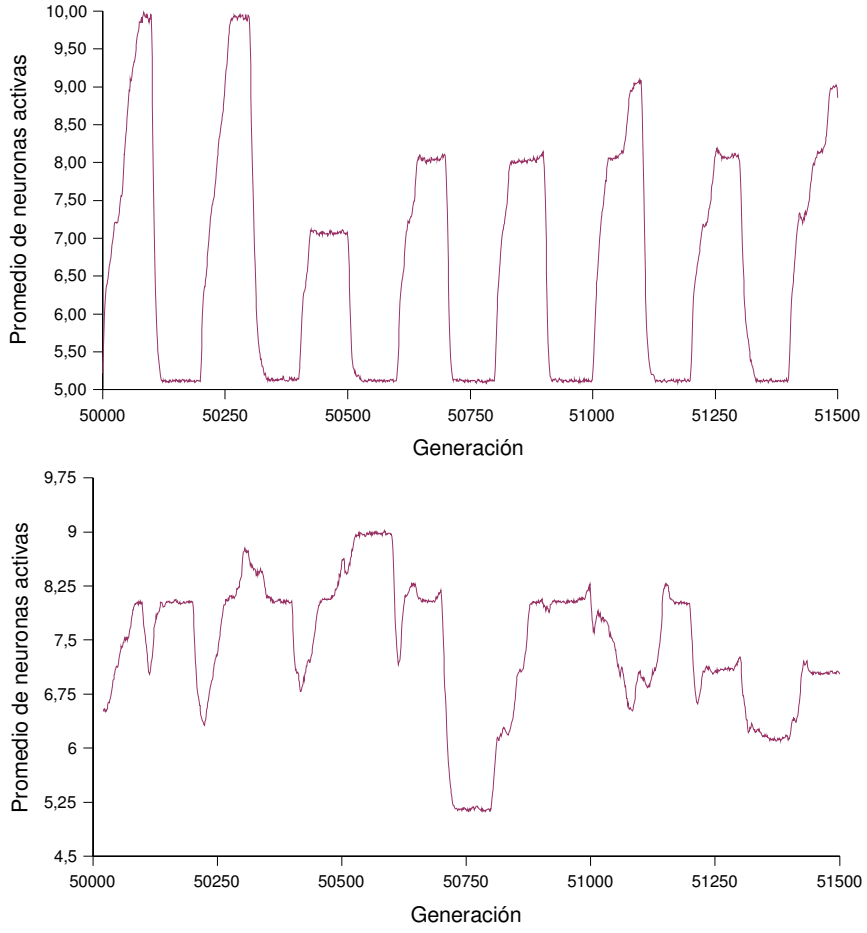


Figura 6. Número medio de neuronas activas en todas las redes de la población al conmutar la función objetivo cada 100 generaciones usando el PBGA. (gráfica superior) y usando el VLCGA (gráfica inferior).

individuos. De otra forma, el cambio del número medio de neuronas activas no podría ser tan rápido. Esto se constata al analizar la gráfica inferior de la figura 6, que corresponde de nuevo al número medio de neuronas activas en la población para la misma evolución con 100 generaciones, pero usando el algoritmo clásico VLCGA. Ahora las funciones deben ser reaprendidas cada vez y el cambio de tamaño promedio no es tan preciso como con el PBGA. De hecho, en la gráfica inferior de la figura 6 se observa un comportamiento de cambio de tamaño en las redes (subidas y bajadas del error) pero de forma gradual. En resumen, el PBGA cambia la estructura de todas las redes de la población en tan sólo 10 generaciones, mientras que a un algoritmo sin memoria genética (sin información en los genes) requiere muchas más generaciones para lograr ese cambio tan profundo.

Como conclusión a este apartado, podemos decir que el uso de genes promotores y de una transformación genotipo-fenotipo favorece la permanencia de información genética en los cromosomas de los individuos. Al igual que ocurre en cualquier proceso biológico real, si la evolución se realiza durante periodos muy grandes de tiempo sobre las mismos objetivos de aprendizaje, esta información genética acabará por perderse al no ser útil al individuo. En el PBGA, incluso tras 100 generaciones de evolución, gran parte de la memoria genética se conserva, lo que le permite adaptarse con rapidez a cambios en la función a aproximar si se le presenta repetidas veces.

En el siguiente apartado, comprobaremos que el nuevo algoritmo es capaz de obtener con precisión el tipo de funciones que aparecen en entornos reales al aplicar el MDB. Este objetivo es logrado a base de descomponer funciones complejas en partes más simples.

Test del PBGA con datos reales

Una de las aplicaciones reales del MDB que hemos llevado a cabo, utiliza el robot hexápodo Hermes II que será presentado en el apartado 7.1.1. En dicha aplicación (apartado 7.1.4), el robot aprende a girar hasta alcanzar un objeto que se sitúa en sus cercanías. El modelo de mundo utilizado se basa en la utilización del sensor virtual que se presenta en el apéndice A, y que proporciona la distancia y ángulo respecto al obstáculo más cercano al robot. Aunque estos datos son de más alto nivel que los que proporcionan directamente los sensores infrarrojos del Hermes II y, por tanto, con un nivel de ruido e incertidumbre menor, son datos obtenidos en tiempo real y muy dinámicos.

Con objeto de probar el PBGA con el tipo de datos que nos vamos a encontrar en los problemas reales, cambiamos las funciones teóricas del apartado anterior por los datos obtenidos en el ejemplo citado del Hermes II. Utilizamos 80 muestras obtenidas del conjunto total disponible en dicho ejemplo, que están formadas por 3 variables de entrada (distancia, ángulo y acción aplicada) y 2 de salida (distancia y ángulo). El resto de valores almacenados en los pares acción-percepción no son relevantes en esta prueba, ya que únicamente nos centramos en la capacidad generalizadora del PBGA. Por tanto, las redes de la población tienen 3 neuronas de entrada, dos capas ocultas de, al menos, una neurona activa y una o dos neuronas de salida activas. Al trabajar ahora con dos salidas, el PBGA puede proporcionar una o dos redes distintas como solución final. Como primera aproximación, no utilizamos algoritmo de reemplazo para estas 80 muestras, es decir, la MCP será de tamaño 80 y no cambiará nunca.

El resultado obtenido se muestra en la figura 7 y es satisfactorio. La línea continua representa la aproximación obtenida por el PBGA para cada salida, y los puntos representan los datos reales. En la gráfica superior, que corresponde a la predicción de la distancia al objeto, vemos cómo hay un punto espúreo (no olvidemos que son datos reales, y este tipo de punto fuera de rango es considerado como ruido) que la red obtenida no es capaz de aproximar. Pero lo más relevante de la figura no se muestra en ella, y es que el PBGA obtiene dos redes diferentes para cada una de las dos salidas, la primera (distancia) con la siguiente configuración:

Neuronas de entrada:	**1 1 1**
Neuronas de la primera capa oculta:	**0 0 1 0 1 1 1 1**
Neuronas de la segunda capa oculta:	**1 1 0 1 1 0 0 1**
Neuronas de salida:	**1 0**

Como vemos, las tres neuronas de entrada están activas tal y como habíamos impuesto, en la primera capa oculta hay 5 neuronas activas y en la segunda otras 5. En la capa de salida, únicamente la primera neurona está activa, como corresponde a una red seleccionada por dicha salida. Este es un esquema de la representación genotípica de la mejor de las redes obtenidas tras la evolución, que se corresponde con una red de tamaño 3-5-5-1 en el fenotipo. Hemos limitado el número máximo de neuronas activas en las capas ocultas a 8 en cada una y, como vemos, es un número suficiente ya que la solución final no necesita activar todas.

Para la otra salida (ángulo), la red obtenida puede ver también esquemáticamente:

Neuronas de entrada:	**1 1 1**
Neuronas de la primera capa oculta:	**0 1 0 1 1 1 1 1**
Neuronas de la segunda capa oculta:	**0 1 1 1 1 1 1 1**
Neuronas de salida:	**0 1**

Esta red es, estructuralmente, muy diferente de la anterior y su fenotipo da lugar a una red de tamaño 3-6-7-1, mayor que en el caso de la distancia. Por tanto, tal y como buscábamos al diseñar el PBGA, en la misma población coexisten y coevolucionan dos especies de redes totalmente diferentes que unidas proporcionan una solución satisfactoria al problema real.

Como última prueba de la capacidad de generalización del PBGA, decidimos utilizar otro conjunto de datos obtenido de un robot real, pero en este caso de un problema sensorialmente más complejo. Para ello, hemos utilizado el robot rodado Pioneer 2, que se presenta en el apartado 7.2, realizando una tarea de seguimiento de paredes. Los detalles de implementación de dicha tarea pueden ser consultados en [Becerra, 03].

Los sensores sónar que posee el robot tienen una gran cantidad de ruido, lo que dificulta mucho el modelado que realizan las redes. De hecho, el algoritmo genético

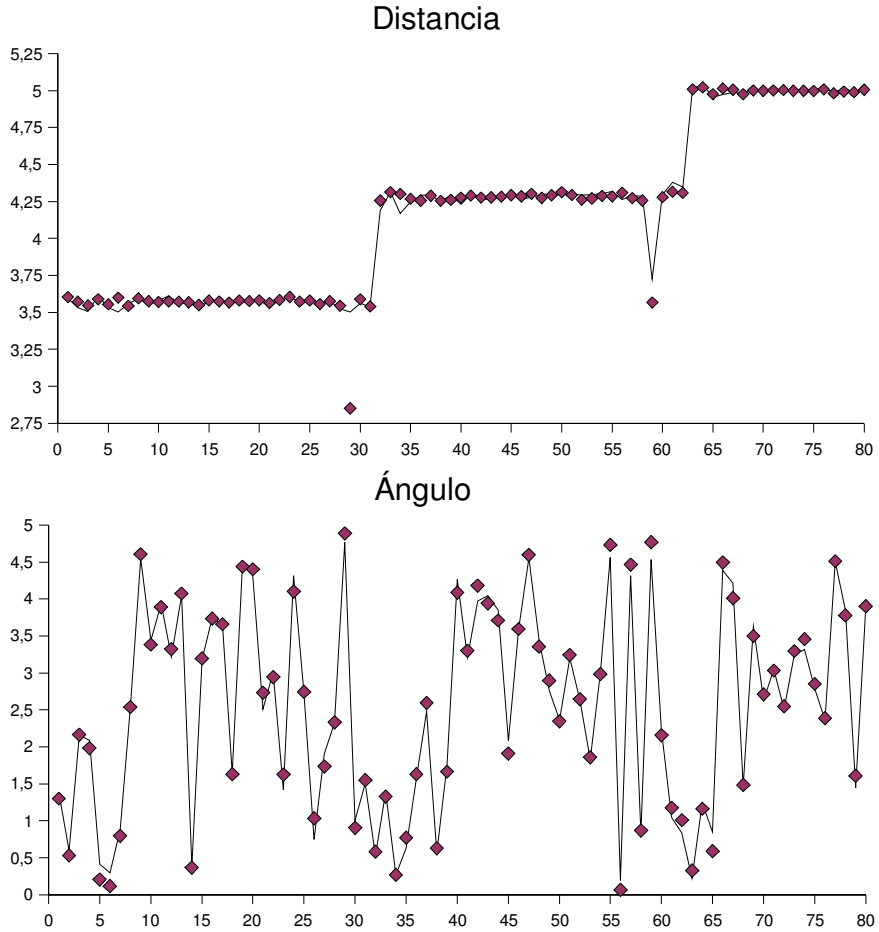


Figura 7. Predicción que realizan las redes obtenidas por el PBGA para las dos salidas, distancia en la gráfica superior y ángulo en la inferior. Los puntos representan los datos reales obtenidos con el Hermes II y las líneas continuas representan la predicción obtenida.

tradicional que utilizamos en las primeras versiones del MDB, no era capaz de obtener una red válida como modelo de mundo en las pruebas realizadas dado el enorme tamaño de la redes a obtener y la dificultad del problema. Como el objetivo final del MDB está a un nivel conceptualmente más alto que el simple modelado de entornos ruidosos, tampoco en este caso hemos utilizado los datos que proporcionan los sensores sónar directamente. Para la tarea a realizar, utilizamos únicamente los 8 sensores de uno de los laterales del robot (el lateral pegado a la pared a seguir) y de estos 8 sensores nos quedamos con las 4 medidas más relevantes (la media de cada dos). Esta simplificación permite trabajar con redes de menor número de entradas y salidas sin perder información (la tarea se lleva a cabo de igual modo con estos 4 sensores simples), pero aún sigue siendo un tamaño muy grande para los algoritmos genéticos clásicos. Así, las redes a obtener en este ejemplo tienen 6 neuronas de entradas (4 sensores y dos actuaciones sobre los motores de las ruedas), dos capas ocultas de, como mínimo, una neurona cada una y 4 neuronas de salida. En este ejemplo, hemos limitado el número

máximo de neuronas en las capas intermedias a 6 y, de nuevo, utilizamos un conjunto fijo de muestras, en este caso de 137, del total de puntos posible y no existe reemplazo en la MCP que almacena estas muestras.

En la figura 8 hemos representado el resultado de predicción de las 4 salidas que se ha obtenido y que, de nuevo, es satisfactorio. Las líneas continuas representan la predicción que realiza la mejor red para cada salida y los puntos representan los datos reales. El PBGA obtiene de forma automática 4 redes diferentes que predicen cada una de las salidas y que unidas nos permitirían realizar un modelo de mundo en el MDB con una precisión más que aceptable. Las 4 redes son totalmente diferentes y sus representaciones fenotípicas tienen unos tamaños de:

Primera salida: 6-4-6-1

Segunda salida: 6-3-4-1

Tercera salida: 6-6-3-1

Cuarta salida: 6-2-4-1

Cada una de estas redes afecta a una salida diferente, de modo que la solución final que proporciona todas las salidas se obtiene de la unión de estas cuatro.

Estos resultados confirman los obtenidos con los datos del Hermes II, y el PBGA es capaz de descomponer una función compleja en funciones más simples, cada una de las cuales realiza una tarea independiente. En la población de este último ejemplo, coexisten y coevolucionan 4 especies diferentes sin afectar en gran medida al tiempo de cómputo y logrando unos niveles de error en las predicciones muy bajos. La obtención de redes con más de una salida activa implica que dichas redes deben ser más complejas porque han de realizar dos o más funciones a la vez. De modo que, si el diseñador no lo fuerza o lo facilita, por ejemplo, en la función de calidad, el PBGA tenderá a obtener una red diferente por salida dado que son menos costosas de obtener y sus resultados son muy adecuados. De cara a la implementación del PBGA en el MDB al trabajar con modelos de mundo complejos (entornos complejos), no es relevante que dicho modelo de mundo esté formado por una única red o por varias, lo importante es que constituya una representación fidedigna del entorno. Tampoco se ralentiza la ejecución del modelo, ya que la obtención simple de las salidas de la red dadas unas entradas es inmediata y además la existencia de estos modelos es relevante en cuanto a que son más reutilizables.

Los resultados de este apartado confirman que el algoritmo PBGA cumple todos los objetivos con los que fue diseñado y permite obtener modelos complejos en partes más simples de forma automática, tanto en el número de ellos como en su estructura. Además, los modelos obtenidos resultan muy buenos en el modelado de la función global.

Damos así por terminado este apartado dedicado a la obtención de modelos y acciones y la inclusión de aprendizaje en el MDB. A continuación analizaremos con detalle otro de los aspectos fundamentales del mecanismo, la Memoria a Corto Plazo.

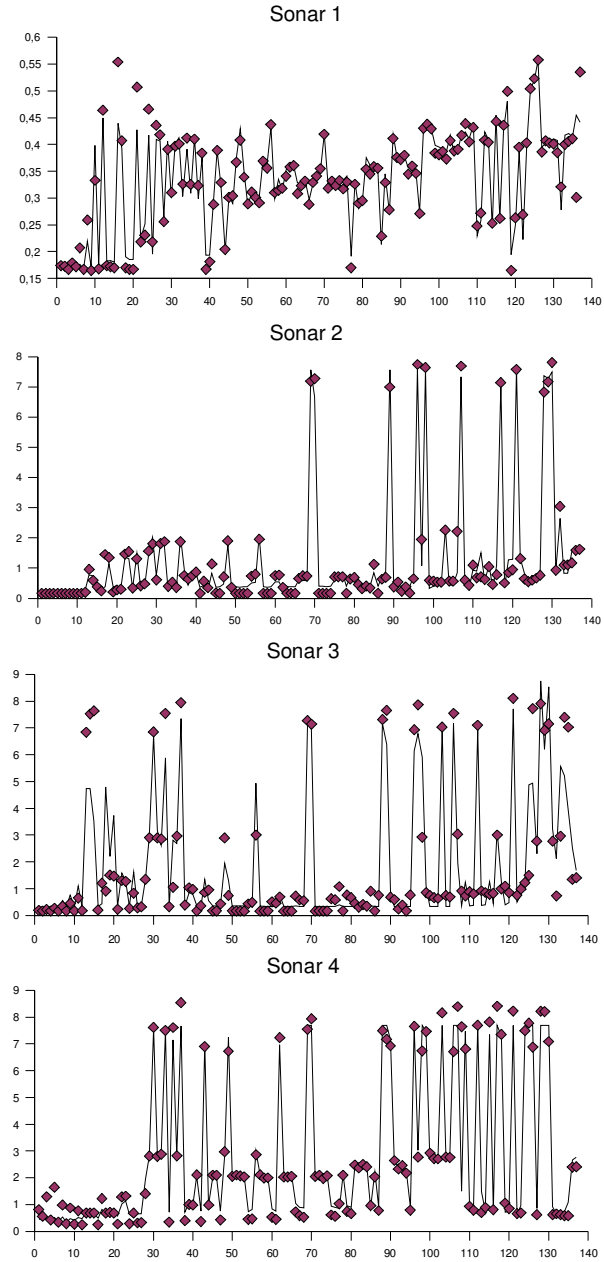


Figura 8. Predicción que realizan las redes obtenidas por el PBGA para las cuatro salidas, una por sónar. Los puntos representan los datos reales y las líneas continuas representan la predicción obtenida.

6.2.2 Memoria a Corto Plazo (MCP)

A la hora de trabajar sobre cualquier mecanismo cognitvo, un aspecto a tener especialmente en cuenta es la capacidad de recordar los acontecimientos pasados por el agente. Si en la vida de un ser se produce un cambio en su entorno cotidiano, éste debe ser capaz de adaptarse modificando sus acciones para seguir colmando sus necesidades. Para ello, su memoria debe contener experiencias pasadas que pueda utilizar como base para afrontar los cambios, es decir, en el caso del MDB, deben estar almacenados distintos modelos de mundo e internos.

Esta memoria se incluye en el mecanismo a través de la *Memoria a Largo Plazo* (MLP), y, en menor medida, a través de las poblaciones iniciales de los procesos evolutivos que no cambian de una iteración a la siguiente. A medida que el número de iteraciones del mecanismo crece, las poblaciones se van perfeccionando con el objetivo de conseguir una población globalmente buena. Además de estas tres bases de *modelos de mundo, modelos internos* y *estrategias*, es necesaria una memoria que guarde los pares acción-percepción que llamaremos M*emoria a Corto Plazo* (MCP). Es necesaria para recordar las consecuencias de nuestras acciones recientes y nos dice qué relación existe en el mundo real entre nuestras percepciones antes y después de llevar a cabo una estrategia.

Los modelos de mundo e internos evolucionan, mejoran, intentando generalizar el comportamiento mostrado en los pares acción-percepción que se guardan en la Memoria a Corto Plazo. Esta memoria no puede guardar todos los pares que se presentan en la vida de un agente ya que, en general, sería una tarea inabordable, pero debe ser suficientemente grande como para poder recordar situaciones relevantes del pasado reciente. En este sentido, el número de pares acción-percepción que se almacenan debe ser suficiente para que las estrategias sean tales, no simples acciones, y el proceso de planificación tenga sentido (al escoger una estrategia debemos tener en cuenta la información reciente), pero no debe ser demasiado grande para evitar que las estrategias que escogemos tengan que generalizar demasiadas posibles situaciones (al escoger una estrategia no podemos tener en cuenta todo nuestro pasado).

Por tanto, el mecanismo utiliza una memoria que guarda los pares acción-percepción pasados para el agente. Internamente deseamos generar modelos que se comporten exactamente como estos pares, para poder así “razonar” y evitar probar acciones aleatoriamente. En el siguiente apartado analizamos con detalle la Memoria a Corto Plazo y desarrollamos una estrategia de reemplazo para la misma.

Gestión de la Memoria a Corto Plazo

Una de las características más importantes del MDB, es que se realiza un modelado en tiempo real del mundo (entendido como un concepto más general que el entorno y que puede incluir entradas sensoriales propias del agente) y del estado interno a partir de la información obtenida tras las etapas de actuación y sensorización. En este apartado, trataremos de establecer procedimientos adaptativos para realizar una extracción de la información más relevante compatible con las limitaciones de almacenamiento de un mecanismo, que por su naturaleza real, no permite un conocimiento completo del entorno. El objetivo es determinar qué muestras del entorno hay que seleccionar y durante cuánto tiempo hay que recordarlas para permitir la obtención de modelos adecuados a la operación del individuo.

Este no es un problema específico del MDB. En robótica autónoma, al trabajar con modelos del entorno, es habitual que la información deba ser obtenida a partir de muestras que se adquieren en tiempo real. El objetivo de muchos estudios es modelar dicho entorno mediante una función matemática de la que se conocen ciertos puntos, mayor número cuanto mayor tiempo de vida ha disfrutado el agente. En un caso ideal, bastaría con esperar el tiempo suficiente hasta obtener un número de muestras que nos permita generalizar mediante cualquier técnica matemática básica. En un entorno estático y simple, ésta sería sin duda la mejor solución, y tras un periodo de aprendizaje con una tarea simple, se podrían llevar a cabo tareas más complejas. El problema es que si queremos trabajar en entornos dinámicos, no tiene sentido esta etapa de aprendizaje previa porque la información obtenida nunca será total y el número de muestras necesario sería muy grande.

Independientemente del algoritmo utilizado para la modelar el entorno, tenemos el problema de que los puntos de los que se dispone en un instante dado se deben almacenar de algún modo en un espacio limitado y, por lo tanto, para almacenar nuevos puntos habrá que decidir a cuáles sustituyen en esa memoria limitada. En términos computacionales, diremos que es necesario disponer de una memoria, denominada Memoria a Corto Plazo (MCP), y de un mecanismo de gestión de la misma. En el caso del MDB, el tamaño de la MCP se ve limitado por el coste computacional que conlleva la utilización de un número excesivo de muestras durante la aplicación real. Esto es debido a que, en cada generación del algoritmo evolutivo encargado de adaptar la población de modelos al mundo real percibido por el agente, cada individuo (candidato a modelo actual) ha de realizar predicciones de cada una de las muestras en la memoria. Por otra parte, el tamaño mínimo de la MCP depende de la complejidad de la función y no es posible determinarlo a priori. La situación más eficiente es poder generalizar con el menor número de muestras posibles, y esto se consigue gestionando de forma adecuada la actualización de la MCP. Así, con el paso de las iteraciones, en la memoria estarán almacenados los puntos mínimos necesarios para obtener la función objetivo.

Por tanto, en este apartado queremos abordar la generalización de una función matemática de la que conocemos distintos puntos con el transcurso del tiempo, pero con la limitación de que el número de puntos totales que se puede utilizar a la vez es fijo. Si particularizamos este planteamiento global para el MDB, las muestras o puntos con los que contamos para aproximar los modelos de mundo e internos, se representan mediante los pares acción-percepción.

En todo este análisis, utilizaremos la nomenclatura establecida en el apartado 6.1 donde una iteración del MDB comienza tras la aplicación de una estrategia en el entorno. Por tanto, en cada iteración tenemos una nueva muestra candidata a ser almacenada en la MCP en forma de par acción-percepción. Como consecuencia, el contenido de esta memoria se puede considerar como la "verdad de mundo" instantánea y tratamos de obtener el modelo que mejor la aproxime.

Recordamos que una de las razones para utilizar técnicas de aprendizaje evolutivo para la obtención de los modelos reside en su gran capacidad a la hora de generalizar una función matemática a partir de muestras. El error que obtiene un cierto individuo a la hora de predecir todos los puntos de la función que se ha de aproximar y que son conocidos en un instante dado, constituye el valor de calidad de cada individuo. Por tanto, la función de calidad es una superficie de error respecto a la función deseada. El conocimiento de nuevos puntos de la función a aproximar refina las redes existentes en la población y la predicción es cada vez mejor.

En concreto, utilizaremos en este apartado el algoritmo genético PBGA que ha sido presentado anteriormente, aunque debemos destacar que el estudio que vamos a realizar sobre la gestión de la MCP, es independiente del tipo de algoritmo evolutivo que utilicemos para obtener los modelos.

En resumen, con objeto de minimizar la cantidad de información que debe ser almacenada en la MCP maximizando la relevancia de dicha información para obtener modelos con la mayor precisión posible, a continuación realizaremos un estudio en profundidad de las estrategias de reemplazo para esta memoria.

Estrategias de reemplazo de la MCP

Nuestro objetivo es obtener resultados lo más generales posibles, de modo que sean aplicables a cualquier problema de obtención de un modelo matemático complejo a partir de muestras. Sin embargo, debemos partir de una serie de premisas impuestas por el objetivo final de la utilización de esta MCP, es decir, la aplicación en el MDB. Por tanto, vamos a establecer con claridad las características particulares de nuestro problema:

- Los pares acción-percepción contienen información relacionada tanto con los modelos de mundo como con los modelos internos. Dada la similitud de comportamiento entre los dos tipos de modelos, a la hora de realizar el siguiente estudio nos centraremos en los primeros por ser más complejos. Es decir, para nosotros *la función que viene representada por las muestras es el modelo de mundo* y se tomará como un modelo del entorno en el que el agente se desenvuelve. Al final del apartado generalizaremos los resultados a los modelos internos teniendo en cuenta sus características particulares.
- El objetivo primordial de la estrategia de reemplazo es que *la MCP almacene las muestras que den lugar a un mejor modelado* en cada instante de tiempo. Una vez logrado este objetivo, se valorará positivamente que la estrategia de reemplazo utilice el menor número posible de muestras con objeto de reducir el tiempo de cómputo.
- La función que se debe modelar es dinámica, esto es, puede presentar una dependencia temporal a distintas escalas, por lo que para cumplir el objetivo anterior de minimizar el error y, dependiendo de los objetivos y circunstancias particulares del agente en un momento dado, puede resultar necesario que se almacenen las muestras más generales o las más actuales. *La estrategia de reemplazo deberá, por lo tanto, ser también dinámica* y se irá adaptando a los posibles cambios de la función y las circunstancias del individuo. En cierto sentido, estamos introduciendo un mecanismo de atención en la MCP.
- *El número de muestras que almacena la MCP es limitado*. Por este motivo, no se podrán almacenar todas las muestras que son relevantes. En general, el número de muestras mínimo para modelar exactamente la función será mayor que el tamaño de la MCP.
- La entrada de nuevas muestras en la MCP se realiza en tiempo real y *la gestión que realiza la estrategia de reemplazo debe ser rápida.*
- El modelado de la función también se realiza en tiempo real a partir del contenido de la MCP, por lo que cuantas más muestras se hayan probado mejor será. *El*

contenido de la MCP en un instante dado se toma como "verdad de mundo instantánea".

- Al realizar un modelado de un entorno real en el MDB, existe ruido en la información adquirida independientemente de la fuente de obtención de los datos. Teniendo en cuenta este ruido, *buscamos modelos válidos* para la realización de tareas en el entorno, *no un modelado perfecto con precisión absoluta* (no es el tipo de problemas a los que será aplicado el MDB).

Todas estas condiciones iniciales y de contorno de nuestro análisis no le restan generalidad, porque el objetivo final sigue siendo el mismo: obtener una función a partir del conocimiento en tiempo real de puntos de la misma. Los requerimientos derivados del MDB no hacen sino darle mayor relevancia al estudio al imponer que la función a modelar pueda cambiar con el tiempo.

Teniendo en cuenta estas premisas, la estrategia de reemplazo que hemos desarrollado incluye los siguientes 4 términos básicos:

- **Antigüedad (A):** al utilizar una función dinámica, nos puede interesar guardar en la MCP la información más reciente y no la más general. Ésta última se puede referir a partes de la función que no son relevantes en el instante actual, por lo que se hace necesaria la presencia de un término temporal que controle la localidad de las muestras almacenadas.
- **Distancia (D):** dado que cada muestra contenida en la MCP no es más que un conjunto de puntos, se puede representar como un vector de longitud igual a la dimensión del espacio de trabajo (que viene definido por las variables de entrada y salida de la función a obtener). La distancia euclídea entre estos vectores nos indica si la información está muy dispersa en el espacio n-dimensional de trabajo o si por el contrario todas las muestras están próximas. Desde el punto de vista de la generalización de una función, se deben buscar los puntos más equiespaciados para tener la mayor información posible del dominio y el recorrido.
- **Funcionalidad (F):** las muestras que se encuentran en la MCP y se predicen con mayor error, son más relevantes para el mecanismo de aprendizaje que las que se predicen con menor error.
- **Relevancia inicial (R):** podemos conocer el error con el que el modelo predice cada nueva muestra susceptible de entrar en la MCP. De esta forma, tenemos en cuenta la importancia de una muestra antes de ser aprendida, que es una medida de la particularidad o singularidad de dicha muestra.

Así, buscamos que las muestras contenidas en la MCP nos aporten generalidad espacial (D), generalidad funcional (F), particularidad (R) y actualidad (representada por el término A).

Estos 4 términos se utilizan para etiquetar cada muestra que entra en la MCP, y dicha etiqueta se utiliza para decidir si una nueva muestra debe reemplazar a otra, o no entrar en la Memoria a Corto Plazo. Por tanto, la estrategia de reemplazo combinará los 4 términos en uno sólo con un factor de preponderancia independiente para cada uno (K_i), dando así la posibilidad de modificar la estrategia en función de las necesidades según la siguiente expresión:

$$E = K_a \cdot A + K_d \cdot D + K_f \cdot F + K_r \cdot R$$

Los efectos de la regulación de estos parámetros se estudiarán en los apartados siguientes.

Se hace necesario aclarar que todo proceso de reemplazo en un memoria tiene dos partes, la entrada de datos y la salida:

- En la fase de entrada de datos, se calcula la etiqueta de la muestra susceptible de entrar en la MCP y se decide si debe reemplazar a otra ya existente porque es más relevante en un momento dado. Esta medida de relevancia la aporta la combinación de los 4 términos anteriores. *En esta fase se decide qué es lo que muestreo de la función.*
- En la fase de salida, una vez que se ha decidido que debe haber reemplazo, se selecciona la muestra que debe abandonar la MCP. Para ello, se puede utilizar el mismo criterio que en la fase de entrada, pero también se puede calcular la etiqueta de cada muestra usando otra combinación de términos diferente. *En esta fase se decide qué información retengo de la función* y, en cierto sentido, cuanto tiempo permanece una muestra en la MCP y por lo tanto cuanto afecta al proceso de adaptación del modelo.

Por ejemplo, en una posible estrategia de reemplazo general, cuando se presenta una nueva muestra, se calcula su etiqueta de entrada aplicando la expresión:

$$E = K_a \cdot A + K_d \cdot D + K_r \cdot R$$

donde no se incluye el término de funcionalidad F ($K_f = 0$) porque, como explicaremos a continuación, no es conocido en ese instante, y se compara con la peor etiqueta (calculadas de nuevo aplicando la expresión anterior) asociada a las muestras existentes en memoria. Si la nueva es mejor (si es mayor), se inicia el proceso de selección de la muestra que debe ser reemplazada, y si es peor la MCP permanece sin variación. En caso de reemplazo, recalculamos las etiquetas de las muestras presentes en la MCP incluyendo el término de funcionalidad. La muestra con una etiqueta peor (de menor valor) es la sustituida. El hecho de utilizar distintos términos para decidir qué entra y qué sale de la MCP, nos permite diferenciar entre qué muestra nos puede aportar algo a priori y qué muestra realmente nos lo aporta (y nos interesa mantener).

En un caso más simple, la expresión para el cálculo de la etiqueta de entrada y salida puede ser igual, con lo que la fase de salida de muestras no implica recalcular las etiquetas y, por tanto, si la muestra nueva es mejor que la peor de las presentes en MCP la sustituye directamente.

Ambos son criterios de entrada elitistas que cumplen la condición básica de maximización de la representatividad. Otra opción más estocástica consistiría en que la muestra reemplazada no fuese la peor, sino una seleccionada aleatoriamente. En este sentido, hemos realizado distintas pruebas utilizando una ventana de selección de tamaño menor al tamaño de la MCP para decidir qué muestra debe ser sustituida, y los resultados obtenidos pueden ser consultados en [Bellas, 02].

En todos los experimentos de este apartado, utilizamos el algoritmo PBGA tratando de aprender las muestras contenidas en la MCP. De cara a utilizar la misma nomenclatura que en el MDB, cada vez que se presenta una nueva muestra candidata a entrar en la MCP, diremos que ha transcurrido una iteración. Para cada iteración, el PBGA evoluciona durante un número de generaciones fijo. En el apéndice B se indican los parámetros con los que se ejecutó el algoritmo PBGA en cada experimento.

Tras este planteamiento básico de la estrategia de reemplazo, pasamos a analizar las características particulares de los términos que forman parte de dicha estrategia con objeto de establecer la importancia de cada uno de ellos en distintas condiciones.

Antigüedad

Cuando abordamos una de las problemáticas básicas del MDB, el aprendizaje en entornos dinámicos, se hace necesaria una adaptación de la estrategia de reemplazo para trabajar con el tipo de funciones derivadas del modelado de estos entornos, donde es muy relevante la localidad temporal de las muestras.

En general, la estrategia de reemplazo debe favorecer la existencia en la MCP de los puntos más relevantes para el modelado de la función. Si ésta es dinámica, los puntos más relevantes no son siempre los más equiespaciados o los más generales, sino que pueden ser los más recientes. Por ejemplo, si en un problema dado tenemos un conjunto de muestras en la MCP relevantes para una función dada, y de repente cambia dicha función, ese conjunto de puntos pasa a ser información engañosa que debemos eliminar de la memoria. Por tanto, con objeto de que la MCP pueda mantener las muestras más recientes en caso de necesidad, se considera un término de antigüedad en la estrategia de reemplazo. Este término se calcula simplemente como:

$$A = t$$

donde cada iteración (cada nueva entrada de una muestra) se utiliza como unidad de tiempo *t*.

Si utilizamos únicamente este criterio temporal en la estrategia de reemplazo de la MCP, tendremos que cada nueva muestra susceptible de entrar tiene una etiqueta asociada que se calcula aplicando:

$$E = K_a \cdot A \quad \text{con } K_a = 1$$

Esta etiqueta se compara con la peor de las presentes en la MCP y si es mejor que ella, la sustituye (de cara a mantener en memoria las muestras más recientes, una etiqueta es mejor que otra si es numéricamente mayor). Esta estrategia temporal se denomina FIFO (first in first out, el primero en entrar es el primero en salir) y es puramente temporal, dado que no utiliza ningún tipo de criterio funcional o de relevancia espacial. En general, produce un aprendizaje temporalmente local de la función a modelar (que como veremos en el siguiente apartado, en el caso especial de los algoritmos evolutivos, puede ser compensado con un tiempo de aprendizaje mayor entre iteraciones).

Desde el punto de vista de la interacción de un agente autónomo con su entorno, es una estrategia simple pero adecuada porque mantiene la información más reciente y, por tanto, la más relevante para la "supervivencia". Por este motivo, y por ser la primera estrategia de reemplazo aplicada al MDB, será analizada con detalle en el apartado siguiente.

Estrategia temporal pura de tipo FIFO

A la hora de abordar el modelado de cualquier función usando una estrategia FIFO surgen dos parámetros muy importantes que debemos fijar y que condicionan de manera crucial el aprendizaje: el tamaño de la MCP y el número de generaciones de evolución entre actualizaciones de la MCP (iteraciones). El primer parámetro nos indica cuántos

puntos se van a usar para modelar la función, y es crítico porque un número insuficiente de puntos no permite la obtención de un resultado adecuado de modo que, como norma general, se tiende a trabajar con tamaños de MCP por encima de los realmente necesarios.

Por otra parte, en una estrategia temporal donde el contenido de la MCP cambia continuamente, aunque todas las muestras sean parte de la misma función, la única forma de conseguir un aprendizaje global es "suavizando" el proceso, es decir, en cada iteración se evoluciona durante un número dado de generaciones que, intuitivamente, no debe ser muy grande para evitar que el modelo obtenido sea particular al contenido de la MCP en esa iteración. Esto permite que de una información temporalmente muy local se puedan realizar modelados de funciones globales.

Tamaño de la MCP y generaciones por iteración

Establecer una cota inferior para el tamaño de la MCP no es posible a priori si usamos una estrategia de reemplazo funcional, porque depende de la complejidad y dinamicidad de la función y, como veremos más adelante, de la calidad de los modelos que se van obteniendo. Pero sí podemos realizar un estudio sobre el tamaño de la MCP con una estrategia de tipo FIFO donde el criterio no depende del modelo. Veremos además en este apartado, la influencia que tiene el número de generaciones de evolución por iteración a la hora del aprendizaje en una estrategia FIFO. Este resultado si será extrapolable en general a otro tipo de estrategia de reemplazo.

Supondremos que el entorno tiene la estabilidad suficiente como para poder ser aproximado a una función matemática. Esta suposición se hace teniendo en cuenta que, para un cierto tamaño de la MCP, el número de muestras que almacena en un instante dado no es suficiente para modelar el entorno y se hace necesario un tiempo de iteración. No estamos haciendo una simplificación, estamos imponiendo que el entorno de trabajo sea predecible y no aleatorio e irreal.

Para comparar los resultados que vayamos obteniendo, vamos a utilizar 3 funciones diferentes de complejidad creciente:

- Una función seno simple: $y = sen(x)$
- Una serie caótica logística: $x(t) = 4x(t-1)/1-x(t-1)$
- Una función de dos variables: $z = x \cdot sen(x) + y \cdot sen(y)$

Tenemos pues tres procesos evolutivos donde, dependiendo de la complejidad de la función, el tamaño de la MCP requerido puede ser mayor y el número de generaciones de evolución también. Intentaremos encontrar un patrón de comportamiento aplicable a cualquier función para que la elección de estos dos parámetros no sea arbitraria en cada problema al usar un reemplazo FIFO.

Es muy importante recalcar que el tamaño de la MCP necesario no se podrá determinar a priori en general, porque la función no es conocida de antemano y además depende de la estrategia de reemplazo usada. En este estudio, utilizamos funciones teóricas por lo que podríamos saber el número mínimo de puntos necesarios para aproximar cada una de ellas, pero esto no es aplicable al modelado de un entorno real. Por este motivo, supondremos que el tamaño de la MCP necesario no se conoce y partiremos de una estimación aproximada que refinaremos de acuerdo con los resultados. Nuestro objetivo no es tanto poder estimar el tamaño de MCP necesario en

cada caso, como encontrar una relación entre dicho tamaño y el número de generaciones de evolución necesarias.

En la figura 9 se presenta la evolución del error cuadrático medio para la predicción que proporciona la mejor red, obtenida tras 1000 iteraciones, de todas las muestras contenidas en la MCP. Las gráficas muestran, en el eje horizontal, la variación en el número de generaciones por iteración para distintos tamaños de MCP (en diferente color). En la figura 10 representamos los mismos resultados pero ahora el eje horizontal indica el tamaño de la MCP para distinto número de generaciones. Las gráficas superiores en ambas figuras corresponden al modelado de la función seno, las centrales corresponden a la serie logística y las inferiores a la función 3D.

Si analizamos con detenimiento la gráfica superior de la figura 9 (función seno), vemos cómo el error permanece prácticamente constante para tamaños de MCP por encima de 8, independientemente del número de generaciones por iteración. Es decir, el algoritmo necesita 8 muestras de la función seno para poder predecirla en todo su dominio con un error inferior al 10%, y el tiempo que le damos para aprender cada nuevo punto no es relevante. Además vemos cómo, al aumentar el tamaño de la MCP, disminuye el error pero en una medida muy pequeña. Así, para un tamaño de MCP de 14 muestras, el error es de un 5% mientras que para un tamaño 40 es de un 3%. Sin embargo, utilizar 40 muestras en lugar de 14 se traduce en un tiempo de cómputo 3 veces mayor, ya que cada nueva muestra ha de ser probada en todas las redes del algoritmo evolutivo en cada generación. Es decir, sólo tiene sentido trabajar por encima de un tamaño 14 en problemas que requieran muy alta precisión.

Lo que es realmente interesante de estos resultados es lo que ocurre para tamaños de MCP por debajo de 8, donde vemos cómo al aumentar el número de generaciones por iteración, el error aumenta. Esto se debe a que cuando el número de generaciones es grande para tan pocas muestras, el algoritmo aprende con exactitud el conjunto de puntos de la MCP en un instante dado pero no es capaz de generalizar y, cuando entra una nueva muestra, la población del algoritmo evolutivo ha convergido hacia una solución particular. Esto implica que, en cada iteración, la función debe ser reaprendida y el error no puede mejorar.

La situación es similar para el caso de la serie logística que aparece en la gráfica central de la figura 9. Aunque la función es más compleja, el tamaño mínimo de MCP para conseguir un error aceptable de un 5% es similar, y está en torno a 14. La mayor complejidad se refleja en que la zona donde el error es independiente del número de generaciones empieza más arriba, en concreto en 14 y no en 8 como ocurría con la función seno. Por debajo de este tamaño, el error aumenta con el número de generaciones tal y como explicamos en el caso de la función seno. Este hecho es también reflejo de que el número de puntos es insuficiente para aproximar la función buscada, y el algoritmo llega a soluciones diferentes dependiendo del tiempo de aprendizaje. Una vez que el número de puntos es suficiente, el tiempo de aprendizaje no es relevante porque siempre se alcanza la misma solución.

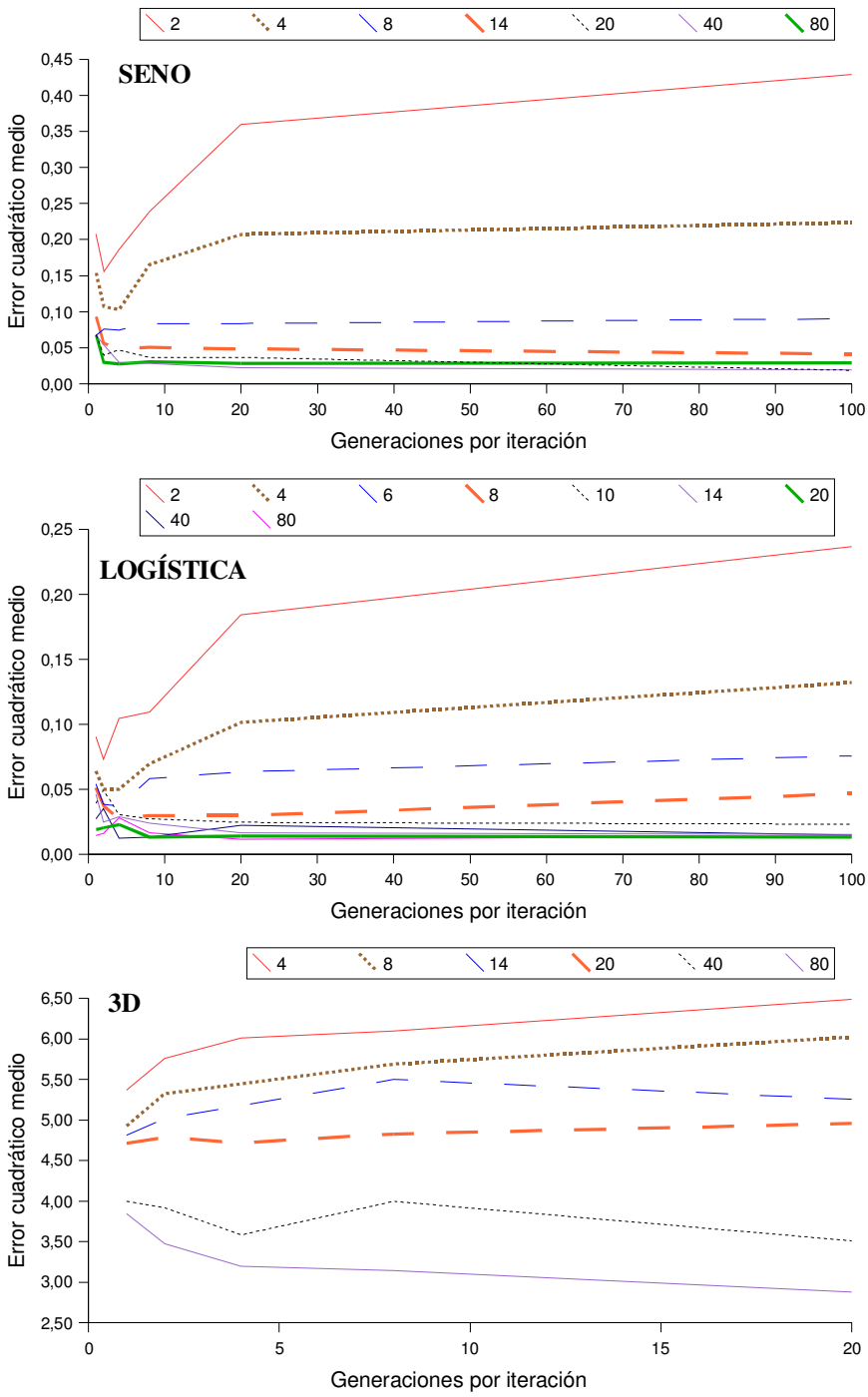


Figura 9. Error cuadrático medio entre la predicción de la mejor red tras la evolución y la función real para distinto número de generaciones por iteración. Los distintos colores de las líneas representan distintos tamaños de MCP.

Para la función 3D (gráfica inferior figura 9), se confirman los resultados y lo que vemos es que el tamaño de MCP donde el error se vuelve independiente del número de generaciones ha ascendido hasta 20. De hecho, en esta función, la zona de estabilidad tiene una cierta pendiente decreciente, es decir, al contrario de lo que ocurría con las otras dos funciones, al aumentar el número de generaciones disminuye el error. Esto es debido a que la función 3D es más compleja y el aprendizaje no es perfecto con los parámetros que hemos utilizado para las otras dos (esta función requiriría más de 1000 iteraciones para ser aprendida).

Analizando ahora las 3 gráficas de la figura 10, podemos ver cómo el error sigue una tendencia monótona decreciente al aumentar el tamaño de la MCP independientemente del número de generaciones por iteración. Parece, por tanto, que existe un punto crítico en el tamaño de la MCP por encima del cual el error no disminuye significativamente, mientras que el coste computacional se multiplica. Este punto crítico coincide con la zona donde las curvas se vuelven cóncavas. Para la función seno y para la serie logística, el punto crítico lo podemos situar por en torno a 10 y para la función 3D en torno a 20 o más arriba. Estos puntos son muy importantes, ya que nos indican que si el tamaño de la MCP está por encima de ellos, la mejora en el error obtenida al aumentar el número de generaciones por iteración es poco relevante frente al coste computacional.

En todos los casos, si usamos solamente una generación de evolución, el error es grande porque no hay tiempo de aprendizaje. Lo mismo ocurre para 2 generaciones, y estas son las curvas de mayor error en todas las gráficas de la figura 2. Pero a partir de 4 generaciones, el error es prácticamente constante, por lo que concluiremos que, una vez conocido el punto crítico en el tamaño de la MCP, el número de generaciones ideal está por encima y en el entorno de 4 (eficiencia mayor).

En la figura 11 se muestra la variación del error cuadrático medio en la predicción de la mejor red en cada generación cuando tomamos como modelo de mundo la serie logística. Las 2 gráficas superiores corresponden a una MCP de tamaño 2, a la izquierda se ha evolucionado durante 2 generaciones por iteración y en la derecha durante 100 generaciones. Como se observa, son dos casos extremos que nos servirán para profundizar en el conocimiento de la dinámica del aprendizaje evolutivo de funciones de calidad muestreadas.

Usando 2 generaciones, se aprecia mejora en el error aunque las oscilaciones son muy grandes. Esto se debe a que estamos usando únicamente dos puntos para predecir la función, con lo que la entrada de uno nuevo implica un aumento muy significativo en la cantidad de información. A lo largo del tiempo es de esperar un aprendizaje, aunque sea lento. De hecho alcanzamos un error de un 5%.

Por otra parte, cuando usamos 100 generaciones, prácticamente no hay aprendizaje porque cada par de puntos es aprendido por toda la población, y la entrada de uno nuevo trae consigo el reaprendizaje de un nuevo modelo. Es decir, en cada iteración se aprende una función particular dada por dos puntos, pero no la serie logística buscada. El error oscila en torno a un 25%.

En las dos gráficas centrales de la figura 11, usamos una MCP de tamaño 8, es decir, muy cerca del punto crítico para esta función (recordamos que estaba por encima de 10). En este caso, el error es muy parecido para 2 y para 100 generaciones (5%) aunque para 100 las oscilaciones son mayores. Este resultado es indicativo de que con 8 puntos ya está claramente definida la serie logística, y el aprendizaje con 100 generaciones no

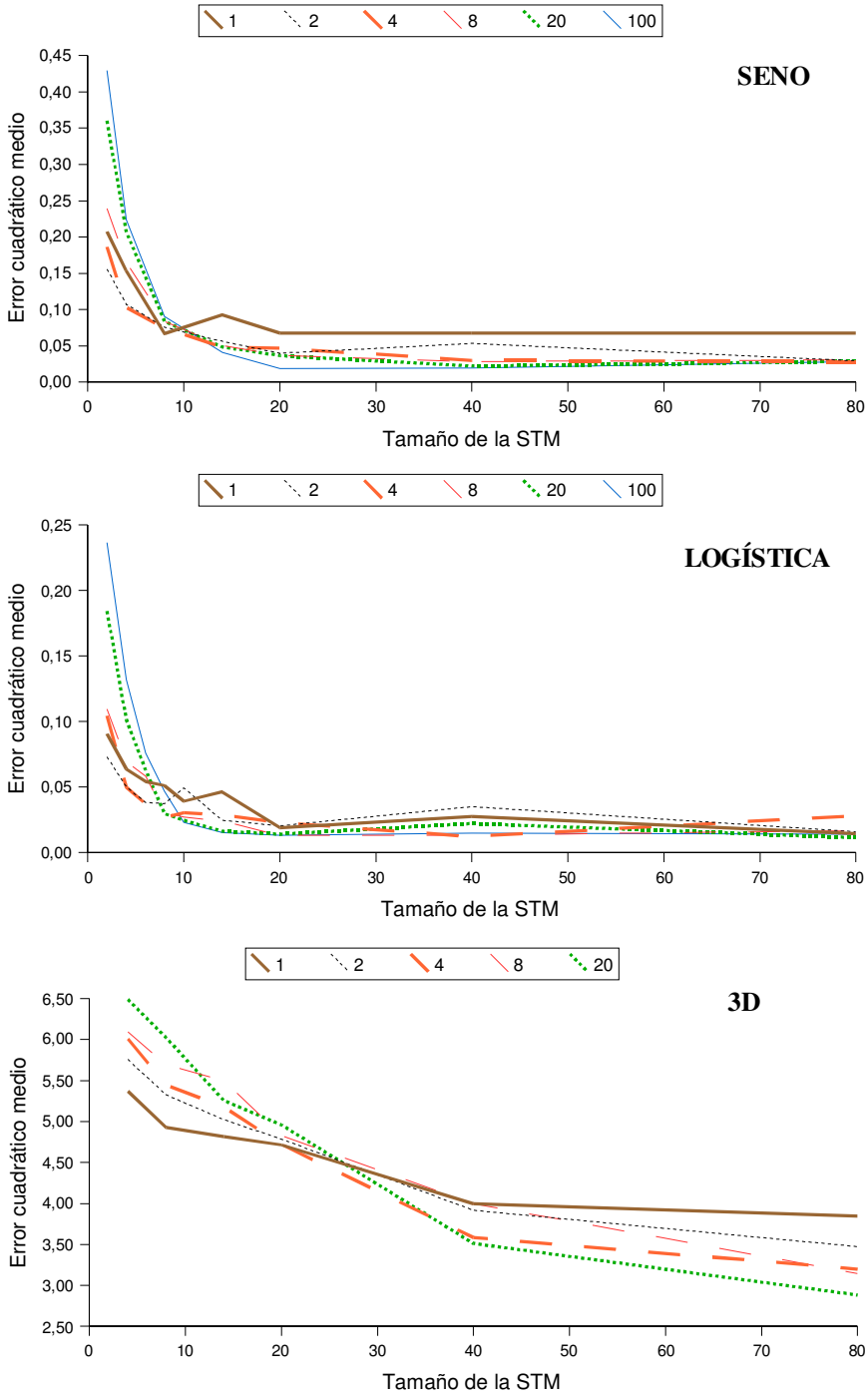


Figura 10. Error cuadrático medio entre la predicción de la mejor red tras la evolución y la función real para distintos tamaños de MCP. Los distintos colores de las líneas representan distinto número de generaciones por iteración.

es ambiguo: tras cada iteración se aprende la misma función. Las oscilaciones aparecen porque estamos ligeramente por debajo del punto crítico, con lo que la existencia de unos puntos u otros dentro de la MCP hace que toda la población haya convergido hacia una solución muy específica. Llega un momento en que las redes no aprenden la función general, sino que se especializan en una zona en concreto de dicha función. La entrada de un nuevo punto en la MCP, hace que las redes que eran muy específicas de la MCP anterior sean deficientes en la predicción del nuevo punto, con el consecuente aumento del error. En cada iteración, la población reaprende el nuevo punto, pero ahora es sólo una octava parte de la función de calidad, con lo que la repercusión en el error final no es tan relevante como en el caso anterior con una MCP de tamaño 2.

Para una MCP de tamaño 20 (gráficas inferiores de la figura 11) ya estamos por encima del punto crítico y, de acuerdo con los resultados de las figuras 9 y 10, el error no depende del número de generaciones. Así, al utilizar 2 generaciones el error se queda en un 2.5% mientras que usando 100 generaciones baja hasta un 1%. Lo que ocurre es que con 2 generaciones el aprendizaje es más lento, por lo cual si esta evolución tuviese más tiempo llegaría a los niveles de error de la de 100. Como ya hemos visto antes, al trabajar por encima del punto crítico, el aumento de generaciones no es perjudicial en cuanto a las oscilaciones ya que la función que aprenden todas las redes de la población es siempre la misma, y no existe transición al entrar un nuevo punto (o si existe está minimizada por el tamaño de la MCP).

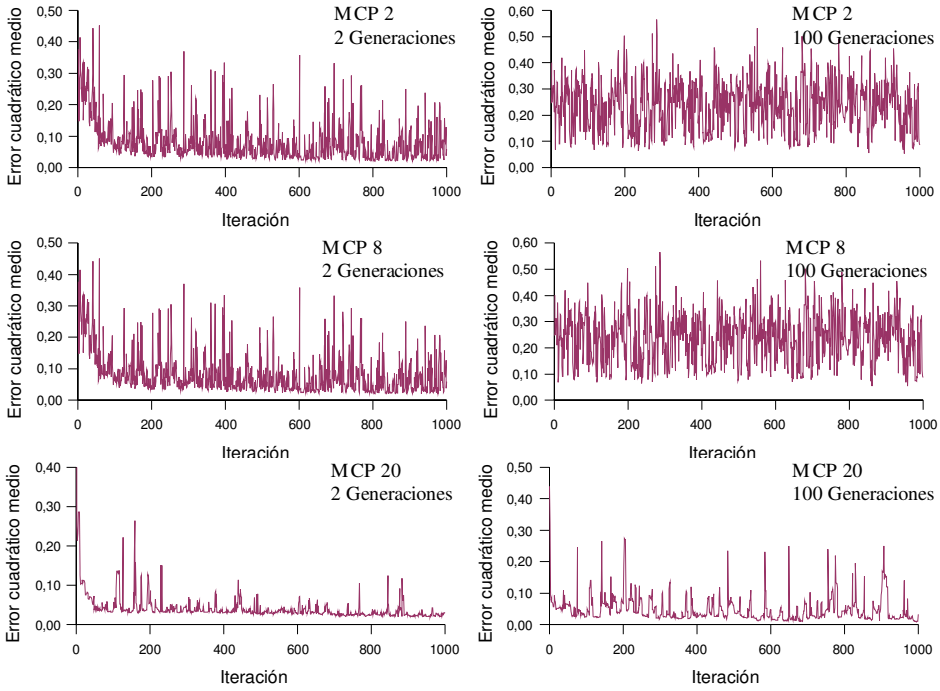


Figura 11. Variación del error durante una evolución para 3 tamaños de MCP y con distinto número de generaciones por iteración usando como modelo la serie logística.

Puntos críticos

Cuando el tamaño de una MCP está por encima del punto crítico que se muestra en las gráficas de las figuras 9 y 10, nos podríamos preguntar ¿por qué no evolucionar simplemente un gran número de generaciones una MCP estática (sin ningún tipo de estrategia de reemplazo), sin introducir más datos?. Existen dos razones para esto. Por una parte, esta estrategia no proporcionaría capacidad para considerar entornos cambiantes y por otra, el punto crítico que hemos estudiado hasta ahora es lo que se ha llamado un punto crítico de muestreo dinámico, esto es, los contenidos de la MCP son suficientes para modelar la señal si cambian a un ritmo razonable. De hecho, se pueden definir tres tipos diferentes de tamaños MCP críticos. Por una parte tendríamos lo que llamamos *punto crítico estático de muestreo consecutivo* (SCS-CP). Este tamaño crítico de la MCP indica cuál ha de ser el número mínimo de muestras para que pueda proporcionar una representación de la señal suficientemente buena si ningún dato nuevo entra en la MCP, y los datos que contiene ésta son muestras consecutivas de la señal. Evidentemente, este punto depende de cuán buena sea la representación de la señal que uno desee y de la frecuencia de muestreo. Para una señal sinusoidal muestreada a 63 muestras por ciclo, el SCS-CP es 63.

El segundo tipo de punto crítico viene dado por el tamaño mínimo de MCP requerido para representar la señal de forma fidedigna por medio de una MCP estática cuyo contenido son puntos aleatorios de la señal. Esto es lo que llamamos el *punto crítico estático de muestreo aleatorio* (SRS-CP). Para la función seno este punto se encuentra alrededor de 20.

Finalmente, tenemos el punto crítico que presentamos anteriormente, esto es, el *punto crítico de muestreo dinámico* (DS-CP). En este caso, la MCP se actualizará con algún tipo de estrategia de reemplazo. Como en el caso de los puntos críticos estáticos, también dependerá de cómo muestreemos las señales o entornos. Obviamente, un DCS-CP (de muestreo consecutivo) presentará un valor normalmente más alto que un DRS-CP (con muestreo aleatorio) dada la mayor probabilidad que presenta un muestreo aleatorio de proporcionar una buena representación de la señal. De hecho, para la función seno el DRS-CP se encuentra alrededor de un tamaño de MCP 10 para la misma frecuencia de muestreo. Veremos en el siguiente apartado la gran influencia que tiene el tipo de muestreo que se realiza al usar una estrategia de tipo FIFO.

Desde un punto de vista práctico, todos estos comentarios nos llevan a considerar que el tamaño mínimo necesario para que la MCP pueda proporcionar una representación suficientemente buena de la señal depende de dos factores: la frecuencia de muestreo y la dinamicidad.

Respecto a la dinamicidad, cualquier estrategia evolutiva actuando sobre una MCP dinámica, esto es, una MCP que se actualiza con nueva información según van ocurriendo nuevas iteraciones con la función o el entorno, verá una MCP virtual que es más grande que la real mientras se mantenga una presión evolutiva pequeña. Por lo tanto, en la aplicación práctica de este tipo de estrategias, la utilización óptima de la MCP tendrá lugar cuando la máxima información sobre la señal se introduzca en ella a través de la estrategia de actualización y reemplazo apropiada, y cuando esta información cambie con el tiempo a un ritmo compatible con el número de generaciones y presión evolutiva del algoritmo entre iteraciones. Cuando se evoluciona utilizando una MCP dinámica, no se quiere que la población converja al contenido actual de la MCP ya que esto nunca modela la función subyacente o el entorno.

Con el conocimiento que ahora tenemos de la influencia del tamaño de la MCP y del número de generaciones por iteración en la evolución con una estrategia de tipo FIFO, pasamos a realizar una serie de pruebas con una función a modelar que utilizaremos durante todo el apartado para comparar los distintos términos de la estrategia de reemplazo. Estas pruebas nos mostrarán las limitaciones y las ventajas de este tipo de estrategia temporal.

Limitaciones de la estrategia FIFO

En la figura 13 hemos representado la evolución del error cuadrático medio en la predicción de las muestras contenidas en la MCP que realiza la mejor red tras cada iteración, al intentar modelar la función de la figura 12 usando una estrategia de reemplazo temporal pura de tipo FIFO para una MCP de tamaño 10. La función de la figura 12 es una campana de Gauss de pequeña amplitud que usaremos para comparar los distintos términos de la estrategia de reemplazo en sucesivos apartados. Para que esta comparación se realice en idénticas condiciones experimentales, utilizamos un conjunto de puntos fijo de la función gaussiana. Para obtenerlos realizamos un muestreo de 645 puntos, de los cuales un 30% pertenecen a la zona de la campana y el resto al valle. Estas 645 muestras se pueden presentar de forma aleatoria, con objeto de simular un caso real en el que un agente está explorando su entorno y lo ve sin orden repetidas veces, o bien de forma secuencial. Este último tipo de muestreo nos da más control sobre el instante exacto en que las muestras pueden entrar en la MCP. La figura 13 corresponde al caso en que los datos son muestreados de forma secuencial.

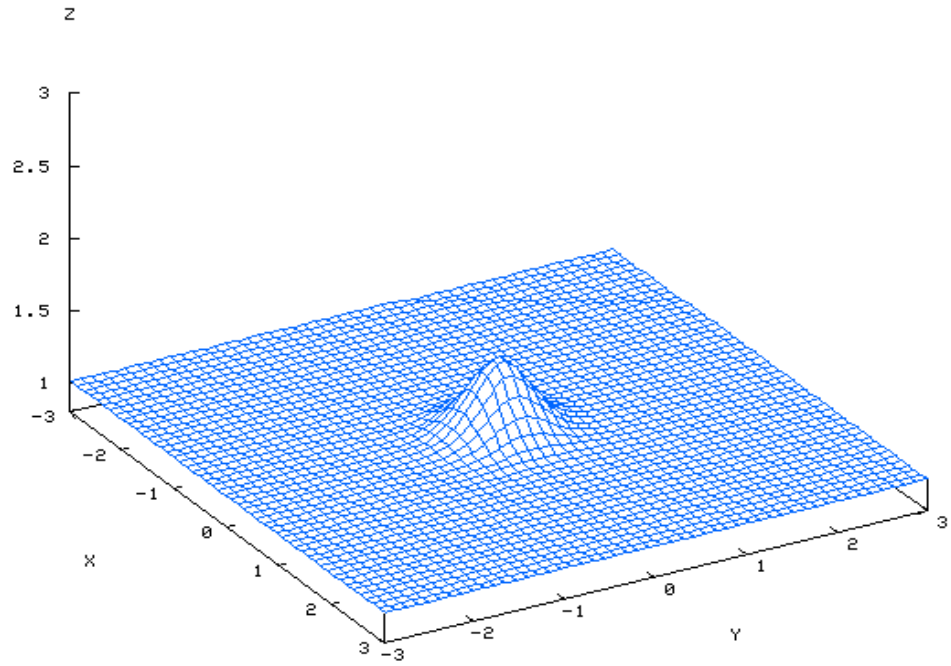


Figura 12. Campana de Gauss que utilizamos como función a modelar para comprobar la relevancia de los datos almacenados por la estrategia de reemplazo.

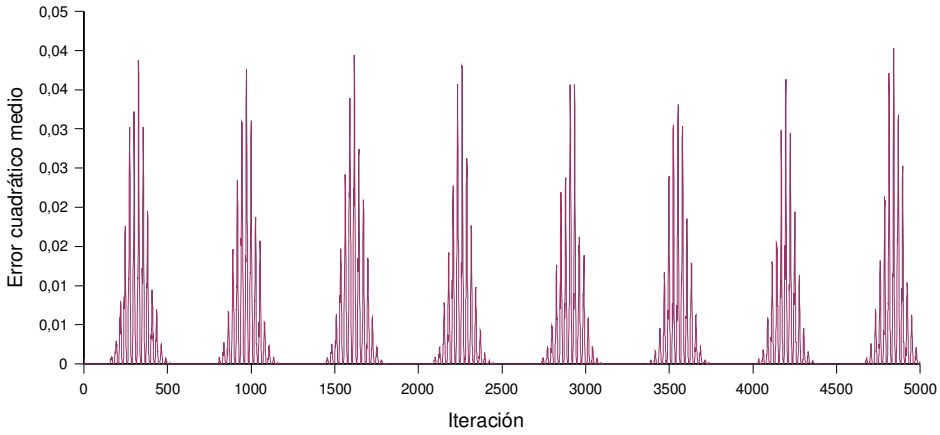


Figura 13. Evolución del error cuadrático medio en la predicción de la MCP que realiza la mejor red en cada iteración usando la función gaussiana. Se ha aplicado una estrategia de reemplazo temporal pura de tipo FIFO para una MCP de tamaño 10 y la función gaussiana ha sido muestreada de forma secuencial.

Volvemos a recalcar que el objetivo de la evolución es obtener un modelo que realice una predicción globalmente buena, en este caso, de la función gaussiana de la figura 12, incluyendo la campana.

En la figura 13 vemos que el error es enorme cuando las muestras que entran en la MCP corresponden a la zona intermedia del intervalo de muestreo, que coincide con la zona de la campana en la figura 12. En las demás iteraciones, el error es prácticamente cero. Con una MCP tan pequeña (sólo 10 puntos para modelar toda la función, por debajo del punto crítico DCS-CP) y sin ningún tipo de criterio funcional de entrada o salida, la evolución oscila continuamente al pasar de la zona plana a la campana. Usamos un tamaño de MCP pequeño buscando la estrategia de reemplazo que consiga

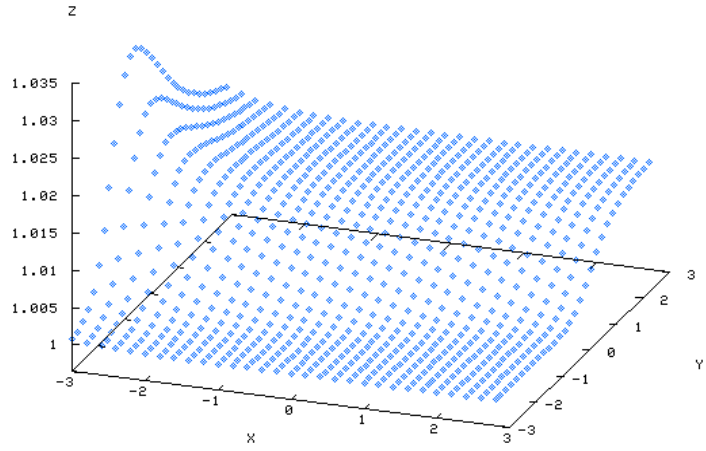


Figura 14. Modelado de la función gaussiana obtenido tras 5000 iteraciones de evolución con una estrategia FIFO y muestreo secuencial de la función test.

un mejor modelado de la función gaussiana con tan pocos puntos. No olvidemos que la minimización del tamaño de la MCP, manteniendo una buena predicción de la función, es uno de los objetivos de la estrategia de reemplazo. Al final del apartado mostraremos cómo sí es posible dicho modelado con un tamaño de MCP de 10.

En el ejemplo que nos ocupa, los contenidos de la MCP en cada instante son diferentes aunque forman parte de la misma función, pero el algoritmo evolutivo no es capaz de generalizar a partir de ellos por ser insuficientes y cambiantes. En la figura 14 mostramos la predicción de la función gaussiana que realiza la mejor de las redes obtenidas tras 5000 iteraciones. La gráfica ha sido representada en una escala ampliada en el eje z y con una orientación diferente con objeto de mostrar con más claridad la predicción obtenida. Como vemos, el resultado es pobre, fundamentalmente porque estamos viendo la predicción de la iteración 5000, que contiene información local sobre los 10 últimos puntos, pero nunca sobre la función global. Esto ocurre en cualquier iteración donde probemos la mejor red existente para distintas zonas de la gaussiana.

Si aumentamos el tamaño de la MCP, conseguimos mejores modelos cada vez porque la información disponible deja de ser tan local y la MCP pasa a guardar más

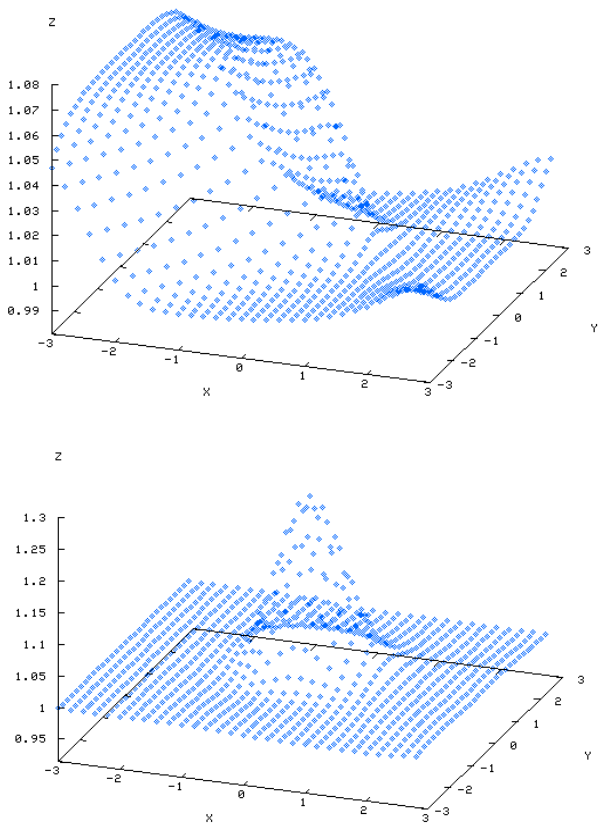


Figura 15. Modelado de la función gaussiana obtenido tras 5000 iteraciones de evolución con una estrategia FIFO y muestreo secuencial de la función test. La gráfica superior corresponde a una MCP de tamaño 50 y la inferior a una MCP de tamaño 300 .

puntos de la singularidad (zona de la campana). La figura 15 muestra, en la gráfica superior, la predicción obtenida usando una MCP de tamaño 50 (que aún es pobre) y, en la inferior, una MCP de tamaño 300 (mejor). Con un muestreo de 645 puntos y teniendo en cuenta que un 30% son de la zona de la campana, al usar 300 muestras, siempre hay un número considerable de puntos de dicha zona en la MCP por lo que existe información para un modelado global.

A la vista de los resultados de predicción, el punto crítico de muestreo dinámico consecutivo (DCS-CP) está entre 50 y 300 para esta función, y no nos interesa situarlo con mayor exactitud. Lo que es relevante de este resultado es que el tamaño de MCP necesario para modelar la función gaussiana globalmente es muy grande. Desde nuestro planteamiento, la estrategia de reemplazo es ineficiente.

Si la entrada de muestras a la MCP se hace de forma aleatoria, la estrategia de tipo FIFO da mejores resultados. Esto es debido a que mantiene siempre un número fijo de muestras que va cambiando, pero no de forma brusca. Es decir, en el caso del muestreo secuencial y para la función gaussiana, la MCP guarda durante muchas iteraciones muestras de la zona plana y, de repente, la información cambia de modo radical al llegar a la campana. Al usar muestreo aleatorio, no existe ese cambio brusco y la función se aprende de modo gradual.

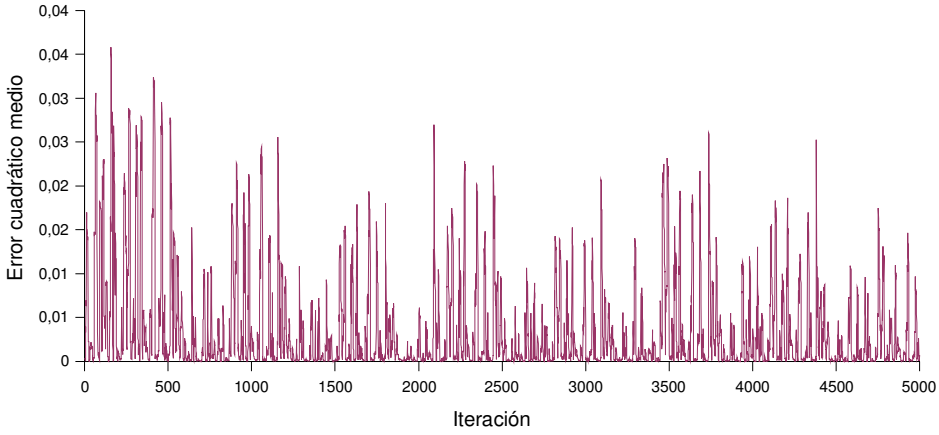


Figura 16. Evolución del error cuadrático medio en la predicción de la MCP que realiza la mejor red en cada iteración para la función gaussiana. Se ha aplicado una estrategia de reemplazo temporal pura de tipo FIFO para una MCP de tamaño 10 y la función gaussiana ha sido muestreada de forma aleatoria.

En la figura 16 se representa de nuevo la evolución del error cuadrático medio al modelar la función gaussiana de la figura 12 con una estrategia de reemplazo temporal pura de tipo FIFO para una MCP de tamaño 10, usando ahora muestreo aleatorio de la función. Si la comparamos con la figura 13, vemos que las oscilaciones aparecen en todas las iteraciones y no concentradas en ciertas zonas. Esta oscilación general se debe a que el error cambia mucho con cada muestra nueva que entra en la MCP. Pero existe una tendencia hacia la mejora de dicho error, que se traduce en la predicción obtenida por el mejor modelo y que se muestra en la figura 17.

Aunque la forma general de la función es adecuada, la amplitud de la campana es muy inferior a la real (z=1.5). Lo que ocurre es que en la iteración 5000 no hay ningún punto de la zona de la campana en la MCP, y aunque la función ha sido aprendida

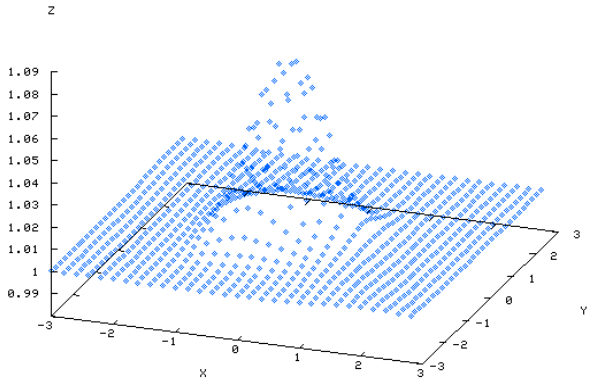


Figura 17. Modelado de la función gaussiana obtenido tras 5000 iteraciones de evolución con una estrategia FIFO en una MCP de tamaño 10 y muestreo aleatorio de la función test.

durante mucho tiempo, la tendencia al usar una estrategia FIFO es de localidad, y por eso la predicción es perfecta en la zona plana que es la representada en la MCP en ese instante. Para ilustrar este hecho, mostramos en la figura 18 la predicción que realiza el mejor modelo en la iteración 3000, donde sí estaban almacenados puntos de la campana (el punto z=1.5 en concreto). Como se observa en la figura, ahora la amplitud es perfecta (valor real = 1.49) en detrimento de la forma de la campana, demasiado ancha. Esta precisión en el modelado es suficiente para nuestra aplicación, lo que no es tan deseable es que el modelado dependa tanto de la zona de la función que se presenta en un instante dado.

Como primeras conclusiones a estos resultados, podemos decir que la forma de presentar los datos a una estrategia de tipo FIFO es crítica, resultando una estrategia eficiente en casos donde la información se presenta de forma aleatoria, y una estrategia

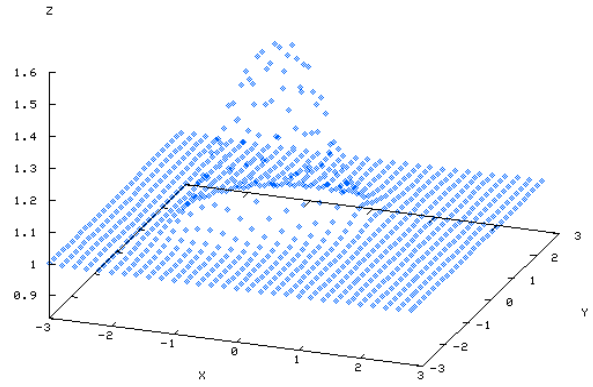


Figura 18. Modelado de la función gaussiana obtenido tras 3000 iteraciones de evolución con una estrategia FIFO en una MCP de tamaño 10 y muestreo aleatorio de la función test.

muy deficiente si se presenta de forma secuencial. La solución en este segundo caso pasa por utilizar un tamaño de MCP muy grande, solución que no es válida en un mecanismo que debe trabajar en tiempo real. Esta dependencia de la forma en que aparece la información, hace que una estrategia FIFO no sea una buena estrategia en un caso general, pero si lo sea si tenemos control sobre la forma de presentar dicha información (control sobre el proceso de enseñanza).

Tras este estudio en profundidad de las estrategias de tipo FIFO, volvemos al planteamiento original de la estrategia de reemplazo para concluir este apartado sobre la inclusión de la antigüedad de las muestras. Por el momento hemos estudiado uno de los dos casos extremos de utilización del tiempo, una estrategia puramente temporal FIFO. El otro extremo será analizado en los apartados siguientes donde no utilizaremos término temporal, sino términos de distancia y funcionales.

Distancia

El siguiente término planteado para la estrategia de reemplazo que vamos a analizar es el de distancia euclídea entre muestras. Como ya se ha comentado en la introducción de este apartado, cada muestra puede ser representada por un vector, tanto en un problema general de modelado de una función a partir de puntos como en el MDB.

En las pruebas que realizaremos en este apartado, el cálculo de la etiqueta asociada a las muestras se lleva a cabo aplicando la expresión:

$$E = K_d \cdot D \quad \text{con } K_d = 1$$

es decir, no utilizaremos el término de distancia junto con el temporal ya visto, sino que es un término que tiene sentido por sí solo (aunque veremos que tiene serias limitaciones) y, en consecuencia, lo analizaremos por separado.

Existen múltiples algoritmos que ponderan la distancia entre vectores buscando maximizar la dispersión de los puntos. De todos ellos, pasamos a mostrar los resultados obtenidos con dos diferentes, uno que utiliza la media y la desviación típica y otro que utiliza las distancias mínimas entre muestras.

Media y desviación típica de distancias

Una posible estrategia que tiene en cuenta la distancia euclídea entre muestras, se basa en la media de las distancias y la desviación típica respecto a dicha media. En concreto, tras calcular la distancia de cada muestra presente en la MCP respecto a todas los demás, utilizamos la media aritmética de éstas para calcular la desviación típica de cada vector respecto a la media. Una vez hecho esto, la estrategia de reemplazo busca maximizar la distancia entre muestras ponderada por la desviación típica, de forma que los puntos estén lo más equiespaciados posible. Por tanto, cada muestra lleva asociado un término de distancia D que se debe maximizar y que se calcula mediante la expresión:

$$D = \frac{(max(d_i))}{\sigma}$$

donde en el numerador tenemos la máxima de las distancias d_i de la muestra respecto al resto de muestras y en el numerador tenemos la desviación típica $\boldsymbol{\sigma}$ respecto a la media de las distancias.

En la gráfica de la izquierda de la figura 19 se observa el conjunto de muestras utilizado como aprendizaje, que pertenecen a la serie logística que han sido representados en un diagrama XY. Tenemos una distribución espacial de puntos simple y, con el criterio de distancia utilizado, buscamos que la MCP almacene un conjunto de ellos lo más distribuido posible.

El contenido final de la MCP se muestra en la gráfica de la derecha de la figura 19 tras 1000 iteraciones de evolución con una MCP de tamaño 10 y, como podemos ver, la distribución se limita a los extremos. Con el criterio utilizado, los puntos de la gráfica están en una distribución de máxima distancia media entra ellos (aunque todos a la misma, o muy similar) y la desviación respecto a la media de todos ellos es prácticamente nula por lo cual su probabilidad de abandonar la MCP es muy pequeña. Se observa como incluso los puntos se repiten dentro de la MCP (en el punto (0,0) hay 4 muestras iguales), porque esta configuración es la que mantiene a los puntos más equiespaciados.

Como se ha explicado en la introducción de este apartado, cada muestra susceptible de entrar en la memoria lo hará si es mejor que la peor de las presentes, esto es, si aporta nueva información a la generalización de la función. Por tanto, en la MCP que se representa en la gráfica de la derecha de la figura 19, es muy poco probable que otra muestra mejore las condiciones de distancia de los puntos que se ven. Con este criterio, se obtiene una distribución de puntos agrupados en los extremos de la función, es decir, en los puntos más alejados. Pero el objetivo de incluir un término de distancia en la estrategia de reemplazo no es detectar estos extremos, sino almacenar puntos distribuidos a lo largo del recorrido de la función. Por supuesto que necesitamos detectar los extremos para poder aproximar, pero también queremos almacenar puntos intermedios que en funciones complejas son necesarios para que el modelado de la función tenga una precisión mínima.

Por ejemplo, con los puntos de la gráfica derecha de la figura 19, se podría obtener un modelado similar a un triángulo, por ser el más simple a partir de esos puntos que no es el resultado buscado. Por tanto, esta estrategia de reemplazo nos proporciona una distribución espacial de la función muy pobre en términos de generalidad.

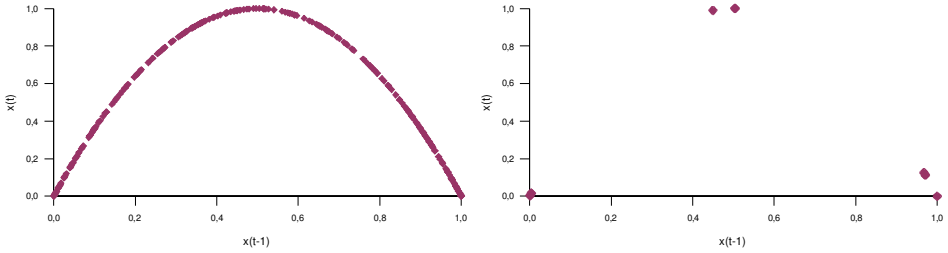


Figura 19. En la gráfica de la izquierda se muestra la distribución espacial de los puntos susceptibles de formar parte de la MCP para la serie logística. En la de la derecha vemos los 10 puntos contenidos en la MCP tras 1000 iteraciones usando una estrategia de reemplazo que maximiza la distancia entre muestras ponderada por la desviación típica respecto a la media. Los puntos de la gráfica de la derecha están superpuestos, es decir, en torno al punto (0,0) hay 4 muestras, en torno al (0.5, 1) hay 2 muestras y en torno al (1,0) hay 4, que hacen un total de 10.

Maximización de las distancias mínimas

De cara a mejorar la distribución obtenida con la estrategia anterior, planteamos una estrategia de tipo min-max, es decir, buscamos la maximización de la distancia mínima entre muestras. En concreto, tras calcular la distancia euclídea de cada muestra a todas las demás, nos quedamos con la menor de ellas como etiqueta para asociar una probabilidad de reemplazo. De este modo, la muestra con una distancia muy pequeña a otra vecina, tiene una gran probabilidad de ser reemplazada. En este caso, cada muestra lleva asociada una etiqueta *D* que se calcula mediante la expresión:

$$D = min(d_i)$$

y que tendrá mayor preponderancia cuanto mayor sea numéricamente.

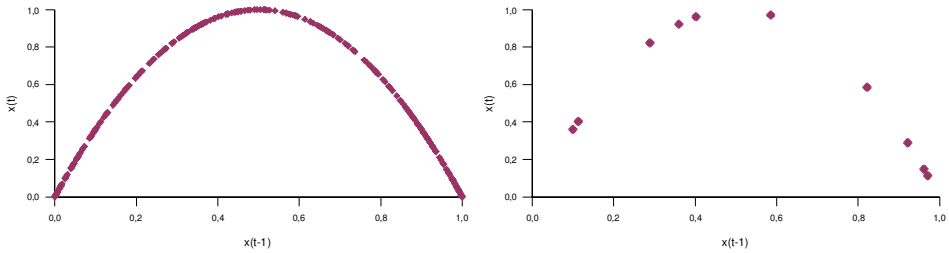


Figura 20. En la gráfica de la izquierda se muestra la distribución espacial de puntos para la serie logística. En la de la derecha vemos los 10 puntos contenidos en la MCP tras 1000 iteraciones usando una estrategia de reemplazo que maximiza las distancias mínimas.

El resultado obtenido con este criterio es mejor que antes y lo mostramos en la figura 20 donde se ha evolucionado durante 1000 iteraciones (una MCP de tamaño 10) al igual que en el caso anterior. Como vemos en la gráfica de la derecha, la distribución de puntos es ahora más adecuada y se adapta al concepto de generalización que estábamos buscando. Lógicamente, si se presentan puntos que mejoren la condición de distancia mínima (por ejemplo un punto muy cercano al 0-0), serán incluidos en la MCP en detrimento de los que se encuentren próximos entre sí.

Se podría esperar una distribución más homogénea de los puntos, pero no debemos olvidar las características particulares de nuestro problema: estamos usando un conjunto limitado de puntos de la función. Por este motivo, con un muestreo aleatorio como el que hemos usado, los 10 puntos más equiespaciados pueden no coincidir nunca a la vez en la MCP. Es decir, la estrategia min-max que aplicamos, o cualquier otra similar, tiende a equiespaciar lo puntos contenidos en la MCP, no todos los puntos del dominio de la función. Esto sólo lo haría, únicamente, tras sucesivas pasadas por todas las muestras posibles.

Con este resultado damos por cerrado el estudio del término de distancia de forma independiente en la estrategia de reemplazo. La idea básica que extraemos es que el algoritmo min-max cumple en principio nuestros requerimientos de generalización espacial de las muestras.

En el siguiente apartado se analiza el término de funcionalidad como complemento al término de distancia establecido. Es decir, consideramos que una buena distribución espacial de las muestras es siempre necesaria y trataremos de añadir nuevos términos que contribuyan a dar mayor relevancia a las muestras almacenadas desde un punto de vista funcional.

Funcionalidad

Al trabajar con problemas reales, las funciones no son tan monótonas como las funciones teóricas que hemos utilizado en el apartado anterior. Es algo común en los entornos reales la existencia de puntos aislados que no siguen el patrón general del resto, y que se suelen englobar en un concepto denominado *ruido*. En este tipo de funciones, el criterio min-max puede resultar ineficiente porque no tiene en cuenta la dificultad inherente al aprendizaje de estas muestras espúreas.

En la figura 21 utilizamos como función a modelar la representada por un conjunto de datos tomado del problema real que veremos en el apartado 7.1.4 y que ya hemos utilizado en la aplicación práctica del PBGA. Los datos corresponden a los ángulos predichos por el sensor virtual del apéndice A mientras el robot Hermes II se desenvuelve en su entorno. En esta figura mostramos la predicción que realiza la mejor de las redes obtenidas tras la evolución junto con la función a aproximar. Como vemos en la gráfica superior, los puntos "interiores" de cada escalón no son predichos por la red ya que con el criterio min-max no están presentes en la MCP. Estos puntos tienen

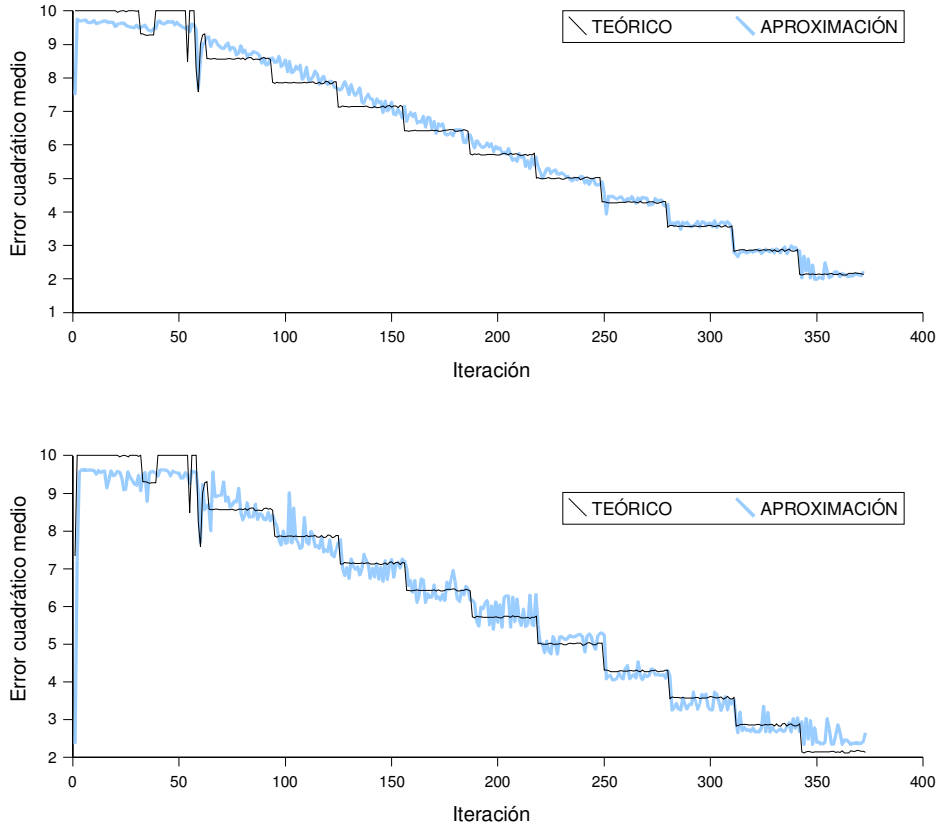


Figura 21. Predicción realizada por la mejor red obtenida en el proceso evolutivo (línea azul) frente a la función real a modelar (línea negra). La gráfica de la inferior muestra el resultado usando una estrategia de reemplazo con criterio funcional y la de la superior sin él.

una probabilidad alta de ser reemplazados según este criterio, porque su distancia a los puntos exteriores de cada escalón es pequeña y además se dan menos veces.

Parece necesaria la inclusión de un criterio de funcionalidad a la hora de asignar probabilidad de reemplazo a las muestras, de modo que aquellas más difíciles de aprender permanezcan más tiempo en la MCP. Para ello, en cada iteración calculamos el error que obtiene el mejor modelo presente en la población en ese instante en la predicción de cada muestra y la etiqueta de todas la muestras se actualiza incluyendo este término de error.

En un caso general donde queremos aproximar una función de n variables $y = f(x_1, x_2, ..., x_n)$, el cálculo del término de funcionalidad sería:

$$F = (y_i - m[x^i_1, x^i_2, ..., x^i_n])^2$$

donde al mejor modelo en una cierta iteración *m,* se le aplican como entradas las variables *x* de cada una de las muestras *i* presentes en la MCP. La predicción *y* que realiza el modelo *m*, se resta del valor real y_i de cada muestra obteniendo un valor de error que elevamos al cuadrado con objeto de ponderar más las peores predicciones.

Las muestras que se predicen mejor tienen más posibilidades de ser reemplazadas porque partimos de la premisa de que la red ya modela esa parte de la función. Aplicando este criterio obtenemos los resultados de la gráfica inferior de la figura 21. Ahora la predicción es mucho mejor en las zonas interiores de los escalones y continúa siendo buena en el resto de la función.

Este nuevo término en el algoritmo de reemplazo no puede estar presente a la hora de decidir si una muestra debe entrar en la MCP, ya que al no estar almacenado no entra en la evolución y no existe una funcionalidad qué probar. Es decir, hasta este instante, en cada iteración la nueva muestra tiene una etiqueta asociada según el criterio min-max y sustituye a otra que sea peor según dicho criterio, si existe. Aquí es donde introducimos el criterio funcional, ya que a la hora de escoger la muestra que debe ser reemplazada añadimos el término funcional al término de distancia. Puede ocurrir que la muestra que tiene un peor valor en distancia no sea reemplazada por tener un error funcional alto, como ocurre en el ejemplo de la figura 21.

Por tanto, el cálculo de las etiquetas asociadas a las muestras quedaría:

$$E = K_d \cdot D + K_f \cdot F$$

$K_d = 1$, $K_f = 0$, para la etiqueta de entrada

$K_d = K_f = 1/2$, para la etiqueta de salida (en estos ejemplos)

donde los factores que acompañan a los términos de distancia y funcionalidad son iguales.

Este criterio utiliza el modelo que se está aprendiendo en tiempo real, por lo que podríamos pensar que, mientras no sea un modelo fiable, el criterio hará asignaciones erróneas y las muestras sustituidas pueden no ser las apropiadas. Pero no debemos olvidar que este criterio se aplica en cada iteración con el modelo más reciente, por lo que las asignaciones de error a las muestras van mejorando al mejorar el modelo, y errores iniciales altos por un modelado pobre se corrigen con el paso del tiempo. Al final, sólo los puntos que son realmente difíciles de aprender tendrán un valor de error alto.

En resumen, hemos añadido un término a la estrategia de reemplazo que tiene en cuenta un criterio funcional con objeto de reforzar el aprendizaje de los puntos con mayor complejidad. Este término añade generalidad a la estrategia porque nos permite trabajar con funciones más complejas. El algoritmo de reemplazo planteado busca ahora tanto los puntos más alejados como los de mayor error en la predicción.

Relevancia inicial

El problema fundamental del criterio de distancia min-max es que tiende a distribuir las muestras en el dominio de definición de la función independientemente de su recorrido. Esto implica que funciones con zonas abruptas no se muestrean de forma óptima, ya que se almacenan por igual puntos de todo el dominio, cuando el óptimo sería guardar más puntos de las zonas más difíciles de modelar. El criterio funcional explicado en el apartado anterior sólo se utiliza para decidir la relevancia de un punto en el proceso de aprendizaje, es decir, la dificultad que dicho punto le representa a las redes del proceso evolutivo. Pero como este criterio no puede ser aplicado a la hora de decidir si un punto debe entrar en la memoria, ocurre que puntos espacialmente poco relevantes pero funcionalmente importantes, no son aceptados en la MCP. Parece que necesitamos un criterio que tenga en cuenta la funcionalidad de un punto antes de ser aprendido.

El mayor problema del criterio de distancia se puede ilustrar con claridad si usamos como función a modelar la campana de Gauss en tres dimensiones que utilizamos al probar la estrategia FIFO y que se muestra en la figura 12. La particularidad de esta función es que contiene muchos puntos en una zona fácilmente modelable (valle) y pocos en la zona de la campana. Además, las distancias máximas en los ejes X e Y son mayores que en el eje Z. La estrategia de reemplazo min-max valorará por igual a los puntos del valle y de la campana en esta función, por lo que la distribución es de esperar que sea homogénea. Pero desde un punto de vista del modelado general de la función, la zona de la campana es muy relevante, harían falta más puntos, mientras que la zona del valle es fácil de modelar, harían falta menos puntos. Este caso se nos puede dar con frecuencia en un problema real porque, como comentamos en el apartado anterior del criterio funcional, en un entorno real aparecen zonas puntualmente singulares con asiduidad. De hecho, podemos pensar en un entorno real análogo a la función representada en la figura 12, una zona llana que tiene un obstáculo en el centro. Como nos podemos imaginar, este tipo de singularidad es muy común y debe ser tratada con profundidad.

En este sentido, proponemos un nuevo término a ser añadido al algoritmo de reemplazo, que es la relevancia inicial de la muestra. Podemos usar el mejor modelo que existe en la población en una iteración determinada y calcular el error con que predice la muestra susceptible de entrar en la MCP. Este error se añade al término de distancia del criterio min-max para decidir si un punto es relevante comparado con los existentes en la MCP. Obviamente, la etiqueta del término a sustituir también debe incluir este error inicial. De acuerdo con el apartado anterior, una vez que se decide que una muestra debe entrar, la selección de la que debe abandonar la MCP se hace teniendo en cuenta además el término funcional. El nuevo término de relevancia inicial cumple además otro cometido básico y es que si una muestra es muy relevante desde el punto de vista de la generalización de la función, pero con el paso del tiempo se aprende muy bien, puede abandonar la MCP porque su término de funcionalidad se hace pequeño. En cambio, este nuevo término de relevancia no cambia y, si una muestra tuvo un error

grande en el momento de entrar en la MCP, dicho error será preponderante a la hora de decidir si debe ser sustituida.

El cálculo del nuevo término sería análogo al de funcionalidad:

$$R = (y_n - m[x_1^n, x_2^n, ..., x_n^n])^2$$

donde el índice n se refiere a la nueva muestra susceptible de entrar en la MCP. Y, consecuentemente, las etiquetas para cada muestra se calculan mediante:

$$E = K_d \cdot D + K_f \cdot F + K_r \cdot R$$

$$K_d = K_r = 1/2,\ K_f = 0, \text{ para la etiqueta de entrada}$$

$$K_d = K_f = K_r = 1/3, \text{ para la etiqueta de salida (en estos ejemplos)}$$

Para comprobar el funcionamiento de la estrategia con este nuevo término, evolucionamos una población de redes con el objetivo de modelar la función de la figura 12. En la figura 22 mostramos la evolución del error cuadrático medio en la predicción de las muestras contenidas en la MCP que realiza la mejor red tras cada iteración para un tamaño de MCP 10, usando el criterio de relevancia inicial junto con el de distancia min-max y usando sólo el de distancia. Como vemos, la evolución en el caso de usar sólo distancia es prácticamente nula y el error no cambia desde las primeras iteraciones. En el caso de usar ambos criterios, el error disminuye con el tiempo de forma casi lineal.

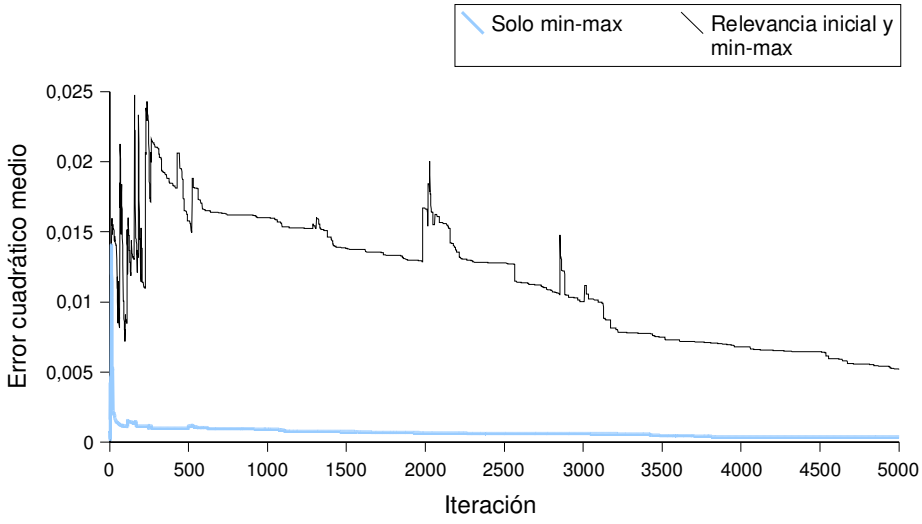


Figura 22. Evolución del error cuadrático medio en la predicción que realiza la mejor red tras cada iteración para un tamaño de MCP de 10 muestras usando el criterio de relevancia inicial junto con el de distancia min-max (linea negra) y usando sólo el de distancia (linea azul).

A la vista de esta gráfica podemos pensar que, con el criterio de distancia, obtenemos un error muy bajo y es una aproximación mejor, pero lo que realmente ocurre es que la predicción de la función global es pobre debido a que la MCP almacena únicamente los primeros puntos que se le presentaron y después no hay reemplazo. Esto es debido a que la distancia respecto a los nuevos puntos no es mejor y no sustituyen a los que entraron

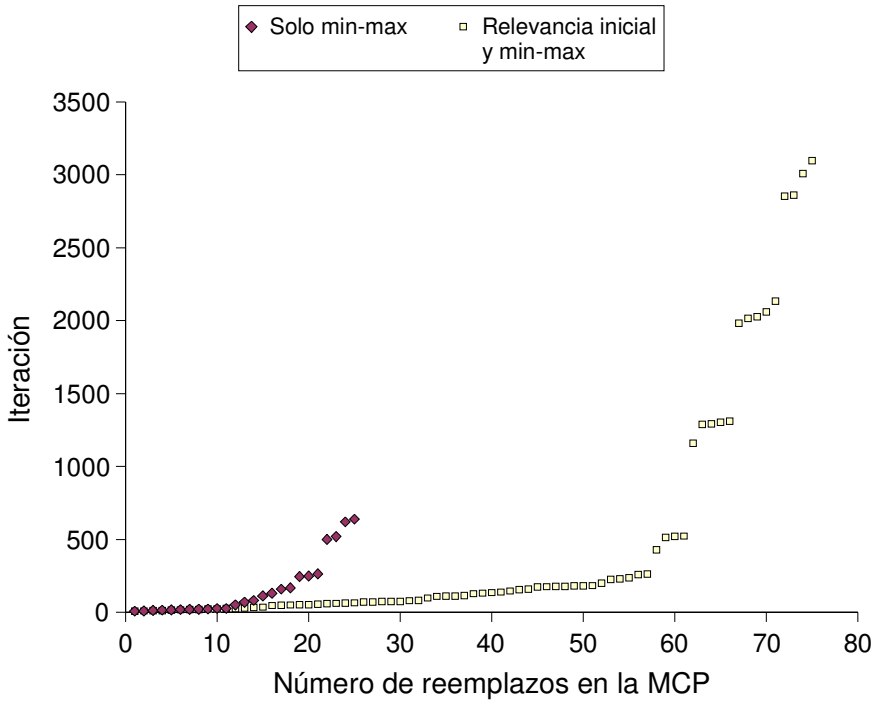


Figura 23. Número de reemplazos que se producen en la MCP frente a la iteración en la que tienen lugar.

al principio. Al añadir el criterio de relevancia inicial, se producen reemplazos durante muchas más iteraciones y esto se refleja en una variación del error.

En la figura 23 se muestra el número de reemplazos realizados en la MCP en ambos casos respecto a la iteración en la que tuvieron lugar. Vemos directamente como, al usar sólo el criterio min-max, se producen exactamente 25 reemplazos y todos ellos en las primeras 640 iteraciones, es decir, en el primer barrido de la función. Al combinar ambos criterios se producen 75 reemplazos, el último en la iteración 3096. Parece claro que, con una configuración estática como la del algoritmo min-max, el error respecto a las muestras contenidas en la MCP es necesariamente bajo porque el algoritmo evolutivo dispone de más de 4000 iteraciones para aprender los mismos 10 puntos.

Siguiendo con el mismo ejemplo, en la gráfica superior de la figura 24 vemos las muestras que se almacenan en la MCP tras 5000 iteraciones si sólo se utiliza el criterio min-max (representados por los puntos sobre la función teórica), y con estos puntos el modelo realiza una predicción de la función gaussiana que mostramos en la gráfica inferior de la figura 24. Esta predicción refleja el hecho de que, aunque el error respecto a la MCP sea muy bajo, el error respecto a la función global puede ser grande. De ahí la gran importancia que tiene que la MCP guarde la información más relevante en cada instante, de modo que el error que se va obteniendo con el algoritmo evolutivo sea reflejo de la mejora en la predicción de la función global. Como vemos, con el criterio de distancia los puntos que se almacenan en la MCP no generalizan la función gaussiana del ejemplo y la predicción que se obtiene de la señal es muy pobre.

CONTENIDO DE LA MCP CON CRITERIO MIN-MAX				
X	**Y**	**Z**	**Etiqueta min-max**	**Iteración de entrada**
-2.700000	-0.300000	1.000000	0.813489	640
-3.000000	3.000000	1.000000	0.776493	247
1.050000	2.700000	1.000000	0.813489	23
2.849999	1.800000	1.000000	0.813489	521
-2.700000	-2.400000	1.000000	0.788241	71
1.949999	-2.700000	1.000000	1.000000	83
-1.800000	1.500000	1.000000	0.776493	24
2.549999	-0.300000	1.000000	0.857493	14
-0.750000	-2.400000	1.000000	0.788241	8
0.150000	0.750000	1.026832	0.844602	9

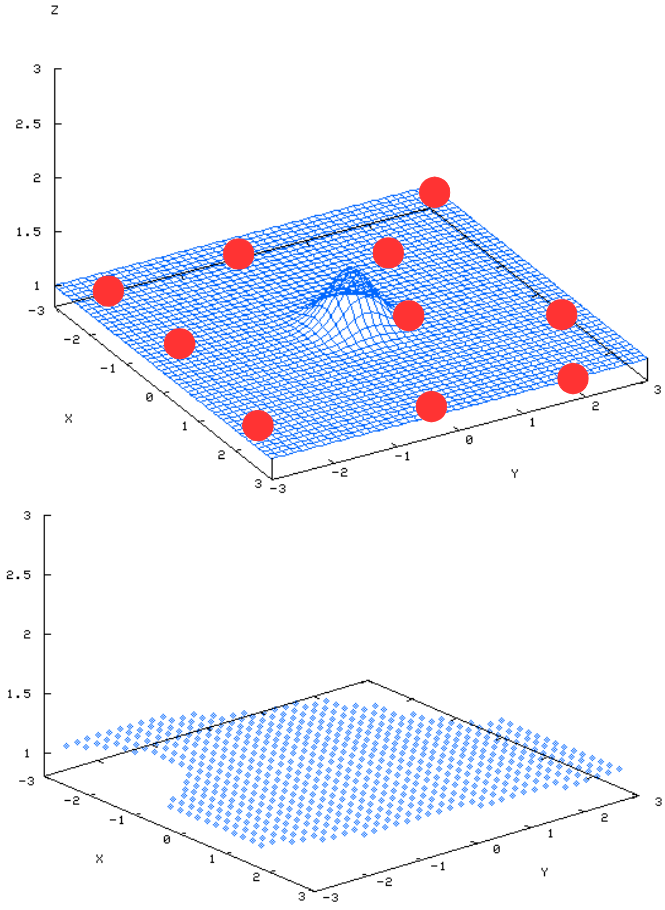


Figura 24. Distribución de muestras final en la MCP (imagen superior) y predicción de la función global (imagen inferior) usando el criterio de distancia min-max en el algoritmo de reemplazo.

Previamente a la figura, presentamos en una tabla los puntos almacenados en la MCP en la iteración 5000 tal y como se utilizan en el MDB. En dicha tabla, además de las coordenadas cartesianas x, y, z de los puntos, mostramos el valor normalizado de la etiqueta que se utiliza en la práctica y la iteración en la que entró cada muestra.

Si analizamos con más detalle la tabla adjunta a la figura 24 y la imagen superior de dicha figura donde mostramos la distribución de los puntos superpuesta a la función teórica, vemos que dichos puntos están bien distribuidos espacialmente en el valle, teniendo en cuenta la función tan particular que estamos usando donde los valores en z son poco relevantes. De modo que el criterio espacial realiza correctamente su función de generalización, pero tiene claras limitaciones que debemos solucionar. La forma final que adquiere la superficie representada en la gráfica inferior de la figura 24, demuestra con claridad la falta de puntos en la zona de la campana, resultando una función prácticamente plana.

En la figura 25 hemos representado resultados análogos a los de figura 24 pero en el caso de usar combinados el criterio de relevancia inicial y el de distancia min-max en el algoritmo de reemplazo de la MCP. Lo primero que se observa es una distribución de muestras menos homogénea en cuanto a su separación espacial pero con más puntos en la zona de la campana, como esperábamos que ocurriera. Hay cinco puntos en el entorno de la campana y cinco en el resto del recorrido de la función. En la tabla que adjuntamos antes de la figura 25, se muestra el contenido de la MCP en la iteración 5000 donde ahora, además de los parámetros que aparecían en el caso anterior, hemos añadido el valor real de relevancia inicial. La columna que representa las etiquetas se utiliza como baremo para decidir qué muestra es reemplazada como comentamos en la sección anterior, por lo que incluye el término de distancia min-max, el de relevancia inicial y el de funcionalidad. La correspondiente etiqueta de entrada no incluye el término de funcionalidad y no se muestra en la tabla porque no es necesario almacenarla en la MCP.

Al estudiar con detalle la tabla, se puede observar que la entrada de los puntos relevantes de la zona de la campana se produce en el primer barrido de la función (iteraciones 232, 261, 264) y no abandonan la MCP hasta el final. Estas muestras dan lugar a que el algoritmo evolutivo obtenga un modelo de la función gaussiana que le permite realizar el seguimiento que vemos en la imagen inferior de la figura 25. Al tener pocos puntos en la zona del valle, la predicción de esta zona es pobre pero, en cambio, la zona de la campana se predice con bastante exactitud.

Debemos recalcar en este punto, que el objetivo del MDB para el cual se realiza este estudio es plantear un esquema cognitivo de alto nivel para un agente autónomo. Por tanto, no podemos buscar una precisión absoluta en el modelado de las funciones sino más bien un modelado globalmente bueno. Bajo esta perspectiva, el modelado que nos ofrece la figura 25 es mucho mejor que el que nos ofrece la figura 24, aunque no sea preciso al 100% en la zona del valle. Además, en este ejemplo simple, el tamaño de la MCP que utilizamos es pequeño con objeto de probar las estrategias de reemplazo en un caso extremo. Si tomamos una MCP de tamaño mucho mayor, incluso sólo con el criterio min-max sería suficiente, pero este razonamiento no es extrapolable a un problema real donde el tamaño de la MCP debería crecer hasta límites inaceptables en términos de tiempo computacional. Es decir, la predicción obtenida no es perfecta porque el número de muestras es pequeño, pero en un caso real vamos a estar en situaciones similares a menudo, y la estrategia de reemplazo cumple los requerimientos de generalización y precisión que demandamos para el MDB.

CONTENIDO DE LA MCP CON CRITERIOS COMBINADOS					
X	**Y**	**Z**	**Etiqueta**	**Iteración de entrada**	**Relevancia inicial**
0.600000	0.900000	1.001440	1.398525	184	0.059482
-1.500000	-2.400000	1.000000	1.081066	176	0.029544
0.600000	-0.000000	1.082650	1.605190	261	0.044729
-2.700000	3.000000	1.000000	0.854799	1983	0.011807
-0.600000	-0.450000	1.030027	1.357477	232	0.059608
-0.750000	0.450000	1.010909	1.261116	111	0.059598
-3.000000	0.600000	1.000000	0.656118	3009	0.002681
-0.000000	0.150000	1.446799	2.015208	264	0.049109
2.250000	2.700000	1.000000	0.710083	2060	0.002431
2.849999	-3.000000	1.000000	1.014099	3096	0.000064

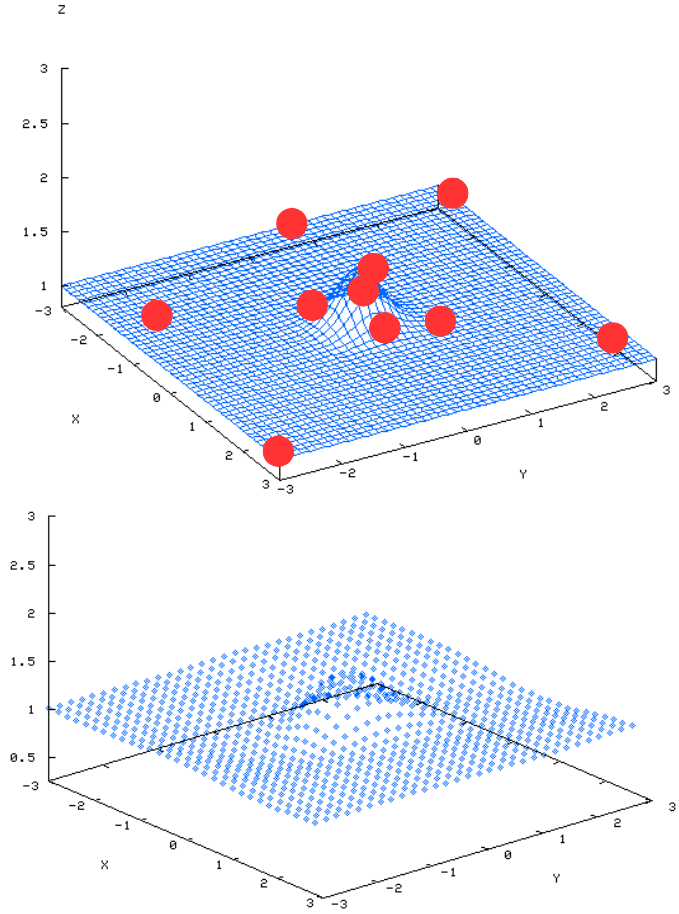


Figura 25. Distribución de muestras final en la MCP (imagen superior) y predicción de la función global (imagen inferior) usando el criterio de distancia min-max y el de relevancia inicial combinados.

Hasta este instante, los datos del muestreo de la función gaussiana se presentaban de forma aleatoria. Pero en un caso real, puede ocurrir que la información aparezca de forma ordenada, y hemos comparado la estrategia de reemplazo con y sin criterio de relevancia inicial también en este caso. Así podemos comprobar el impacto de la entrada de los puntos más relevantes (campana) en instantes de tiempo conocidos. En este caso, realizamos de nuevo un muestreo de 645 puntos de la función de la figura 12, pero barriendo ahora el eje y de -3 a 3 para cada valor x de -3 a 3. En la figura 26 vemos el contenido final de la MCP superpuesto a la función teórica para una evolución que usaba un tamaño de MCP de 14 muestras. Las gráfica de la izquierda corresponde al uso de criterio de distancia únicamente y la de la derecha a los dos criterios unidos. Adjuntamos también una tabla con el contenido real de la MCP en la iteración 5000.

SÓLO MIN-MAX MCP TAMAÑO 14					MIN-MAX Y RELEVANCIA MCP TAMAÑO 14				
X	**Y**	**Z**	**Etiqueta**	**Iteración de entrada**	**X**	**Y**	**Z**	**Etiqueta**	**Iteración de entrada**
-1.500000	0.600000	1.002704	0.796335	2052	-1.500000	-3.000000	1.000000	1.612806	750
2.849999	-0.300000	1.000000	0.796340	1278	-2.100000	-1.800000	1.000000	0.576068	4582
2.849999	3.000000	1.000000	1.000000	644	0.750000	0.600000	1.079013	0.675970	4976
-0.000000	0.900000	1.098949	0.797908	1618	-1.800000	3.000000	1.000000	1.055319	4619
0.150000	2.849999	1.000000	0.830075	1007	1.949999	2.700000	1.000000	0.566176	4450
1.350000	1.800000	1.000020	0.830075	1825	2.549999	-2.700000	1.000000	0.838609	3829
-0.300000	-0.600000	1.203285	0.798184	1554	-0.150000	0.750000	1.155184	0.728439	4815
-1.500000	3.000000	1.000000	0.862498	125	0.600000	2.849999	1.000000	0.710044	4958
-2.100000	-3.000000	1.000000	0.841022	2643	2.250000	0.900000	1.000004	0.639279	4465
1.350000	-0.600000	1.006357	0.796340	1172	0.900000	-1.800000	1.000152	0.495114	4990
1.649999	-3.000000	1.000000	0.978426	1185	0.150000	-0.000000	1.477999	1.224980	4219
-0.000000	-2.100000	1.000074	0.803329	312	1.050000	-0.600000	1.026832	0.500216	3731
-2.700000	-1.500000	1.000000	0.841022	26	0.150000	-0.900000	1.094595	1.485612	4858
-3.000000	0.900000	1.000000	0.796335	13	0.750000	-3.000000	1.000000	0.568310	4959

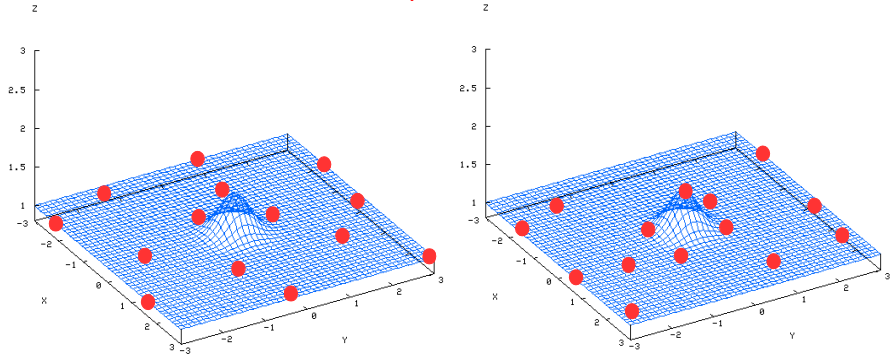


Figura 26. Distribución de las muestras contenidas en una MCP de tamaño 14 usando el criterio de reemplazo min-max (izquierda) y ambos criterios (derecha) con muestreo secuencial de la función test.

La principal característica de estas gráficas es, de nuevo, la distribución de puntos en torno a la campana. Al usar sólo el criterio min-max, hay 3 puntos en dicha zona mientras que al combinar los dos criterios se almacenan 5, dos de los cuales están en la parte más relevante. Con este tipo de barrido secuencial, la entrada de muestras al usar ambos criterios es mucho más periódica y se producen relevos en la MCP hasta el final

(reflejado por la iteración de entrada) debido a que el aprendizaje de la función es más complejo y los modelos cambian durante más tiempo. Como el criterio de relevancia inicial y el funcional dependen del modelo, cuanto más tarde en estabilizarse, más tarda en estabilizarse también la MCP.

SÓLO MIN-MAX MCP TAMAÑO 22					*MIN-MAX Y RELEVANCIA MCP TAMAÑO 22*				
X	**Y**	**Z**	**Etiqueta**	**iteración de entrada**	**X**	**Y**	**Z**	**Etiqueta**	**iteración de entrada**
-0.150000	1.350000	1.000049	0.716860	2883	-0.150000	-0.000000	1.446798	1.717891	295
2.849999	-3.000000	1.000000	0.833333	1914	2.250000	-1.500000	1.000000	0.645082	3812
2.549999	0.600000	1.000000	0.687184	1905	-2.700000	-3.000000	1.000000	1.587224	666
-1.200000	0.600000	1.000062	0.666667	1428	-0.150000	-2.100000	1.000000	0.893420	4155
1.649999	3.000000	1.000000	0.745356	560	0.450000	-0.600000	1.030027	0.814196	4914
2.849999	2.400000	1.000000	0.745356	642	0.300000	-0.900000	1.005554	0.695660	1660
-3.000000	-1.200000	1.000000	1.000000	6	0.900000	-0.150000	1.007784	0.262652	4998
0.450000	-1.800000	1.000000	0.712001	1039	-0.000000	0.600000	1.082650	1.230855	3551
0.900000	-0.600000	1.001440	0.696273	2415	2.849999	3.000000	1.000000	1.017957	3224
1.949999	-1.800000	1.000000	0.745356	2500	-3.000000	-0.600000	1.000000	1.133944	2588
-1.200000	-1.200000	1.000000	0.716069	132	-0.600000	-1.200000	1.000062	0.995424	3432
-3.000000	1.800000	1.000000	0.666667	1306	-0.900000	-0.150000	1.007784	0.753747	804
2.849999	-0.600000	1.000000	0.687184	632	0.300000	0.300000	1.203285	0.773438	2958
-0.300000	-0.300000	1.203284	0.696273	266	0.600000	-2.700000	1.000000	0.898887	418
-2.400000	0.600000	1.000000	0.666667	54	0.450000	0.750000	1.010909	0.662302	4923
1.350000	-3.000000	1.000000	0.745356	519	-0.150000	0.450000	1.162326	0.921480	1588
-0.300000	2.849999	1.000000	0.786165	281	0.300000	-0.000000	1.318814	1.273660	4891
1.350000	0.900000	1.000001	0.687184	532	0.300000	2.849999	1.000000	1.211647	4904
-1.500000	2.100000	1.000000	0.786165	122	-0.000000	0.450000	1.181655	1.148498	3550
-1.200000	-3.000000	1.000000	0.833333	126	-2.100000	3.000000	1.000000	1.016641	3953
-3.000000	3.000000	1.000000	0.666667	20	-1.200000	-3.000000	1.000000	1.389305	3996
-2.700000	-3.000000	1.000000	0.833333	21	2.549999	-3.000000	1.000000	0.729179	3828

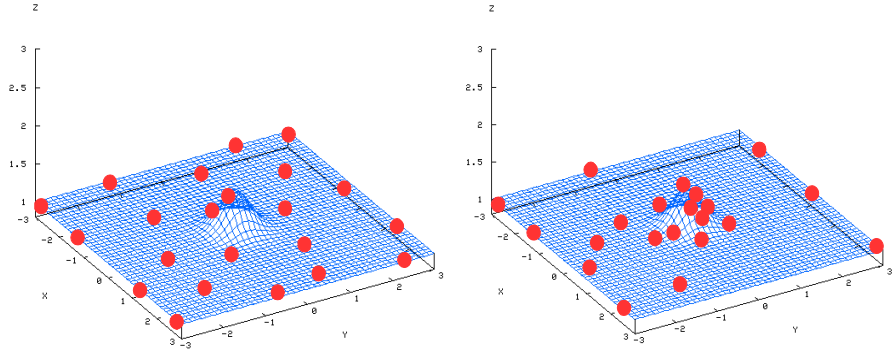


Figura 27. Distribución de las muestras contenidas en una MCP de tamaño 22 usando los criterios de relevancia y min-max con un muestreo secuencial de la función gaussiana.

La figura 27 nos permite ver un caso extremo ya que representa una MCP de tamaño 22 y queda más clara la distribución de puntos en torno a la campana en un caso y otro. Lo que ocurre siempre es que el equiespaciado es mejor al usar el criterio min-max por

separado, aunque con este tamaño de MCP los puntos sean más de los necesarios para representar la función y, aún en el caso de utilizar los criterios combinados, la distribución de puntos por todo el dominio es buena. Adjuntamos previamente a la figura una tabla con el contenido de la MCP en la iteración 5000 en un caso y otro.

Como conclusión a estas pruebas comparativas entre criterios, queda patente la necesidad de incluir el término de relevancia inicial para conseguir la generalización máxima del contenido de la MCP en términos de relevancia funcional. Este término es más necesario cuanto menor sea el número de muestras que almacena la MCP, ya que la necesidad de optimización es mayor.

En los casos prácticos, no se puede optar por una solución que conlleve tamaños de MCP enormes, por lo que se hace necesaria esta mejora. Mejora que va en detrimento del equiespaciado de la muestras, con lo que la conclusión más importante que podemos sacar de este estudio es que los términos que entran en juego en la estrategia de reemplazo no son absolutos sino que, dependiendo de la función que queramos aproximar, cada término debe tener más o menos preponderancia. Así, en el caso de trabajar con una función continua, monótona y lineal, el término de distancia es suficiente para la generalización, por lo que bastaría con ajustar el término de relevancia inicial a cero o un valor bajo.

Al igual que ocurría con el término de funcionalidad, este criterio de relevancia inicial utiliza el modelo que se está aprendiendo en tiempo real. Pero en este caso funciona mejor cuanto mejor sea dicho modelo y, mientras es inadecuado, puede asignar valores incorrectos a las muestras que hace que parezcan muy relevantes cuando no lo son. Este problema no existía en el caso del término funcional porque éste es un valor dinámico que se actualiza en cada iteración.

En el MDB, esto se traduce en que en las primeras iteraciones, mientras se está llenando la MCP, el criterio de relevancia inicial no es fiable y debe estar deshabilitado. El criterio de distancia por el contrario, no depende del modelo sino de las muestras y puede estar presente desde las primeras iteraciones. Por tanto, en un caso real donde se requiera mucha precisión en el comportamiento desde el principio, este criterio debe aparecer sólo tras un periodo inicial de modelado en distancia puro que busque los puntos más generales de la función y después se debe intentar un modelado más preciso.

CONTENIDO DE LA MCP CON MIN-MAX Y RELEVANCIA RETARDADA					
X	**Y**	**Z**	**Etiqueta**	**Iteración de entrada**	**Relevancia inicial**
-0.750000	-0.300000	1.019146	1.419380	1713	0.208686
-0.450000	0.600000	1.030027	2.033223	1653	0.195132
-2.700000	2.100000	1.000000	1.443705	2639	0.000556
-3.000000	-2.100000	1.000000	1.645072	2953	0.001603
1.050000	-3.000000	1.000000	0.876006	4889	0.000004
0.600000	0.750000	1.004964	1.314265	214	0.146888
2.849999	-0.300000	1.000000	1.905056	4883	0.275474
0.000000	0.000000	1.500000	1.213508	411	0.179622
0.300000	-0.600000	1.052700	1.771669	237	0.191696
0.900000	2.549999	1.000000	1.302640	2211	0.159053

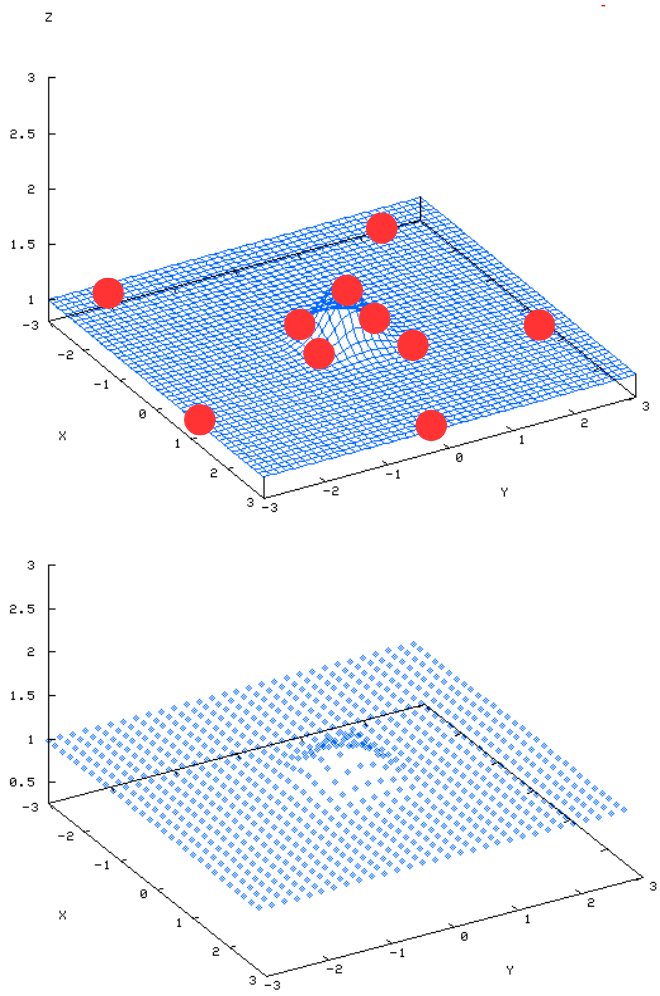


Figura 28. Distribución de muestras final en la MCP (superior) y predicción de la función global (inferior) usando el criterio min-max y el de relevancia inicial pero entrando éste último de forma progresiva.

Hemos comprobado este hecho repitiendo el ejemplo que se muestra en la figura 25 (modelado de la función gaussiana usando un muestreo aleatorio y los criterios de distancia min-max y de relevancia inicial), pero haciendo que el criterio de relevancia vaya entrando de forma paulatina a partir de la iteración 50. Los resultados están en la figura 28. Previamente mostramos una tabla con las muestras contenidas en la MCP, y a continuación la función real y la predicción que realiza el modelo obtenido a partir de estas muestras.

De las múltiples pruebas realizadas con la función test de la figura 12, este es el único caso en que hemos logrado que se almacene en la MCP el punto z=1.5, esto es, el punto más alto de la campana y el más relevante en términos de construir la función con el menor número de puntos y con la mayor exactitud. La distribución espacial es mejor que en el caso de la figura 25 y, además, la predicción es ahora perfecta en la zona de la campana.

Esta es una primera prueba de la necesidad de un planteamiento dinámico de la estrategia de reemplazo para hacer frente a los problemas también dinámicos del modelado que queremos realizar. Los distintos términos que entran en juego en el algoritmo de reemplazo deben ser adaptables a diferentes condiciones y necesidades respondiendo a los cambios posibles de la función a modelar. En un caso con información estática, bastaría con que los términos de la estrategia de reemplazo tuviesen siempre el mismo factor de contribución (fácilmente optimizable).

El término de funcionalidad que estamos utilizando incorpora las características dinámicas del entorno a la estrategia a través del modelo actual que se utiliza en su cálculo.

Conclusiones al estudio de la estrategia de reemplazo

La principal conclusión del estudio sobre la estrategia de reemplazo que hemos planteado, es que es una estrategia dinámica que se puede adaptar a las distintas características de la función a aproximar.

Como resumen, diremos que cuando una muestra es susceptible de entrar en la MCP, calculamos su distancia a las demás mediante el algoritmo min-max, calculamos su relevancia inicial en función del error con que es predicha por el mejor modelo y guardamos la iteración del MDB en la que aparece. De la ponderación adecuada a cada caso de estos tres términos obtenemos la etiqueta que decide si la muestra debe o no entrar en la MCP. El proceso de selección de la muestra que abandona la MCP no cambia e incluye, además de estos tres términos, el de funcionalidad. Es decir:

$$E = K_d \cdot D + K_f \cdot F + K_r \cdot R + K_a \cdot A$$

$$K_d = K_r = K_a = 1/3,\ K_f = 0, \text{ para la etiqueta de entrada}$$

$$K_d = K_f = K_r = K_a = 1/4, \text{ para la etiqueta de salida (en un caso equilibrado)}$$

El valor que toma cada uno de los factores K_i depende de las necesidades del problema, pero podemos dar una visión general de lo que aporta cada término y cuando se debe aplicar:

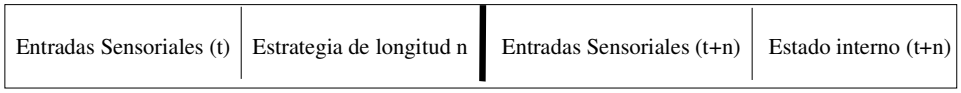
- **Distancia:** si la estrategia de reemplazo sólo incluye el término de distancia, las muestras que almacena estarán bien distribuidas en el dominio y recorrido de la función, pero no se tendrán en cuenta criterios de dificultad en el aprendizaje. Por tanto, las funciones deben ser suaves y monótonas o este criterio individualmente no nos servirá. Pero, por otra parte, es un término muy importante de cara a la generalización (sobre todo en las primeras etapas del proceso) y debería estar siempre presente en mayor o menor medida, pues su inclusión no tiene nunca consecuencias negativas.
- **Funcionalidad y relevancia inicial:** ambos pueden ser tratados dentro de una misma categoría, en el sentido de que aportan un criterio de practicidad a la estrategia: las muestras más difíciles de aprender son más importantes. El término de funcionalidad debería estar siempre presente independientemente del problema, porque en el caso de que todas las muestras sean igual de fáciles o difíciles de aprender, no aporta ningún perjuicio y, en caso contrario, permite aproximar funciones más complejas. El término de relevancia inicial no es importante en funciones simples y bastaría con el de distancia al calcular la etiqueta de entrada (y el de funcionalidad en la de salida). Por otro lado, si sólo

incluimos términos funcionales en la estrategia, aquellas muestras que más difíciles son de aprender serán almacenadas y mantenidas, pero la tendencia del modelo será a aproximar muy bien esas zonas difíciles y no aproximar las zonas fáciles.

- **Antigüedad:** ya hemos visto los resultados que se obtienen usando una estrategia temporal pura FIFO y usando una estrategia sin tiempo en los apartados anteriores. Entre estos dos extremos nos podemos mover libremente al aumentar la preponderancia del término temporal frente a los otros. En aquellos problemas donde se necesite un modelado más local del entorno, sobre todo en funciones muy dinámicas donde no es posible realizar un modelado global, aumentaremos el factor temporal y las muestras más antiguas tenderán a desaparecer. De hecho, en el apartado 6.2.3 estudiaremos la Memoria a Largo Plazo y necesitaremos que el término temporal aumente y disminuya su contribución en la estrategia de reemplazo de forma inmediata, porque tendremos que tratar el caso extremo de cambio completo de la función a modelar. La utilización una estrategia que incluya sólo este término también es posible, y ha sido probada en los ejemplos del apartado 7.1. En realidad, el criterio temporal podría ser utilizado siempre aunque, al contrario de lo que ocurre con el de distancia y con el de funcionalidad, su inclusión puede ser negativa en funciones estáticas al aumentar el número de muestras necesario para el modelado.

La estrategia de reemplazo aplicada al MDB

Al particularizar el planteamiento general de este apartado al MDB, debemos aclarar que nuestro conjunto de muestras se representa por los *pares acción-percepción* introducidos en el apartado 5.2. Dichos *pares acción-percepción* son estructuras más complejas que simples puntos de una función, puesto que guardan información tanto de los modelos de mundo como de los modelos internos. Es decir, de acuerdo con la definición de par acción-percepción, tenemos el siguiente esquema conceptual:



Por una parte, las *entradas sensoriales* en el instante t y la *estrategia* son las variables de entrada de una de las funciones a modelar (el modelo de mundo) y las *entradas sensoriales* en t+n son las variables de salida. Pero además, estas *entradas sensoriales* en t+n son variables de entrada de la otra función a modelar (modelo interno) y el *estado interno* en t+n constituye la variable de salida.

Cada par acción-percepción es, por tanto, una muestra diferente para la obtención de los modelos de mundo y de los modelos internos. Es decir, para un cierto par, las variables asociadas a los modelos de mundo pueden ser muy relevantes para la generalización mientras que las variables asociadas a los modelos internos pueden ser poco relevantes y viceversa. Por este motivo, la MCP utilizada en el MDB guardará pares que se utilizarán únicamente en la evolución de los modelos de mundo y pares que se utilizarán en la evolución de los modelos internos. El tamaño de la MCP debe ser suficiente para que se puedan llevar a cabo ambos modelados.

De esta forma, cada par acción-percepción nuevo que se presenta, tiene en el MDB dos etiquetas asociadas (una para el modelo de mundo y otra para el modelo interno) que se calculan aplicando la expresión general:

$$E = K_d \cdot D + K_f \cdot F + K_r \cdot R + K_a \cdot A$$

con los valores K_i que corresponda. El hecho de tener que usar etiquetas diferentes viene impuesto por la propia estructura del problema: la información que puede ser relevante para una tarea puede no serlo para otra. Además, el cálculo de los términos es diferente en un caso y otro:

- En un caso general, la distancia *D* asociada a una muestra se calcula aplicando:

  $$D = min(d_i)$$

  donde d_i es la distancia entre la representación vectorial de dos muestras contenidas en la MCP. Para los modelos de mundo, esta representación vectorial no debe incluir el *estado interno*, por lo que el cálculo de la distancia entre dos pares acción-percepción será:

  $$d_i = d_i^{s(t)} + d_i^{A} + d_i^{s(t+n)}$$

  que está formado por la suma de las distancias entre variables iguales, es decir, la distancia entre la *sensorización en t*, la distancia entre las *estrategias* y la distancia entre la *sensorización en t+n*.

  Al calcular el término *D* para la evolución de los modelos internos usaremos:

  $$d_i = d_i^{s(t+n)} + d_i^{I(t+n)}$$

  formado por la suma de las distancias entre la *sensorización en t+n* y el *estado interno en t+n*.

  Por tanto, como vemos, la contribución a la etiqueta es diferente en un caso y en otro por estar afectando a variables diferentes.

- En cuanto a los términos de funcionalidad y relevancia inicial, ambos utilizan el mejor modelo del que se dispone en un instante dado. De acuerdo con la notación matemática introducida en el apartado 5.1, el cálculo del término de funcionalidad para los modelos de mundo sería:

  $$F = (s_i(t) - W[s_i(t-1), A_i(t-1)])^2$$

  donde al mejor modelo de mundo en una cierta iteración *W* se le aplican como entradas la sensorización s_i y acción A_i en el instante t-1 de cada una de las muestras *i* presentes en la MCP. La predicción que realiza el modelo *W*, *s(t)*, se resta del valor real $s_i(t)$ de cada muestra obteniendo un valor de error que elevamos al cuadrado con objeto de ponderar más las peores predicciones. Para el término de relevancia inicial:

  $$R = (s_n(t) - W[s_n(t-1), A_n(t-1)])^2$$

  donde el subíndice *n* se refiere a la nueva muestra sobre la que se aplica el modelo de mundo.

  Asimismo, para los modelos internos, a la hora de calcular el término de funcionalidad usaremos:

  $$F = (I_i(t) - I[s_i(t), i_i(t)])^2$$

  que ahora depende de la percepción interna *i(t)* y de la sensorización *s(t)*. Para el término de relevancia inicial:

$$R = (I_n(t) - I[s_n(t), i_n(t)])^2$$

En resumen, al usar los mejores modelos para el cálculo de estos dos términos, en el MDB se hace necesaria la existencia de dos etiquetas.

- Por último, el término temporal no depende del modelo y su cálculo es idéntico para la evolución de los modelos de mundo y para los modelos internos :

$$A = t$$

Por tanto, al aplicar la estrategia de reemplazo al MDB, para cada par acción-percepción susceptible de entrar en la MCP calculamos dos etiquetas diferentes (que pueden usar distintos términos) y realizamos el proceso de reemplazo como hemos establecido en el apartado anterior. Ahora la MCP guarda muestras asociadas a dos funciones diferentes, por lo que la entrada de una muestra relevante, por ejemplo, para el modelo de mundo, debe sustituir a una muestra asociada a dicho modelo y nunca al otro. Como vemos, se trata simplemente de guardar información importante para los dos modelos en la misma memoria y de gestionarla sin mezclas.

Por último, debemos recalcar que los modelos de mundo son más complejos, en general, que los modelos internos. Las características más importantes de la estrategia de reemplazo que hemos desarrollado, como el dinamismo, son consecuencia de la dificultad del modelado de un entorno real. Los modelos internos pueden ser dinámicos (cambio en las motivaciones del comportamiento) y complejos también, por lo que todo el desarrollo realizado es directamente aplicable, pero carecen del ruido y la indeterminación de los modelos de mundo.

En el apartado siguiente, analizamos la interacción que se produce entre el algoritmo PBGA y la MCP.

PBGA y MCP en el mecanismo global

Las evoluciones utilizadas en el apartado dedicado al estudio del PBGA, no se ajustan a la idea de Mecanismo Cognitivo que estamos exponiendo, ya que utilizan una MCP estática que contiene toda la función a aprender. Como hemos dicho en repetidas ocasiones a lo largo de este trabajo, esta opción no es posible en un entorno y en un problema real donde los puntos de la función se conocen en tiempo real. Nos podemos preguntar si el PBGA se comporta igual de bien al no tener todos los puntos disponibles y tener que utilizar una estrategia de reemplazo. La respuesta se obtuvo en el estudio anterior, donde explicamos en profundidad dichas estrategias, ya que todos los resultados que acabamos de mostrar se lograron utilizando el PBGA.

De modo que la interacción entre el PBGA y la estrategia de reemplazo de la MCP es buena, algo que es básico para el buen funcionamiento del MDB, ya que la estrategia se ocupa de que la información que el PBGA debe utilizar sea lo mejor posible y éste de obtener los modelos más adecuados a dicha información. El estudio de las estrategias de reemplazo de la MCP se realizó con independencia del algoritmo usado para obtener los modelos, y el estudio sobre el PBGA tampoco tuvo en cuenta el tipo de muestra ni cómo se obtiene. El buen comportamiento de los algoritmos genéticos que evolucionan redes neuronales al trabajar con muestras también lo posee el PBGA, ya que a la hora del diseño inicial nos basamos en un algoritmo de este tipo.

Existe, sin embargo, una característica del PBGA que modifica el esquema básico del mecanismo en cuanto a la Memoria a Corto Plazo, y es que ahora la población

puede contener individuos diferentes que no tienen por qué utilizar las mismas muestras. Es decir, dado que un modelo obtenido con el PBGA puede ser composición de varios, cada uno afectando a distintas salidas, ¿cuál de ellos se utiliza en la estrategia de reemplazo de la MCP como criterio funcional?. Además, el término de distancia en dicha estrategia trata las muestras como un todo, y ahora ciertas variables de salida pueden no afectar a los modelos. Por tanto, al utilizar el PBGA, se hace necesaria la existencia de una MCP diferente para cada salida con el objetivo de que cada modelo utilice la información más relevante para su evolución.

La estrategia de reemplazo será aplicada a cada nueva muestra tantas veces como salidas tengan las redes, utilizando en cada caso el modelo que corresponda. El término de distancia utilizará únicamente las variables comunes a todas la muestras, es decir, las de entrada y una variable de salida diferente cada vez. De modo que, cada MCP almacenará las muestras más relevantes para una salida determinada que pueden coincidir o no coincidir con las del resto de salidas. Esta característica derivada del PBGA es muy importante porque modifica la estructura básica del MDB.

Para terminar este apartado de enlace entre PBGA y MCP, debemos recordar que el PBGA surgió de la complejidad en la obtención de ciertos modelos de mundo. Está, por tanto, pensado para trabajar con funciones dinámicas y complicadas que son conocidas en tiempo real a base de muestras. Por este motivo, es directamente aplicable a la obtención de los modelos internos aunque no tiene por qué ser necesaria su utilización. Dado que ambos procesos evolutivos (el de los modelos de mundo y de los modelos internos) son independientes y paralelos, los algoritmos que se aplican en cada uno de ellos también lo son. Será recomendable su aplicación a la evolución de los modelos internos cuanto más se parezcan sus características a las de los modelos de mundo en cuanto a dinamicidad y complejidad.

Damos así por finalizado este extenso análisis de la Memoria a Corto Plazo y en el apartado siguientes pasamos a estudiar otro de los elementos básicos del MDB, que es el correspondiente a la Memoria a Largo Plazo (MLP).

6.2.3 Memoria a Largo Plazo (MLP)

Uno de los elementos básicos de cualquier comportamiento considerado inteligente es la memoria, pero no entendida simplemente como un lugar donde almacenar eventos pasados, sino como un sistema de almacenamiento y gestión de la información disponible que haga dicho comportamiento eficiente. Los procesos de aprendizaje autónomo se caracterizan por su alta complejidad, tanto en seres vivos como en agentes artificiales. Por este motivo, resulta crucial la existencia de un sistema que permita gestionar la experiencia adquirida, es decir, recordar situaciones pasadas evitando así un nuevo proceso de aprendizaje. Las soluciones a problemas que se repiten son más fáciles de obtener, más rápidas de aplicar y, en general, mejores (dado que se almacenan soluciones satisfactorias conocidas) en cualquier sistema inteligente que tratemos.

Parece, por tanto, necesario que el MDB también posea esta capacidad de recordar situaciones pasadas y soluciones adecuadas, de modo que en el caso de tener que afrontarlas de nuevo, la respuesta sea inmediata y eficiente. En las primeras versiones del mecanismo, utilizamos únicamente una Memoria a Corto Plazo que guardaba información sobre las condiciones del entorno y del agente en cada instante de tiempo, pero ni los modelos aprendidos ni las condiciones en las que se obtuvieron eran almacenados.

En el diagrama de bloques del MDB presentado en el apartado 6.4, esta memoria se incluía ya a través de los bloques Memoria de Modelos de Mundo, Memoria de Modelos Internos y Memoria de Estrategias. En este apartado, concretaremos el diagrama de bloques un poco más transformando estas tres memorias en una única denominada genéricamente Memoria a Largo Plazo (MLP), que es la encargada de almacenar modelos y situaciones relevantes mientras un agente autónomo lleva a cabo una determinada tarea usando el mecanismo. Se denomina Memoria a Largo Plazo porque la información que almacena se mantiene durante mucho tiempo (incluso podría almacenarse para siempre).

La inclusión de un elemento de memoria en el MDB, implica el desarrollo de un mecanismo de gestión de dicha memoria que controle qué se almacena y qué se elimina por lo que, al igual que en el caso de la MCP, se hace necesaria una estrategia de reemplazo aunque aplicada sobre información muy diferente (no sólo son variables, sino funciones). Debemos desarrollar además una metodología que permita la reutilización automática del contenido de la Memoria a Corto Plazo, acelerando de este modo los procesos de aprendizaje.

Pasamos a explicar con mayor detalle estas ideas básicas con objeto de establecer el funcionamiento general del MDB al incluir este elemento.

Modificaciones al funcionamiento general del MDB

Al trabajar con entornos dinámicos, los modelos de mundo e internos pueden cambiar y, con el paso del tiempo, volver a repetirse. La idea detrás de la inclusión de la Memoria a Largo Plazo en el mecanismo, es que los modelos que permiten que el agente realice una cierta tarea de forma correcta, son susceptibles de ser almacenados para que, si se vuelven a dar las mismas condiciones, puedan ser aplicados evitando así el proceso de reaprendizaje o bien, si las condiciones son similares, puedan ser adaptados de forma inmediata.

Para poder detectar si las condiciones de aprendizaje son las mismas, además de almacenar los mejores modelos, la MLP debe almacenar la información con la que fueron obtenidos dichos modelos, es decir, la Memoria a Corto Plazo asociada. De este modo, seremos capaces de comparar las condiciones actuales con las pasadas y así decidir qué modelo debe ser aplicado.

Hasta ahora hemos contemplado la necesidad de almacenar también las estrategias que resultaron adecuadas, pero éstas, salvo en problemas muy complejos que se escapan al estudio actual, son más fáciles y rápidas de obtener conocidos los modelos, como veremos en los apartados de ejemplo 7.1 y 7.2. En este estudio, por tanto, la MLP no incluirá las estrategias que dieron un buen resultado por lo que el proceso de optimización deberá ser repetido en cada iteración.

Esta simplificación da lugar al diagrama de bloques de la figura 29, donde mostramos la Memoria a Largo Plazo como un único elemento en conexión con las poblaciones de modelos de mundo e internos, pero no con la de estrategias.

El mecanismo de gestión de la MLP es complejo, y será explicado en profundidad a continuación. Nos centramos, por ahora, en los principales cambios que se producen en el mecanismo básico que, de forma esquemática, son los siguientes:

1. Antes de aplicar la estrategia obtenida en cada iteración al entorno, el mecanismo de gestión de la MLP comprueba si el modelo de mundo actual y el modelo interno actual deben ser almacenados en función de algún tipo de criterio funcional y de estabilidad. Como vemos en la figura 29, debe existir una relación directa entre las memorias poblacionales que almacenan los modelos actuales y la MLP.
2. Si un modelo es seleccionado en una cierta iteración del mecanismo para ser almacenado en la MLP, la MCP en dicha iteración también se almacena, por lo que, como se observa en la figura 29, existe comunicación directa entre ambas memorias.
3. En cada nueva iteración, antes de empezar los proceso evolutivos, los modelos almacenados en la MLP se introducen en la población como semillas sustituyendo a los peores individuos, por lo que la conexión entre las memorias poblacionales y la MLP debe ser bidireccional.

Además de estos tres puntos básicos, la inclusión de la Memoria a Largo Plazo modifica el funcionamiento del MDB de un modo más profundo ya que, como veremos en el siguiente apartado, provoca cambios en la estrategia de reemplazo que se utiliza en la MCP.

El primer punto anterior, implica que el mecanismo de gestión de la MLP no se tenga que ejecutar en cada generación del PBGA, sino que los modelos se prueban cada cierto número de generaciones (cada iteración), permitiendo así un aprendizaje más profundo de la MCP. El segundo punto establece uno de los conceptos básicos de la MLP que vamos a utilizar, y es que no sólo nos interesan los modelos adecuados, sino también las condiciones en las que fueron obtenidos. El tercer punto es el más interesante porque en él se toma la decisión de usar la información almacenada en la MLP como semilla, y no directamente. Esto evita la necesidad de un nuevo mecanismo que decida si las condiciones en un cierto instante son repetición de unas pasadas, y en función de esta decisión escoja el modelo que se debe utilizar. Consideraremos que todos los modelos de la MLP pueden aportar algo y entran en la población como nuevos individuos. En

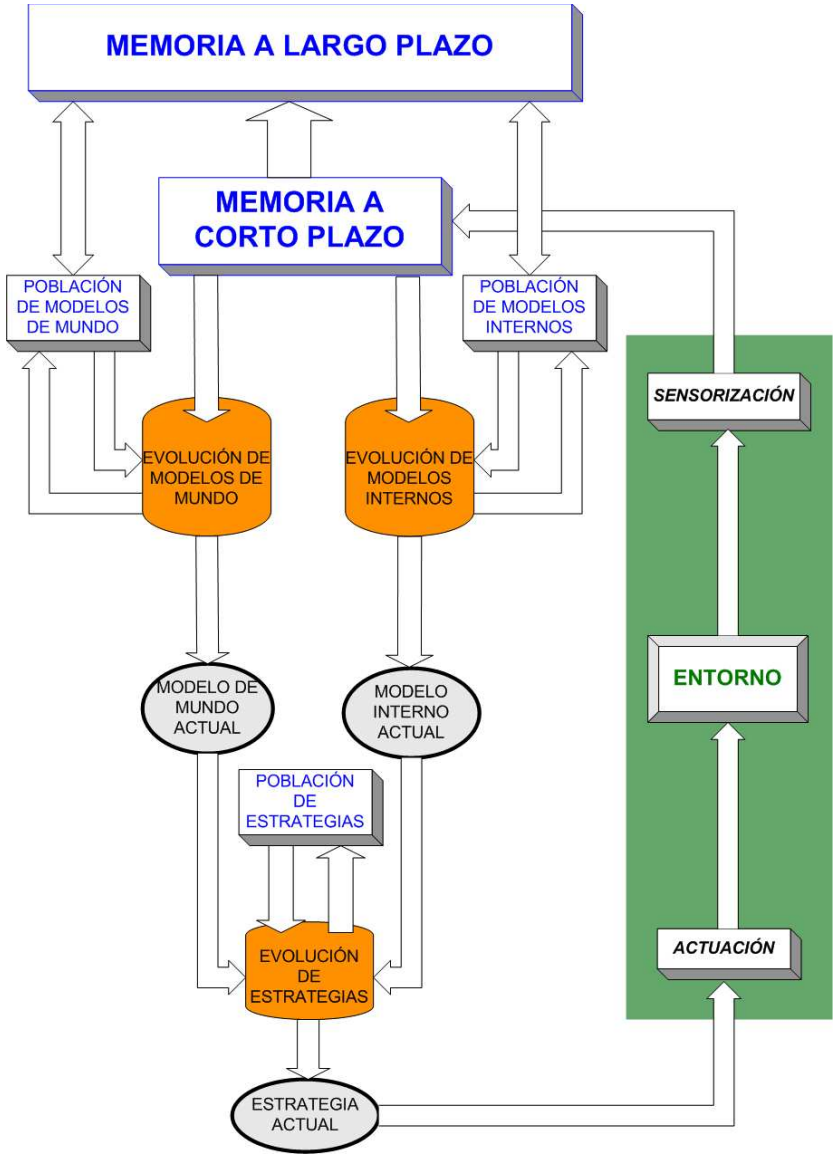


Figura 29. Diagrama de bloques del MDB incluyendo la Memoria a Largo Plazo

caso de que el agente se encuentre con una situación que ya ha aprendido, la MCP actual guardará muestras equivalentes a una de las MCP presentes en MLP (son muestras de la misma función) y uno de los modelos que se inyecta en la población, tendrá de forma inmediata una calidad muy buena y será seleccionado. Lo más interesantes de esta metodología reside en la posible aparición de soluciones (redes neuronales en el PBGA) por combinaciones en los procesos de cruce a partir de los modelos inyectados desde la MLP.

Como los modelos de mundo e internos evolucionan por separado, debemos aplicar el mecanismo de gestión por separado y utilizar también una MLP independiente para cada uno. Esto no cambia el concepto de una única MLP, ya que es equivalente el utilizar una gran memoria global que contenga modelos de mundo e internos mezclados, que dividir esta en submemorias de menor tamaño. A la hora de pasar los modelos a la población, la división es clara ya que afectan a procesos evolutivos diferentes.

Dado que el algoritmo PBGA puede dar lugar a distintos modelos por salida que utilizan distintas MCP, deben existir también distintas MLP por salida. Incidimos de nuevo en la idea de una gran Memoria a Largo Plazo con diferentes partes que afectan a modelos distintos, en este caso submodelos obtenidos con el PBGA. A la hora de pasar los contenidos de la MLP a la población, no existe esta diferenciación por salida y entran todos en la misma, ya que dicha población es única.

La parte más importante de este estudio reside en el diseño e implementación del mecanismo de gestión de la MLP, que pasamos a explicar con detalle en los siguientes apartados.

Mecanismo de gestión de la MLP

A la hora de diseñar el mecanismo de gestión para la Memoria a Corto Plazo, simplemente nos centramos en encontrar los criterios a utilizar para poder comparar las muestras que se presentan al MDB en cada iteración y decidir cuáles son mejores y cuáles peores. Cada una de estas muestras es siempre susceptible de entrar en la MCP, por lo que la estrategia de reemplazo se aplica en cada iteración.

Abordar este problema para la Memoria a Largo Plazo tiene una diferencia fundamental, y es que no todos los modelos que se obtienen en tiempo real son candidatos a entrar en la memoria. Esto es consecuencia de que el objetivo de optimización de la MLP es muy diferente al de la MCP, ya que ahora la información no debe ser relevante desde el punto de vista de la generalización, sino que debe ser información que cumpla dos requisitos básicos: dar lugar a tareas eficientes y ser información estable. Por tanto, previo a la aplicación de una estrategia de reemplazo para la MLP, se ha de comprobar si el modelo candidato a ser almacenado cumple dichos requisitos, es decir, la primera etapa del mecanismo de gestión de la MLP es establecer con claridad un *criterio de acceso*. A continuación, si el modelo es seleccionado, forma parte de una *estrategia de reemplazo* similar a la utilizada en el caso de la MCP, pero con las particularidades de la información que se va a almacenar. En general, se debe plantear una estrategia de reemplazo basada en una serie de términos que permitan comparar unos modelos frente a otros, de modo que la MLP almacene los más relevantes. Como vemos, el planteamiento es análogo al de la MCP y al de cualquier proceso de reemplazo en una memoria.

El mecanismo de gestión planteado hasta el momento y que utiliza un criterio de acceso y una estrategia de reemplazo es el más general posible, y no impone ninguna condición sobre el tipo de modelo que se puede almacenar. A continuación pasamos a detallar las características particulares del criterio de acceso y de la estrategia de reemplazo de la MLP que hemos aplicado.

Criterio de acceso a la MLP

Externamente al MDB, es muy simple decidir si una cierta combinación de modelos y estrategia debe ser almacenada para uso futuro, ya que basta con comprobar si el agente

lleva a cabo de forma satisfactoria una determinada tarea con una cierta estabilidad (lógicamente, si el éxito es puntual, los modelos asociados no son generales). Pero el agente no dispone de esta información de forma directa ya que, de cara a calibrar lo buena que es una estrategia aplicada, únicamente dispone del estado interno predicho. Por tanto, la decisión de si un comportamiento es o no satisfactorio se convierte en una decisión subjetiva del propio agente.

La filosofía que subyace a la inclusión de la MLP en el mecanismo, nos conduce a pretender almacenar en ella todos los modelos que predicen de forma adecuada ciertas partes del entorno y del propio agente (modelos de mundo e internos). Esta decisión de lo que es o no adecuado, puede estar basada únicamente en el error con el que los modelos predicen las muestras contenidas en la MCP. Así, si durante varias iteraciones, los modelos de mundo que se van obteniendo mantienen un valor de error cuadrático medio estable, cualquiera de ellos se convierte en un candidato a entrar en la MLP porque predice con estabilidad una zona del entorno. El problema de utilizar este tipo de criterio de estabilidad es que el error puede ser estable pero a un nivel inaceptable, es decir, asociado a modelados pobres. Debemos asumir que el algoritmo encargado del aprendizaje, el PBGA, es capaz de minimizar el error con el que los modelos predicen las muestras, por lo que el nivel de error (la calidad de la predicción) en una zona estable no forma parte del criterio de entrada en la MLP para el mecanismo de gestión. Esta suposición será confirmada más adelante en los ejemplos prácticos pero, hasta este instante, todas las pruebas realizadas con el PBGA y con el algoritmo genético clásico muestran que el error con que predicen los modelos únicamente se estabiliza en aquellas zonas donde la predicción es buena. Mientras esto no ocurre, el error oscila. Es común en los algoritmos evolutivos, que la calidad no se estabilice hasta una cierta generación y a partir de ese punto ya no cambie de forma significativa. Por tanto, hemos asumido que si detectamos una zona estable en el error, el modelo correspondiente es el mejor posible.

En resumen, los modelos que consideramos susceptibles de entrar en la MLP son aquellos que *mantienen un nivel de error cuadrático medio, respecto a sus correspondientes memorias a corto plazo, estable dentro de un umbral denominado* U_{ee} *(umbral de error de estabilidad) durante un cierto número de iteraciones* U_{ie} *(umbral de iteraciones de estabilidad) del MDB.* En los ejemplos prácticos analizaremos el valor más adecuado para estos umbrales y cómo condicionan el funcionamiento de la MLP.

A efectos prácticos, el criterio de estabilidad desarrollado se aplica en el siguiente orden:

- En las primeras iteraciones del MDB, los modelos presentes en la población no han tenido tiempo suficiente para evolucionar de forma adecuada. Tras esta primera etapa, comenzamos a almacenar en una *memoria temporal* de tamaño *m* el error con que el modelo de mundo e interno actual predice su correspondiente MCP, de modo que podemos establecer un umbral de error de estabilidad U_{ee} en función de la diferencia existente entre los errores almacenados en la *memoria temporal* y los que se van obteniendo. Por ejemplo, podemos asumir que si durante 20 iteraciones (Ui_e = *20)* el error de todos los modelos está por debajo de un 10% (U_{ee} = *10%)*, el modelo es estable. Con este criterio, 21 iteraciones después de empezar a aplicar el mecanismo de gestión, el modelo que se obtiene debe comparar su error con el de los 20 predecesores.

- Esta *memoria temporal* donde se almacenan los errores funciona como una cola FIFO, de modo que el último dato sustituye al más antiguo pero sólo almacena un número fijo de datos.
- Cada nuevo modelo, una vez se ha llenada la *memoria temporal* por primera vez, compara su error con todos los presentes en dicha memoria.
- En caso de detectar estabilidad, el modelo es candidato a entrar en la MLP y su error se almacena en la *memoria temporal.*
- Si no se detecta estabilidad, el modelo actual simplemente se aplica al MDB como de costumbre y su error también se almacena en la *memoria temporal.*

Cuanto menor sea el tamaño de la *memoria temporal*, menos restrictivo es el criterio de estabilidad y más modelos serán candidatos a entrar en la MLP. Por el contrario, si dicho tamaño es excesivo, el criterio es demasiado restrictivo y será muy difícil encontrar modelos estables. El umbral de error de estabilidad U_{ee} y el umbral de iteraciones de estabilidad U_{ie}, pueden permitir que la detección de estabilidad sea poco restrictiva si tienen valores bajos y viceversa.

En general, cuando un modelo es satisfactorio y estable durante muchas iteraciones, será candidato a entrar en la MLP muchas veces. Es decir, este criterio de acceso no tiene en cuenta que el modelo obtenido en distintas iteraciones del MDB pueda ser el mismo, simplemente lo "marca" como susceptible de entrar en la MLP. Será labor de la estrategia de reemplazo el decidir si debe ser o no almacenado. Por el contrario, si tras un periodo de estabilidad el modelo empeora (por ejemplo porque cambia la función a obtener), su error será más alto y el criterio de estabilidad no se satisfará.

Al utilizar un criterio de selección independiente del resultado que proporciona la estrategia que se aplica en el entorno real, corremos el riesgo de almacenar modelos que son estables, pero no relevantes. Es tarea de la estrategia de reemplazo el comparar los modelos entre sí y decidir cuáles permanecen y cuáles se sustituyen de la MLP. Como veremos en los ejemplos prácticos, este tipo de criterio conduce al almacenamiento de modelos intermedios mientras las predicciones no son buenas, porque el error se estabiliza en diferentes partes de la misma función.

Más adelante volveremos sobre el criterio de estabilidad del error, para explicar de forma más concreta los valores reales que hemos utilizado tanto en el tamaño de la memoria temporal como en el umbral de estabilidad. A continuación analizaremos la estrategia de reemplazo de la MLP una vez que un modelo es candidato a ser almacenado.

Estrategia de reemplazo de la MLP

Como explicamos en el estudio de la MCP, cualquier estrategia de reemplazo para una memoria consta de dos etapas, entrada y salida, en las que se decide qué información sustituye a la ya existente. La necesidad de estas dos etapas reside en que las memorias no son infinitas y no es posible almacenar toda la información que se presenta. Pero el caso que nos ocupa de la MLP es especial porque almacena información que pasa por un criterio de selección previo como acabamos de explicar. Por este motivo, ¿hasta qué punto un modelo que ha resultado estable no debe ser almacenado?. Y en cuanto a la etapa de salida, ¿el hecho de eliminar un modelo de la MLP no implica la eliminación de un modelo que resultó estable en el pasado y que no sabemos si volverá a ser necesario en el futuro?. Estas y otras preguntas similares inciden en el hecho de que la

MLP no debería estar limitada en cuanto a tamaño, al menos conceptualmente (existe un límite computacional). Con un criterio de estabilidad adecuado sobre un algoritmo capaz de aproximar la función, todos los modelos seleccionados son importantes y es complicado establecer a priori un grado de importancia que permita eliminar uno u otro de forma selectiva. Lo que sí puede ocurrir es que un modelo estable ya esté almacenado en la MLP, por lo que la estrategia de reemplazo es necesaria para evitar modelos duplicados.

En este sentido, para poder decidir si un modelo es igual o diferente a otro, no podemos realizar una comparación estructural, ya que distintas funciones pueden dar lugar a las mismas salidas. En el caso de utilizar el PBGA, los modelos son redes neuronales, por lo que la comparación estructural no tiene sentido. Aquí reside una de las diferencias fundamentales entre esta estrategia de reemplazo y la diseñada para la MCP, porque ahora para establecer un criterio de reemplazo debemos trabajar con la información fenotípica que proporcionan los modelos y no con la información genotípica directa de las muestras en el caso de la MCP.

La única comparación posible se debe realizar a nivel funcional, es decir, debemos comparar las salidas que proporcionan distintos modelos ante las mismas entradas. Para llevar a cabo esta comparación funcional con los modelos del mecanismo, se deben almacenar en la MLP no sólo el modelo que resulta estable, sino también su Memoria a Corto Plazo asociada. El mejor modelo de cada iteración está optimizado para una MCP (que no cambia durante dicha iteración), por lo que almacenarla implica almacenar las condiciones en las que el modelo resultó satisfactorio. Así, cada modelo en la MLP tiene una MCP asociada que podemos utilizar para compararlo con otros modelos en iguales condiciones. Utilizando esta estrategia, la MLP almacena el número mínimo de modelos estables que ha conseguido aprender.

Hemos organizado la estrategia de reemplazo en 3 etapas de funcionamiento básicas que parten de la selección de un modelo como estable y, por tanto, candidato a entrar en la MLP en una cierta iteración del MDB, y que son las siguientes:

1. Si la MLP está vacía, el modelo entra directamente (con su MCP asociada). Si no está vacía, el modelo se compara, a nivel de respuesta, uno por uno con todos los modelos presentes en la MLP para comprobar si es redundante.
2. Un cierto modelo de la MLP se aplica sobre todas las muestras de su MCP asociada y sobre todas las muestras de la MCP del nuevo modelo (que corresponde a la MCP de la iteración actual del MDB). Del mismo modo, el nuevo modelo se aplica sobre su propia MCP y sobre la del modelo de la MLP escogido. Así conocemos 4 valores de error cuadrático medio (calidades propias y calidades cruzadas) y aplicamos la siguiente regla:
   - Si los dos valores de error del modelo nuevo son mejores que los del existente en la MLP, el nuevo lo sustituye directamente y la estrategia finaliza ya que no es necesario comparar con el resto de modelos en MLP.
   - Por el contrario, si los dos valores de error del modelo nuevo son peores que los del existente en la MLP, el nuevo no entra definitivamente en la MLP porque el existente lo engloba y la estrategia también finaliza.
   - Si ambos modelos son mejores con su MCP asociada pero peores con la que no les corresponde, debemos decidir si las MCP pertenecen a la misma función y, en ese caso, quedarnos con el mejor de los dos modelos. Para ello, establecemos un

umbral de similitud U_s en los errores del modo siguiente: si el error del modelo nuevo con la MCP del modelo ya existente y el error del modelo existente con la MCP del nuevo están dentro de dicho umbral de similitud U_s, entonces consideramos que los modelos corresponden a la misma función, aunque a distintas zonas (cada MCP tiene puntos de distintas zonas). Para decidir qué modelo se queda en la MLP, los aplicamos sobre una MCP que contiene todas las muestras (las del nuevo modelo y las del existente) y nos quedamos con el modelo que proporcione un menor error. De nuevo, la estrategia finaliza.

- Si cada modelo es mejor que el otro con su MCP asociada, pero es peor con la que no le corresponde y no se cumple la condición anterior de similitud, simplemente repetimos el cálculo de los errores cruzados para el siguiente modelo de la MLP.

3. Para finalizar, si tras haber comparado con todos los modelos de la MLP no se cumple ninguno de los criterios anteriores a), b),c) y d), el modelo es almacenado en una nueva posición de la MLP.

Vemos que la estrategia de reemplazo planteada se basa en comparaciones de errores cruzados partiendo siempre de la base de que las muestras que se almacenan en la MCP son una representación general de una función o parte de ella. Esto permite que un modelo que se aplica a una MCP que no es la que le corresponde, proporcione un nivel de error muy diferente al del modelo propio de dicha MCP. En cambio, cuando un modelo se aplica a otras muestras diferentes de las almacenadas en su MCP, pero parte de la misma función, el nivel de error será similar y estaremos en la necesidad de utilizar un umbral de similitud U_s (punto c) para decidir si las MCP son iguales o no. Aquí aparece una de la cuestiones más problemáticas de esta estrategia, y es que una función con muestras muy dispares puede dar lugar a modelos diferentes. Por tanto, el buen comportamiento generalizador de esta estrategia depende de lo relevante y general de las muestras almacenadas en la MCP. Un modelo obtenido con una MCP muy general será capaz de predecir correctamente otra MCP más particular de la misma función, pero no al revés. Este es el primer punto donde vemos que la MCP está directamente relacionada con la eficiencia de la gestión de la MLP.

Si comparamos la estrategia de reemplazo planteada con la utilizada para la Memoria a Corto Plazo en el apartado 6.2.2, se observan grandes diferencias. La más importante consiste en la dificultad de encontrar algún criterio para catalogar los modelos aparte del error que proporcionan. Al igual que en el caso de la MCP, podríamos utilizar un término temporal que añadir al de error de tal forma que los modelo más antiguos tuviesen una mayor probabilidad de ser reemplazados. Como hemos explicado, en principio no vamos a imponer ningún límite superior al tamaño de la MLP, por lo que no es necesario este término de antigüedad. Otra de las diferencias notables con la estrategia de reemplazo de la MCP proviene de nuevo de esta inexistencia de tamaño máximo en la MLP, y es que un modelo que no es similar a ningún otro, es almacenado. El hecho de no imponer un número máximo de modelos únicamente será un problema si el tiempo de vida del agente es muy grande y además con condiciones muy cambiantes. Es decir, la MLP tendrá un tamaño excesivo e ineficiente si el agente está continuamente aprendiendo modelos nuevos, que no es el objetivo del presente trabajo.

Hasta este punto, hemos concretado nuestro planteamiento particular para un esquema de gestión de la MLP como un concepto general y, hasta cierto punto, abstraído de su funcionamiento en el MDB real. El siguiente apartado relaciona de forma detallada la nueva memoria y los problemas para los que ha sido diseñada.

Problemas reales. Inestabilidad

Si un determinado agente que utiliza el MDB se desenvuelve en el mismo entorno estático y con el mismo objetivo durante su tiempo de vida, la utilización de una Memoria a Largo Plazo no es necesaria. Una vez que ha aprendido el modelo de mundo y el modelo interno, como estos no cambian, no tiene sentido el almacenarlos ni para su aplicación directa ni para ser inyectados como semillas en la población.

Es al trabajar en entornos dinámicos donde la presencia de la MLP cobra importancia. Y por entorno dinámico entenderemos tanto un entorno complejo que no puede ser aproximado mediante una única función, como uno que cambia repetidas veces. Desde el punto de vista del agente, tras realizar una tarea en un entorno determinado, podría cambiar a otro distinto e incluso volver al primero. Del mismo modo, al trabajar con motivaciones dinámicas es cuando se hace necesaria una MLP para los modelos internos. En este sentido, el MDB debe ser capaz de recuperar rápidamente un modelo interno ya aprendido para una motivación pasada que vuelve a aparecer.

Tanto el criterio de estabilidad de los modelos como la estrategia de reemplazo de la MLP, son independientes de que el entorno y la motivación sean dinámicos. De hecho, en todo el mecanismo la parte más sensible a estos cambios es la MCP. Y aquí reside el mayor problema de los entornos dinámicos, porque la estrategia de reemplazo de la MCP puede mezclar muestras de funciones diferentes tras producirse un cambio de entorno o motivación. En el estudio de la Memoria a Corto Plazo vimos que, al trabajar con entornos dinámicos, se hacía necesaria la inclusión del término de antigüedad para evitar hasta cierto punto estas mezclas. Al incluir la MLP en el mecanismo, cualquier MCP que contenga muestras incoherentes puede inducir a error y dar lugar al almacenamiento de modelos que corresponden a una parte inestable de la función. Es decir, aunque las muestras que se almacenen sean de funciones diferentes, el PBGA puede llegar a modelos que las aproximen en conjunto con estabilidad en el error (aunque sean modelos pobres). En nuestro concepto de MLP, no consideramos que estos modelos de zonas de transición deban ser almacenados en la MLP, es decir, nos interesan únicamente los modelos que corresponden a zonas estables del entorno o a motivaciones que se mantienen.

Como el criterio de estabilidad no utiliza el nivel de error del modelo (cuán bueno es en la predicción), debemos evitar en lo posible la adquisición de modelos de zonas aparentemente estables pero que corresponden a transiciones de entornos o motivaciones. Como hemos explicado con anterioridad, en el PBGA el error no se estabiliza hasta que una función se ha aprendido bien, pero este comportamiento es independiente de las muestras que utilice. De esta forma, si el contenido de la MCP no es el adecuado, el PBGA tenderá a modelarlo de igual modo y es posible que genere modelos estables de zonas intermedias.

Se hace necesario, pues, el desarrollo de un *criterio de inestabilidad* que detecte de forma automática los cambios tanto en el entorno como en la motivación. Esto requiere que previamente definamos con claridad a qué nos estamos refiriendo al hablar de cambio en los modelos:

> *"Consideramos que existe un cambio del entorno o un cambio en la motivación de un agente que utilice el MDB, si durante un cierto número de iteraciones U_{ii}, las nuevas muestras que se presentan corresponden a funciones diferentes a las almacenadas en la MCP"*

Mientras el PBGA está aprendiendo los modelos, aparecen oscilaciones en el error que no deben ser confundidas por este criterio de inestabilidad con cambios reales. Para un observador externo, es trivial el detectar un cambio brusco en el entorno o en la motivación, pero internamente el agente únicamente puede utilizar la información que posee: las muestras y los modelos que las aproximan. A nivel de muestra no es posible discriminar el cambio, porque puntos iguales pueden pertenecer a distintas funciones. Por tanto, la detección se debe realizar a nivel de los modelos, es decir, analizando cómo cambian éstos a lo largo del tiempo.

Al igual que en el caso del criterio de estabilidad, los cambios en el error cuadrático medio entre la salida que proporciona el mejor modelo y todas las muestras almacenadas en la MCP, indican si los modelos están en proceso de aprendizaje o se han estabilizado o, como pretendemos detectar ahora, están en una zona inestable. Cada muestra nueva que entra en la MCP para una misma función, sólo provoca oscilaciones en el error mientras los modelos no son buenos, es decir, en las primeras iteraciones. Tras este periodo, la entrada de nuevas muestras no provoca cambios significativos en el error, excepto cuando la muestra no se corresponde con el resto de las almacenadas en la MCP. Aunque su influencia en el error final está ponderada por el tamaño de dicha MCP (menor influencia cuanto mayor sea el tamaño), provoca un cambio brusco en el error que aumenta de forma clara, como veremos en las figuras 32 y 34 del apartado dedicado a la aplicación de la MLP. Esta diferencia en el error se produce tanto en un cambio de función (cambio del entorno o de la motivación) como ante la presencia de un punto espúreo en una misma función. Por este motivo, el criterio de inestabilidad debe absorber estos cambios puntuales y sólo activarse en caso de que el cambio en el error se mantenga.

De forma esquemática, el criterio de inestabilidad que hemos desarrollado se aplica del modo siguiente:

- Cada nueva muestra susceptible de entrar en la MCP se aplica en el modelo actual (obtenido en la iteración anterior) tanto para los modelos de mundo como para los modelos internos.
- El error con que dicha muestra es predicha, se compara con el error medio de todos los modelos almacenados en la *memoria temporal* que utilizamos para el criterio de estabilidad (memoria que almacena el error cuadrático medio de cada modelo en las n últimas iteraciones), y si está fuera de un umbral de error de inestabilidad U_{ei} (se escapa del patrón de error de la últimas iteraciones), el modelo se marca como *último modelo estable* e inicia un contador de inestabilidades (que indica que se ha detectado una inestabilidad puntual). Por el contrario si el error para la nueva muestra está dentro del umbral U_{ei} (es similar a la media de la últimas iteraciones), el mecanismo continúa sin cambio.
- Si ha sido detectada una inestabilidad puntual, en la siguiente iteración la nueva muestra se aplica al *último modelo estable* y de nuevo se compara el error obtenido en la predicción con el error medio de la memoria temporal. Si el nuevo error es mayor que el umbral de inestabilidad se continúa el proceso de prueba con el *último modelo estable* y se aumenta el contador de inestabilidades. Si el error es menor que dicho umbral, el contador se vuelve a cero y eliminamos el *último modelo estable*.
- Cuando el contador de inestabilidades llega a un límite que denominamos umbral de iteraciones de inestabilidad U_{ii}, el criterio detecta una inestabilidad global. Una vez detectada, se deben llevar a cabo tres medidas inmediatas:

1. Vaciar las muestras de la MCP mediante la activación de una estrategia temporal pura de tipo FIFO.
2. Detener la aplicación del criterio de estabilidad y, por tanto, de la estrategia de reemplazo de la MLP.
3. Detener la aplicación del criterio de inestabilidad.

- Una vez se ha detectado una inestabilidad global, debe transcurrir un número de iteraciones del MDB igual al tamaño de la MCP para asegurar que en dicha MCP no hay muestras anteriores a la detección de la inestabilidad, con lo que se puede restaurar el funcionamiento normal del mecanismo, es decir:

  1. Se reanuda la estrategia de reemplazo de la MCP con los términos que corresponda.
  2. Se reanuda la aplicación del criterio de estabilidad y, por tanto, de la estrategia de reemplazo de la MLP.
  3. Se reanuda la aplicación del criterio de inestabilidad.

Como se observa, la respuesta global del mecanismo se ve afectada por este criterio al más bajo nivel ya que cambia la forma en que se almacena información en la MCP y, por tanto, cambia lo que los modelos deben aprender.

Los 3 primeros puntos del esquema anterior indican, en primer lugar, que la inestabilidad debe ser detectada antes de que la nueva muestra sea aprendida. En consecuencia, el criterio se aplica tras la adquisición de una nueva muestra independientemente de si entra o no en la MCP. En ese instante, el modelo recién utilizado es la mejor solución que posee el mecanismo y está optimizado para predecir el comportamiento de las muestras almacenadas en la MCP. Por este motivo, puede ser utilizado para detectar si la nueva muestra sigue el patrón de las ya existentes. La *memoria temporal* de errores se utiliza para detectar estabilidades al comparar el error de cada modelo obtenido con el error de los anteriores, por lo que nos sirve también en este caso. Así, comparamos el error con que la nueva muestra es predicha por el modelo actual con los errores contenidos en la *memoria temporal*. Esto nos dirá si el nuevo error es similar o se escapa de la tendencia reciente, que es nuestro objetivo. Aquí es donde se debe definir un umbral de error de inestabilidad U_{ei} , valor que nos indicará cuándo se considera que el nuevo error es muy diferente del resto. Como será explicado en los ejemplos prácticos, la elección de este valor condiciona en gran medida la detección de una inestabilidad y encontrar el más adecuado en todos los casos es una tarea compleja.

Si el error de la nueva muestra es muy diferente de los demás almacenados en la *memoria temporal*, el modelo se guarda como función para el testeo de la inestabilidad y es aplicado de nuevo en la siguiente iteración sobre la siguiente muestra por ser el *último modelo estable* conocido. Por supuesto, la ejecución del MDB continúa normalmente y el modelo que se utiliza en la selección de la estrategia es el obtenido en el proceso evolutivo, no el que se ha guardado. Este *último modelo estable* se aplica hasta que uno de los errores con las nuevas muestras vuelve a estar dentro del umbral de inestabilidad o bien hasta si no lo está durante un número límite de iteraciones denominado umbral de iteraciones de inestabilidad U_{ii} . En caso de que el error esté dentro del umbral de error U_{ei}, el modelo marcado como *último modelo estable* deja de ser necesario y el criterio de inestabilidad en la iteración siguiente utilizará el modelo actual de dicha iteración como antes de la primera detección. Lo único que hemos hecho

ha sido utilizar el último modelo estable mientras no se conocía otro pero si el error vuelve a estar dentro del rango de las últimas iteraciones, el modelo que proporciona dicho error pasa a ser el último modelo estable conocido. Esta forma de proceder nos permite detectar inestabilidades puntuales durante más de una iteración.

Al producirse un cambio en el entorno o en la motivación, las nuevas muestras pueden no ser almacenadas en la MCP según la estrategia de reemplazo utilizada. Por este motivo, la detección de inestabilidad se debe realizar con cada nueva muestra independientemente de si es o no seleccionada para entrar en la MCP.

El umbral de iteraciones de inestabilidad U_{ii} indica cuándo consideramos que la inestabilidad no es puntual, sino que realmente se ha producido un cambio de zona del entorno o bien un cambio de motivación del comportamiento. Cuanto mayor sea este umbral, más difícil será la detección de la inestabilidad con las consiguientes mezclas de información en la MCP. Es, de nuevo, un parámetro crucial a la hora de establecer la sensibilidad del mecanismo a las detecciones y tampoco es nada sencillo de fijar en general como veremos en los ejemplos prácticos.

Cuando se detecta una inestabilidad global, se llevan a cabo tres medidas inmediatas de cara a evitar el almacenamiento de modelos correspondientes a zonas de transición, no deseados en la MLP. La primer es obvia, y consiste en intentar "limpiar" la MCP de muestras de la anterior zona estable. La forma más adecuada de hacer esto es anulando los factores que controlan los términos de distancia, funcionalidad y relevancia inicial en la estrategia de reemplazo de la MCP, dejando únicamente el término temporal. Esto equivale a la utilización de una estrategia temporal pura de tipo FIFO. Este tipo de estrategia permite la eliminación progresiva de las muestras más antiguas y la entrada de las más actuales, aunque sin ningún tipo de criterio funcional o espacial. Una vez que se detecta la inestabilidad, han de transcurrir como mínimo tantas iteraciones como muestras se almacenan en la MCP para asegurar que ha sido "purgada". Las otras dos medidas que se deben llevar a cabo consisten en detener la aplicación del criterio de estabilidad y del criterio de inestabilidad mientras los modelos que se van obteniendo pertenezcan a zonas de transición.

En la práctica, no basta con esperar un número de iteraciones igual al tamaño de la MCP para reanudar el mecanismo de gestión de la MLP, ya que los modelos que se obtienen en esa fase se han obtenido con memorias a corto plazo que contenían muestras mezcladas. No es hasta que han transcurrido tantas iteraciones como el doble del tamaño de la MCP, cuando podemos asegurar que los modelos son totalmente adecuados a la nueva zona, y no a la de transición. En este punto, se restauran los factores para los términos de la MCP y se reanuda la aplicación de los criterios de estabilidad e inestabilidad.

El planteamiento del mecanismo de gestión de la MLP que hemos hecho, lleva al almacenamiento de todos los modelos que son estables independientemente de si el entorno o la motivación cambian totalmente o simplemente responden a modelos difíciles de aproximar con una única función. Aquí entra la gran relevancia del mecanismo de detección de inestabilidades, que es el responsable de que, por ejemplo, una zona del entorno requiera un único modelo más complejo o varios simples.

A continuación, pasamos a analizar los resultados experimentales obtenidos con este mecanismo de gestión de la MLP, exponiendo previamente las particularidades adoptadas en la implementación práctica del mismo.

Aplicación práctica del mecanismo de gestión

La primera particularidad del experimento realizado reside en la utilización de los modelos de mundo únicamente en todo el estudio por ser, en general, más complejos. Pero, al igual que en estudios anteriores donde se ha realizado esta simplificación, la generalización de los resultados obtenidos a los modelos internos es inmediata.

Antes de mostrar los experimentos concretos que se han llevado a cabo, hemos tomado una serie de decisiones en la implementación real de la MLP que debemos exponer. La primera y más importante está relacionada con la estrategia de reemplazo de la MCP que hemos utilizado. No queremos que el proceso de obtención de los modelos se vea limitado a zonas reducidas de las funciones motivado por la presencia en la MCP de muestras demasiado recientes. Es decir, pretendemos que los modelos almacenados en la MLP sean lo más generales posibles y que únicamente entren nuevos si realmente se produce un cambio relevante en la función a aproximar. Por tanto, no incluiremos el término temporal en el cálculo de las etiquetas para las muestras de la MCP ya que dicho término conlleva el almacenamiento de muestras locales y condiciona la permanencia de las más generales. La expresión a utilizar quedaría:

$$E = K_d \cdot D + K_f \cdot F + K_r \cdot R + K_a \cdot A \quad \text{con } K_a = 0$$

En el apartado 6.2.2 se concluyó la importancia de este término temporal en entornos dinámicos, dada la gran dificultad del modelado a partir de muestras muy dispares, como las que se almacenan en la MCP para este tipo de entornos. Pero al utilizar el mecanismo de gestión de la MLP expuesto en los apartados anteriores, evitamos la mezcla de muestras en la MCP con la ejecución del criterio de inestabilidad, por lo que el término temporal en la estrategia de reemplazo de la MCP no es imprescindible. En resumen, los modelos que se van obteniendo utilizan siempre muestras lo más generales y relevantes posibles y, si se detecta una inestabilidad, la estrategia de reemplazo de la MCP simplemente anula los 3 factores K_d, K_f y K_r de la expresión anterior para el cálculo de la etiqueta y establece $K_a = 1$, con lo que se aplica una estrategia de tipo FIFO. Así aseguramos que, entre inestabilidades, los modelos que se obtienen predicen globalmente cada zona estable.

A la hora de implementar en la práctica la MLP, nos encontramos con que el tamaño de la MCP (el número de muestras que utilizamos para aproximar los modelos de mundo e interno) es una referencia muy útil a la hora de fijar los umbrales que entran en juego en el mecanismo de gestión. Por este motivo, no referiremos a partir de ahora a dicho tamaño como *M*.

Cada una de las tres partes que componen el mecanismo de gestión de la MLP que hemos desarrollado, tiene una serie de características de implementación propias que pasamos a analizar:

1. **Criterio de estabilidad:** la ejecución de este criterio comienza tras 2*M* iteraciones del MDB, ya que la *memoria temporal* que almacena los errores es de tamaño *M* y no se empieza a llenar hasta que han transcurrido *M* iteraciones (no utilizamos los errores mientras la MCP se está llenando). El umbral de error de estabilidad U_{ee} utilizado ha sido de un 20% y el umbral de iteraciones de estabilidad U_{ie} tiene un valor igual al tamaño de la MCP *M*. Esto implica que los errores que proporcionan los modelos que se van obteniendo se deben mantener en un intervalo relativo de un 20% durante *M* iteraciones consecutivas para que el último modelo sea candidato a entrar en la MLP.

2. **Estrategia de reemplazo:** hemos aplicado un umbral de similitud *Us* de un 30% para los errores cruzados que se calculan en esta etapa.
3. **Criterio de inestabilidad:** la ejecución de este criterio comienza, al igual que en el de estabilidad, tras 2*M* iteraciones del MDB una vez que la *memoria temporal* está llena. El umbral de error de inestabilidad U_{ei} utilizado ha sido de un 30% y el umbral de iteraciones de inestabilidad U_{ii} tiene un valor de $M/2$, por lo que este criterio es menos restrictivo que el de estabilidad. Además, para evitar que durante el cambio de entorno un modelo proporcione un error estable de nuevo de forma puntual y el criterio de inestabilidad no se active, realizamos un promedio de los errores antes de compararlos. Es decir, el umbral U_{ei} no se aplica directamente al error del *último modelo estable* sino que utilizamos el error promedio de las 4 últimas muestras para evitar que una de ellas sea muy bien predicha por el *último modelo estable* de forma casual. Por último, una vez que se detecta una inestabilidad esperamos 2*M* iteraciones hasta volver a aplicar el algoritmo de reemplazo de la MCP (con los términos de distancia, funcionalidad y relevancia inicial) y los criterios de estabilidad e inestabilidad.

Todo el desarrollo llevado a cabo en este apartado no sirve de nada si los modelos que residen en la Memoria a Largo Plazo no son convenientemente utilizados en el proceso de evolución. Como ya hemos explicado al principio de este apartado, hemos decidido que dichos modelos sean inyectados en las poblaciones de los algoritmos evolutivos, con el objetivo primordial de acelerar el proceso de aprendizaje. Al utilizar el algoritmo PBGA para la obtención de los modelos, estamos favoreciendo la existencia de individuos con partes (neuronas) dedicadas a tareas específicas, de tal forma que los modelos que se introducen en la población desde la MLP, pueden contener estas partes que son útiles al combinarlas con otras. Al aplicar el operador de cruce del PBGA, es posible que incluso una capa de neuronas entera pase de un individuo a otro.

Además de acelerar el aprendizaje mediante combinaciones de módulos neuronales básicos, es posible que el operador de mutación aplicado sobre los modelos que provienen de la MLP lleve a modificaciones de éstos que también favorezcan el aprendizaje. Estamos introduciendo así un aprendizaje en distintas escalas de tiempo, y no únicamente en tiempo real, que es lo que ocurre si no se utiliza Memoria a Largo Plazo.

La inyección de los modelos desde la Memoria a Largo Plazo implica la sustitución de los peores individuos de la población en cada iteración. Obviamente, para que este proceso tenga lugar, es necesaria la existencia de algún modelo en la MLP, por lo que nunca se podrá dar antes de 2*M* iteraciones del MDB puesto que el criterio de estabilidad no se aplica antes.

En el siguiente apartado, estudiaremos el comportamiento del MDB con y sin MLP ante un problema teórico de cambio de modelo de mundo simulando así un entorno dinámico.

Resultados experimentales

La introducción de la Memoria a Largo Plazo en el MDB y el consecuente desarrollo de un mecanismo de gestión, tienen su origen en la complejidad de los entornos dinámicos reales para los que el MDB ha sido diseñado. Dicha complejidad se manifiesta, sobre todo, a la hora de aprender los modelos de mundo, por lo que la reutilización de

información previamente aprendida es fundamental. Bajo estas premisas, las pruebas prácticas de funcionamiento de la MLP deben contemplar cambios profundos en los modelos de mundo. Así, hemos utilizado, en primer lugar, funciones teóricas como modelos a aproximar con objeto de poder extraer conclusiones al utilizar funciones perfectamente conocidas y, en un segundo experimento, hemos utilizado un simulador con el robot real Pioneer 2.

Funciones teóricas

Al igual que en el estudio de la estrategia de reemplazo de la MCP, el objetivo de las pruebas que vamos a llevar a cabo consiste en aproximar una función teórica que se conoce a partir de muestras, de forma análoga a la evolución de los modelos de mundo en el MDB. Dicha función se muestra en la figura 30 y representa una campana de Gauss en tres dimensiones. Es muy similar a la utilizada en el estudio de la MCP, con la diferencia de que ahora la amplitud es mayor, de modo que el rango es el mismo en las tres variables. El muestreo que hemos realizado contiene 440 puntos que son susceptibles de entrar en la MCP. Como algoritmo para llevar a cabo la aproximación utilizamos el algoritmo PBGA con redes de dos entradas y una salida (los parámetros concretos de cada evolución pueden ser consultados en el Apéndice B) y el mecanismo de gestión de la MLP con las características explicadas en el apartado anterior.

Tras 440 iteraciones (todas las muestras han sido probadas) con esta función, la cambiamos por otra diferente que mostramos en la figura 31 y que es una función senoidal en 3 dimensiones. La población en la iteración 441 habrá convergido hacia la función gaussiana, por lo que cualquier modelo que se pruebe con las muestras de la función senoidal es de esperar que proporcione un error grande. De nuevo, hemos muestreado esta función en 440 puntos, por lo que la mantenemos como objetivo

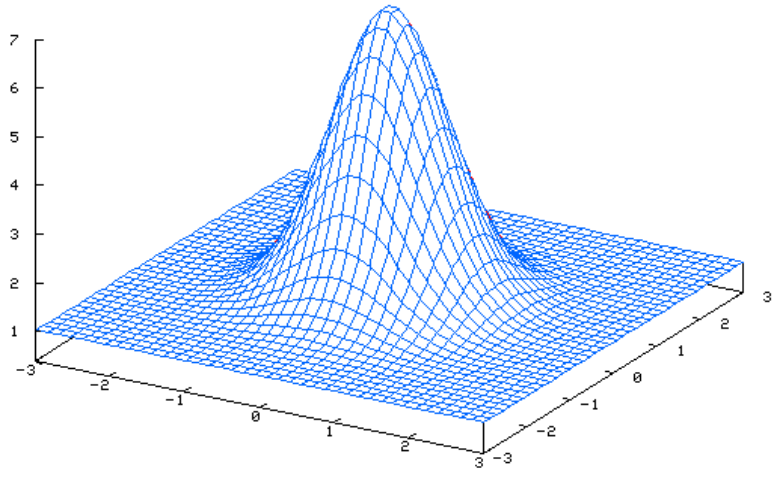


Figura 30. Campana de Gauss $z = 6*exp(-(x^2+y^2))+1$ utilizada para probar el mecanismo de gestión de la MLP.

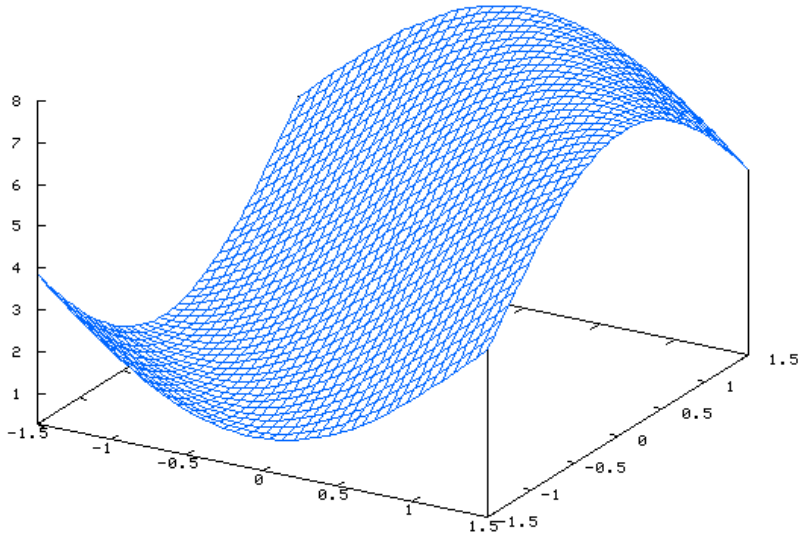


*Figura 31. Función senoidal z = 3.25*sen(x+y) + 4.25 utilizada para probar el funcionamiento del mecanismo de gestión de la MLP*

durante otras 440 iteraciones y, en ese instante, volvemos a utilizar la función gaussiana, después la senoidal y así sucesivamente cambiando entre las dos funciones cada 440 iteraciones.

Esto nos permite simular una situación real donde el MDB se encuentra con un entorno cíclico de tal forma que, una vez ha aprendido cada una de las zonas, la respuesta debería ser inmediata. En cambio, en una evolución análoga pero sin MLP, cada cambio de función debería provocar mezclas de muestras diferentes en la MCP con consecuencias nefastas para la obtención de modelos adecuados. El resultado obtenido se observa en la figura 32 donde hemos representado el error cuadrático medio entre las predicciones del mejor modelo de mundo en cada iteración y las muestras contenidas en la MCP. La evolución se ha prolongado durante 4000 iteraciones, por lo que se han realizado 8 cambios de función.

La gráfica superior de la figura 32 corresponde a la evolución sin MLP y, como se observa con claridad, durante la primeras 440 iteraciones el error disminuye de forma usual al utilizar el PBGA hasta un nivel de 0.06 aproximadamente. En cuanto se produce el cambio de función gaussiana por senoidal, el error aumenta de forma abrupta hasta 0.45, por lo que es obvio que la nueva muestra ha entrado en la MCP. Desde ese instante hasta la iteración 880, el error disminuye aunque se queda en un valor de 0.3 muy por encima del logrado con la función gaussiana. En los subsiguientes cambios de función (claramente diferenciables por el gran aumento en el valor de error) la mejora del error es menor cada vez y, dentro de cada zona, la evolución no es satisfactoria. Como hemos explicado anteriormente, este comportamiento es debido a la mezcla de muestras de distintas funciones que se produce en la MCP. La solución a este problema consiste en aumentar el factor que controla el término temporal en la estrategia de reemplazo de la MCP. De esta forma, las muestras más antiguas son reemplazadas

evitando así que en una zona haya muestras de la anterior. Pero el ajuste de este factor no es simple, dado que se ha de asegurar que las muestras caduquen antes de 440 iteraciones, algo que sólo se puede afirmar si la estrategia es de tipo FIFO. De otro modo, el resto de términos en la estrategia de reemplazo pueden mantener muestras muy relevantes aunque sean antiguas. Debemos recalcar que la existencia de una única muestra que se corresponda con la zona en la que está el agente, condiciona toda la evolución y no es deseable en absoluto.

Por tanto, la solución al no utilizar MLP es aplicar una estrategia reemplazo de tipo FIFO para la MCP, con lo que el aprendizaje se vuelve muy local y, como principal inconveniente, ha de ser realizado de nuevo cada vez que se cambia la función. De hecho, esta fue la solución aplicada en las primeras versiones del MDB pero, con la estrategia de reemplazo para la MCP planteada en el apartado anterior, el no utilizar una MLP implica que el mecanismo no pueda tratar con entornos dinámicos de forma eficiente.

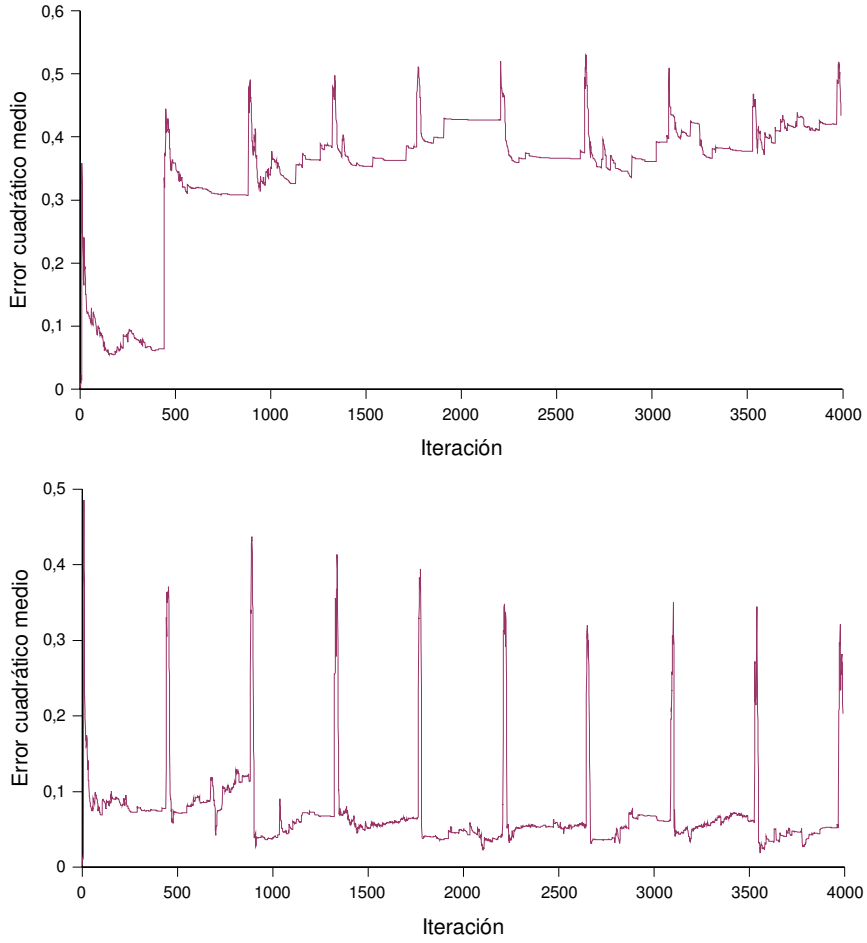


Figura 32. Error cuadrático medio para las predicciones del mejor modelo de mundo en cada iteración sobre las muestras contenidas en la MCP. En la gráfica superior no se ha utilizado MLP y en la inferior si.

En la gráfica inferior de la figura 32 hemos aplicado el mecanismo de gestión de la MLP expuesto en los apartados anteriores y, como vemos, el comportamiento global es totalmente diferente aunque, hasta la iteración 440, es muy parecido a la gráfica superior. El nivel de error alcanzado ahora está en torno a 0.8 y, de nuevo, se produce un aumento brusco del mismo al cambiar a la función senoidal en la iteración 440. Pero la diferencia fundamental reside en la respuesta del mecanismo al cambio de función, ya que al ser detectado el aumento continuado en el error, el criterio de inestabilidad se activa y la MCP se vacía de forma gradual. En este ejemplo, hemos utilizado una MCP de tamaño 20, por lo que la inestabilidad se detecta tras 5 iteraciones consecutivas con aumento del error $(M/4)$.

Si observamos con detalle la gráfica inferior de la figura 32, vemos cómo el error siempre disminuye bruscamente al cambiar de zona y después aumenta hasta estabilizarse en un nivel muy similar al obtenido en dicha zona con anterioridad. Esto se debe a que, hasta que han transcurrido 40 iteraciones en una zona estable, el PBGA debe aproximar una MCP de tipo FIFO y, a partir de esa iteración, se reanuda la estrategia de reemplazo con el consiguiente cambio en la información que se debe aprender. Los modelos almacenados en la MLP y que son inyectados en la población, resultan adecuados para las muestras que guarda la MCP tras un cambio de zona y proporcionan un error bajo, pero a medida que la MCP se llena con las muestras más relevantes de una determinada zona, dichos modelos proporcionan, lógicamente, un error similar al que obtuvieron cuando fueron almacenados en la MLP.

Para este experimento en concreto, en la iteración 140 se almacenó el primer modelo de mundo en la MLP y, en sucesivas iteraciones, se produjeron nuevos modelos estables que resultaron peores y también mejores. De hecho, el modelo definitivo en esta primera zona se obtiene en la iteración 438. Tras el primer cambio de zona, se produce una nueva entrada en la MLP en la iteración 501 y una segunda en la iteración 613. Es decir, la estrategia de reemplazo con los parámetros utilizados, no detecta que estos dos modelos sean iguales y los almacena en la MLP. En realidad, son dos modelos similares pero sus MCP asociadas corresponden a dos zonas diferentes de la función senoidal y para la estrategia de reemplazo esto es análogo a tener entornos diferentes. Este problema del almacenamiento de un número mayor de modelos del estrictamente necesario, puede ser solventado utilizando una MCP de mayor tamaño, de modo que nunca se puedan almacenar muestras muy dispares de una misma función. Además, si el umbral de similitud U_s del 30% que hemos aplicado se aumenta, también se favorece el almacenamiento de un menor número de modelos.

En resumen, el mecanismo de gestión de la MLP desarrollado proporciona unos primeros resultados muy prometedores, y es capaz de optimizar el funcionamiento del mecanismo ante un cambio brusco en el entorno y futuras recuperaciones del entorno original. La MLP almacena todos los modelos que resultan estables y su introducción como nuevos individuos en la población permite una respuesta rápida con un error bajo, agilizando el proceso de aprendizaje. Además, este planteamiento de gestión de la MLP evita la utilización de una estrategia de reemplazo de tipo FIFO en la MCP, incluso evita la necesidad del término temporal, de modo que se favorece que la información almacenada en la MCP sea la más general y relevante.

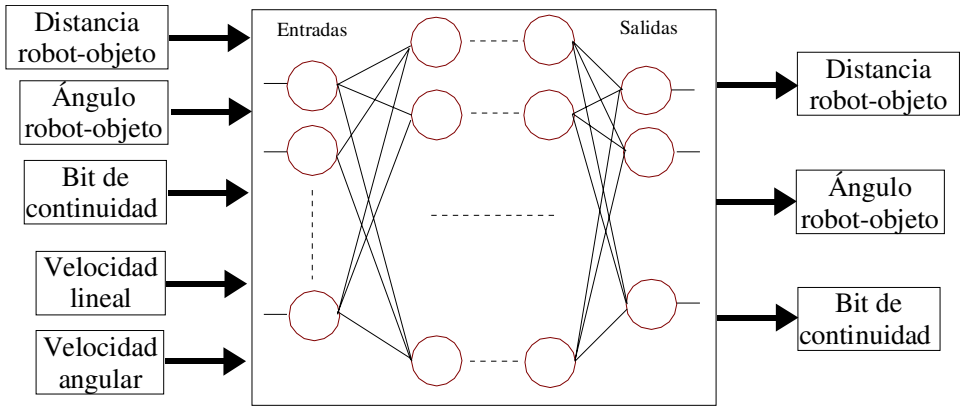
Probaremos a continuación la MLP en un problema más real, utilizando para ello el robot Pioneer 2 que será presentado en el apartado 7.2.1.

Caso real

En el ejemplo anterior, no se ha utilizado el MDB completo sino simplemente la parte correspondiente a la evolución de los modelos de mundo. Una vez comprobado el funcionamiento general de la MLP, pasamos a aplicarlo a un problema más cercano al objetivo final. Para ello, hemos utilizado un modelo en simulación del robot Pioneer 2 que será aplicado también en el ejemplo del apartado 7.2. La tarea a realizar es simplemente la captura de un objeto que se sitúa en su cercanía, es decir, el robot debe aprender por sí mismo a controlar las acciones que ejerce sobre las ruedas para alcanzar el objeto.

La figura 33 muestra de forma esquemática el montaje experimental a vista de planta. El objeto de forma cilíndrica debe ser detectado por medio de los 16 sensores sónar que dispone el robot y, aplicando acciones sobre las ruedas, debe girar y avanzar hacia él minimizando la distancia. El objeto se sitúa siempre en el sector marcado en color oscuro en la figura 33, tanto en la iteración inicial como cada vez que lo alcanza. La posición inicial del robot es cualquiera dentro de la circunferencia interior del sector, pero únicamente se devuelve al robot a dicha posición si alcanza el objeto. Por tanto, el experimento es simple y analizaremos los resultados únicamente en el marco de la MLP.

El modelo de mundo que vamos a utilizar posee 5 entradas y 3 salidas, que corresponden a 3 valores de sensorización aportados por los sensores sónar tanto en la entrada como en la salida predicha, y dos valores de actuación que corresponden a la velocidad angular y a la velocidad lineal de los motores. Para la evolución se ha aplicado el algoritmo PBGA sobre redes de 5 entradas y 3 salidas como máximo. Este modelo de mundo se puede representar de forma esquemática:



Las variables utilizadas vienen impuestas por las especiales características del robot Pioneer 2. Con los sensores sónar podemos saber en todo instante la menor distancia al objeto, ya que su rango de funcionamiento impide que sea detectado por más de un sensor al mismo tiempo. De este modo, conocemos la distancia a la que se encuentra el objeto y qué sensor la detecta, por lo que podemos conocer de forma directa el ángulo que forma el objeto respecto a la parte frontal del robot. Estas son las dos primeras entradas sensoriales al modelo de mundo del esquema anterior. La tercera entrada sensorial es necesaria porque existe una discontinuidad en los ángulos que impide que las redes del PBGA los puedan predecir. Si tomamos como origen de ángulos la parte

frontal del Pioneer 2, también reside en la parte frontal el ángulo de valor 360º, de modo que dos giros opuestos conducen a la misma percepción. Para contrarrestar esta ambigüedad, incluimos un bit que indica si el ángulo se encuentra a la derecha o a la izquierda de la zona frontal del robot. Estas 3 variables conforman la sensorización del robot y, como en todo modelo de mundo, son las variables que deben ser predichas.

Las acciones que hemos utilizado son la velocidad lineal y angular con que se mueve el robot. El tiempo de ejecución de estas acciones ha sido ajustado para conseguir que el robot siempre pueda alcanzar el objeto si la velocidad es máxima. Estas acciones son obtenidas mediante un proceso evolutivo que no utiliza el PBGA porque los cromosomas de los individuos son directamente las velocidades.

En este ejemplo, la motivación del comportamiento es alcanzar el objeto, por lo que el estado interno será mejor cuanto menor sea la distancia entre el robot y el objeto. De modo que no es necesaria la evolución de un modelo interno independiente y el valor de estado interno viene dado directamente por la distancia robot-objeto que proporciona el modelo de mundo.

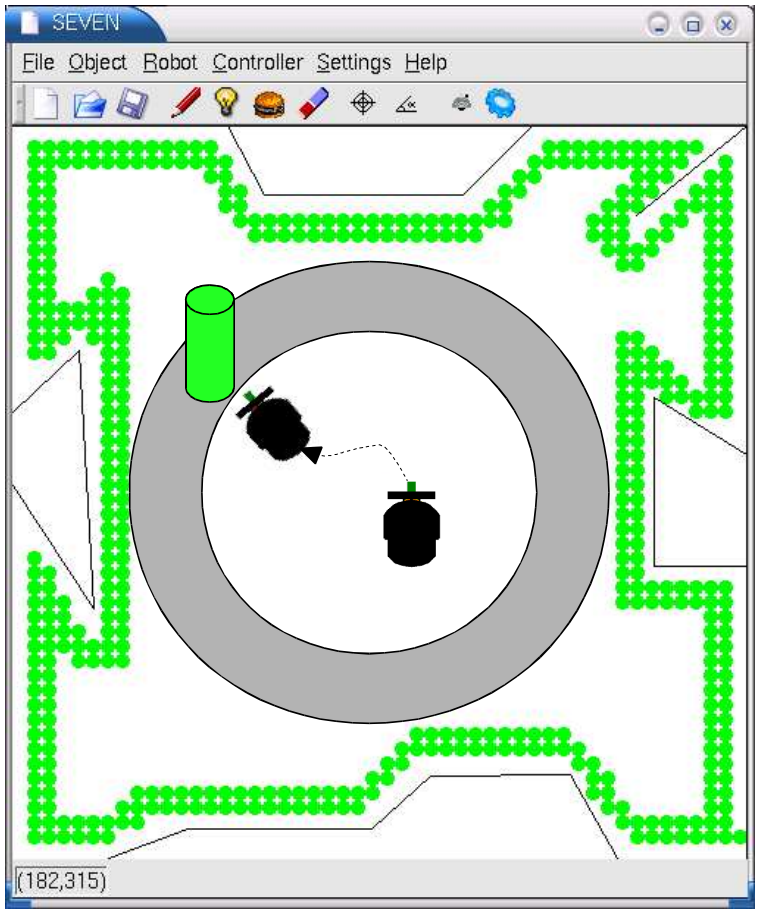


Figura 33. Esquema del montaje experimental utilizado donde el robot Pioneer 2 debe alcanzar un objeto cilíndrico que se sitúa en un sector del espacio (en gris).

La MCP en este ejemplo tiene un tamaño de 40 muestras y tanto la estrategia de reemplazo de dicha MCP como el mecanismo de gestión de la MLP, utilizan los parámetros expuestos en el anteriormente, es decir, los que consideramos más adecuados a priori.

De cara a comprobar el funcionamiento del mecanismo de gestión de la MLP en el MDB completo para el montaje experimental propuesto, debemos simular un cambio de entorno que conlleve un cambio de modelo de mundo. Esto implica que debe cambiar la función que relaciona la sensorización y la actuación. De las múltiples opciones existentes, hemos optado por modificar el valor percibido de distancia y ángulo que proporcionan los sensores de sónar de tal forma que, cada 600 iteraciones del MDB, la percepción de la distancia al objeto real tras aplicar una acción es un 60% menor que antes, y el ángulo cambia de signo. Estos cambios se pueden comparar con los que se producirían en un caso real cuando el Pioneer 2 cambia del entorno ideal mostrado en la figura 33 a otro donde los sensores fallen o simplemente cambien su rango de detección

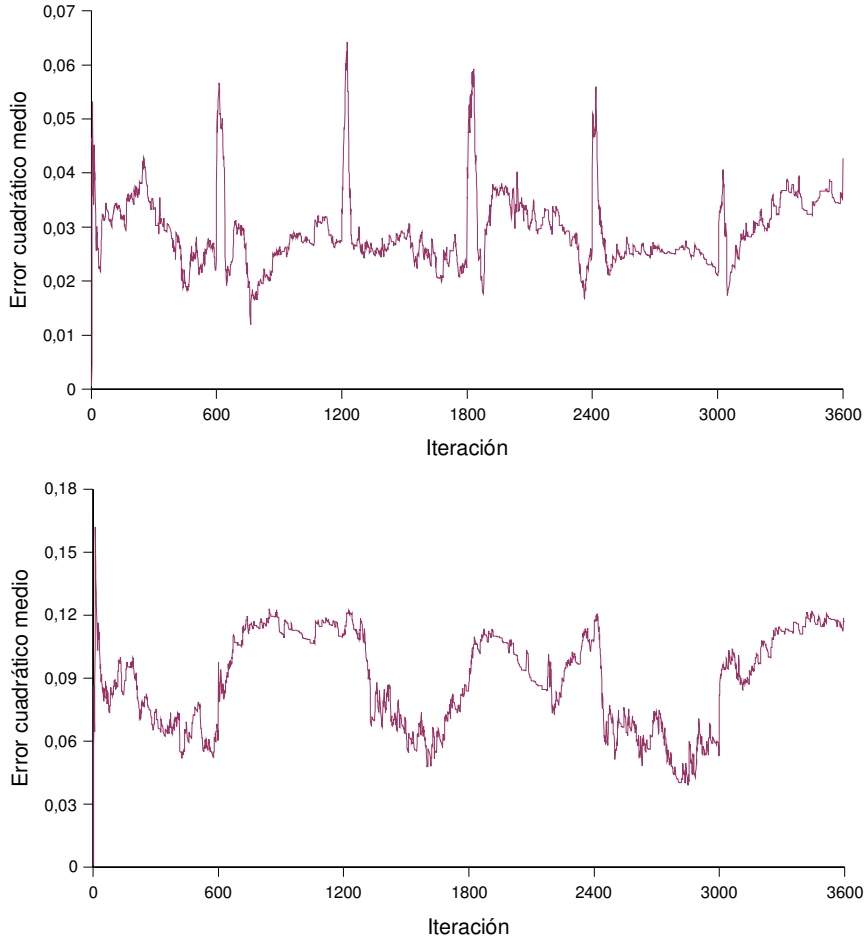


Figura 34. Error cuadrático medio para las predicciones del mejor modelo de mundo en cada iteración sobre las muestras contenidas en la MCP. En la gráfica superior se representa el error en la distancia robot-objeto y en la inferior el error en el ángulo robot-objeto

y las lecturas no se correspondan con el caso ideal. Es decir, estamos transformando el entorno simple y estático en un entorno dinámico más complejo. Es de esperar que, tras 600 iteraciones aprendiendo el modelo correcto, el fallo simulado produzca un gran error tanto en la predicción de la distancia como en la del ángulo.

En la figura 34 se observa la evolución del error cuadrático medio para las predicciones del mejor modelo de mundo en cada iteración sobre las muestras contenidas en la MCP. La gráfica superior corresponde a la distancia y la inferior al ángulo. Como vemos, en ambos casos hay oscilaciones claras en el error cada 600 iteraciones, debido a los cambios en la sensorización y podemos distinguir seis zonas diferentes. Para la predicción de la distancia, el comportamiento es muy similar al de la gráfica inferior de la figura 32 aunque con más oscilaciones en las zonas estables. Se detecta con claridad el cambio de modelo y, al igual que con la función teórica, mientras se utiliza una estrategia de reemplazo de tipo FIFO en la MCP, el error disminuye mucho. Después se reanuda la estrategia de reemplazo general y el error aumenta hasta estabilizarse. Sin el mecanismo de gestión de la MLP que hemos aplicado, tras cada cambio de entorno, el contenido de la MCP sería poco discriminatorio para el comportamiento del Pioneer 2.

La gráfica inferior de la figura 34 confirma los resultados obtenidos para la distancia, aunque la predicción en el entorno que simula un fallo (iteraciones 600 a 1200, 1800 a 2400 y 3000 a 3600) es pobre, es decir, el PBGA no encuentra una red que sea capaz de predecir el ángulo con tanta precisión como en el caso ideal. De nuevo, la utilización de la MLP permite que, al regresar a la zona inicial, el robot se adapte rápidamente a las condiciones del entorno y el proceso de reaprendizaje se reduzca.

Al observar con detalle la figura 34, vemos que las transiciones entre cambios del entorno no son tan bruscas como en la figura 32 y nos podemos preguntar si realmente los modelos que se inyectan en la población son utilizados o si el PBGA sería capaz de llegar a la misma solución sin necesidad de dicha inyección. Al detectar inestabilidades, evitamos la mezcla de muestras que se produciría en el MDB sin MLP, por lo que aunque no se inyecten los modelos almacenados, al cambiar de entorno, el PBGA tendría que aprender de nuevo a partir de una FIFO. Es de suponer que si no se utiliza MLP, el cambio en el error se haga más lentamente porque mientras la MCP contiene muestras mezcladas, el PBGA está aprendiendo a partir de información confusa. En cambio, la presencia en la población de modelos adecuados a las nuevas muestras que van llenando la MCP, debería acelerar el aprendizaje.

Estas hipótesis se confirman con los resultados que mostramos en la figura 35, donde hemos representado la evolución del error cuadrático medio para la predicción del ángulo inyectando en cada iteración los modelos que se almacenan en la MLP (línea punteada) y sin inyectar dichos modelos (línea continua). Por tanto, al no utilizar la MLP, no utilizamos el criterio de estabilidad pero sí el de inestabilidad, de modo que evitamos las mezclas en la MCP. Al igual que en el ejemplo de la figura 34, cada 600 iteraciones cambiamos el entorno, en este caso el cambio consistía por una parte en intercambiar sensores, es decir la distancia pasaba a ser el ángulo y viceversa y por otra en disminuir su valor un 80%.

En la figura se observa directamente que en las zonas de cambio (600-1200, 1800-2400, 3000-3600 y 4200-4800) el error que se obtiene utilizando la MLP es muy bajo. Esto es debido a que la primera vez que se utiliza este entorno (entre las iteraciones 600 y 1200), el PBGA logra dicho nivel de error bajo, y en los subsiguientes

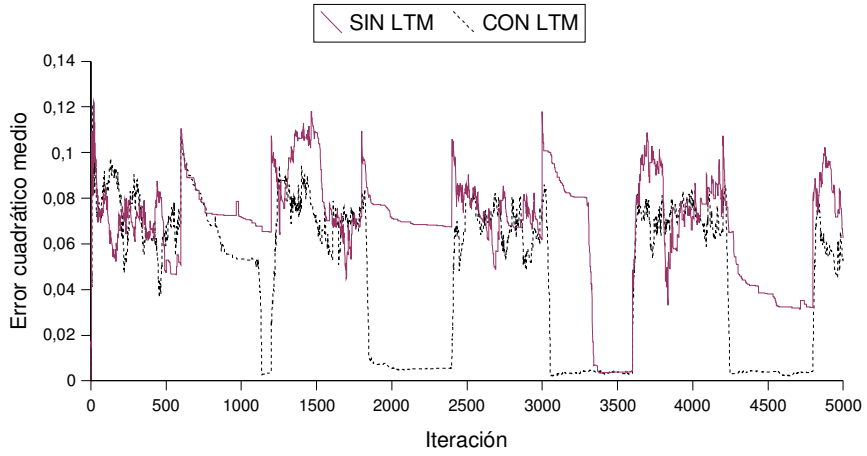


Figura 35. Error cuadrático medio para las predicciones del mejor modelo de mundo en cada iteración sobre las muestras contenidas en la MCP para el ángulo robot-objeto. La línea punteada representa la ejecución del MDB utilizando MLP y la línea continua sin utilizar MLP.

cambios a ese mismo entorno, el modelo obtenido permite alcanzar de nuevo el mismo nivel de error. En el caso de no utilizar la MLP, el aprendizaje se realiza cada vez desde la misma situación inicial en la población, y únicamente en el intervalo 3000-3600 logra un nivel de error similar al caso que utiliza MLP. Pero como vemos, en el siguiente intervalo 4200-4800, el nivel de error vuelve a ser alto. Este resultado es muy importante y justifica plenamente la inclusión de la MLP en el MDB porque, en un caso real, nos interesa que el nivel de error obtenido en el pasado en un determinado entorno, sea el que se logre siempre que en el futuro se vuelva al mismo entorno y, como vemos, sin la presencia de la MLP esto puede no ocurrir.

Además del nivel de error alcanzado, que podría ser igual en ambos casos, en la figura 35 vemos cómo el tiempo que transcurre hasta que el error se estabiliza en una zona es menor al utilizar la MLP como consecuencia de la "ayuda" que suponen los modelos que se inyectan en el proceso de aprendizaje. Por lo tanto, este tiempo es siempre menor al utilizar la MLP, característica muy relevante a la hora de trabajar en problemas reales.

Al presentar el algoritmo PBGA, vimos que posee una memoria genética que le permite almacenar información de distintas funciones mediante el uso de los genes promotores que activan o desactivan conexiones neuronales. Por este motivo, si el mecanismo de gestión de la MLP no encuentra ningún modelo estable en un entorno, el PBGA posee la capacidad de retener información sobre dicho entorno y de adaptarse con rapidez en caso de que se presente de nuevo. Obviamente, esta información se mantiene si el tiempo transcurrido entre el cambio de entorno no es muy grande. Una situación similar ocurre en la figura 35, donde el error en los entornos originales (iteraciones 0-600, 1200-1800, 2400-3000 y 3600-4200) no se estabiliza y no se almacena ningún modelo en la MLP para estas zonas. Por este motivo, ambas evoluciones son análogas y es la información genética que posee el PBGA la que logra transiciones rápidas.

Volveremos a incidir sobre el funcionamiento global del mecanismo en un entorno dinámico en el ejemplo de aplicación del apartado 7.2, pero el llevado a cabo aquí nos da una idea clara de las ventajas que representa la inclusión de la MLP en el MDB.

Conclusiones al estudio de la Memoria a Largo Plazo

El diagrama de bloques de la figura 29 representa el esquema final del MDB que hemos desarrollado. La principal mejora respecto a las primeras versiones, reside en la capacidad de almacenar modelos que han resultado satisfactorios, eliminando así la necesidad de reaprenderlos. Aunque en muchos problemas reales donde se aplique el MDB no se haga necesaria la utilización de esta memoria porque el entorno no sea tan dinámico como se plantea en el estudio teórico, no cabe duda de que es una característica básica para todo modelo cognitivo. Como ya comentamos al inicio de este apartado, la memoria es una pieza fundamental del engranaje inteligente de cualquier ser o agente artificial.

Hasta este instante, la única forma de aplicar el MDB a entornos dinámicos, era utilizando una estrategia de reemplazo de tipo FIFO para la MCP. De otra forma, cada cambio en la función a modelar, conllevaba mezclas de muestras en dicha MCP, con lo que los modelos obtenidos resultaban erróneos. De acuerdo con los resultados prácticos obtenidos, la conclusión más importante de este estudio es que ahora el MDB sí está realmente preparado para trabajar con cualquier tipo de entorno manteniendo una estrategia de reemplazo en la MCP que maximice la generalidad y relevancia de las muestras. Pero en este sentido debemos realizar una puntualización muy importante y es que, al no utilizar el tiempo en la estrategia de reemplazo de la MCP, las muestras no "caducan". Esto implica que, si el criterio de inestabilidad no detecta un cambio de entorno, las muestras se mezclarán con el consecuente error en los modelos que se obtienen, que intentarán aproximar ambos entornos.

Los cambios de entorno que hemos utilizado hasta ahora han sido bruscos, pero en un entorno real donde se produzca un cambio sutil podemos tener problemas y el MDB debe reaccionar lo antes posible. La solución consiste en activar el término temporal en la estrategia de reemplazo, de modo que las muestras más antiguas sean cada vez más susceptibles de abandonar la MCP. Como consecuencia, perdemos generalidad en los modelos que se van obteniendo, ya que tienden a aproximar zonas más pequeñas del entorno, pero el funcionamiento adecuado del MDB no se ve afectado.

El mecanismo de gestión de la MLP que hemos tenido que desarrollar, consta de dos partes diferenciadas, que son la selección de un modelo bien aprendido y relevante para el agente como candidato a ser almacenado, y la estrategia de reemplazo que debe mantener en la MLP el número mínimo de modelos diferentes que son aprendidos. Además, la incorporación de este mecanismo de gestión al MDB implica que se deben detectar con claridad los cambios en el entorno para evitar el almacenamiento en la MLP de modelos de zonas de transición. El funcionamiento del criterio de inestabilidad implica cambios profundos en el MDB, ya que condiciona la información que se almacena en la MCP, modificando para ello la antigüedad de las muestras.

El concepto de MLP es más general que el expuesto en este apartado y se podría extender para almacenar no sólo modelos, sino partes de modelos u otras estructuras básicas que combinadas en instantes futuros favorezcan la interacción del robot en su entorno.

Con esto damos por finalizado esta parte dedicada al estudio en profundidad de los

procesos de búsqueda de modelos y al análisis de las memorias que entrar en juego en el mecanismo. Hemos presentado un nuevo algoritmo evolutivo desarrollado especialmente para trabajar en entornos reales, el PBGA, hemos desarrollado una estrategia de reemplazo para la Memoria a Corto Plazo y, por último, hemos desarrollado una técnica para reutilizar la información aprendida en los procesos evolutivos de forma automática basada en la inclusión de una Memoria a Largo Plazo en el mecanismo.

En el apartado siguiente presentamos un generalización a la operación interna del MDB, donde se plantea una metodología que permita la obtención y combinación automática de los modelos aprendidos, generalizando así los procesos de búsqueda y aprendizaje que tienen lugar en el mecanismo.

6.3 Características avanzadas

A continuación presentamos una mejora sobre el algoritmo PBGA que nos permitirá una reutilización más eficiente de todos los modelos aprendidos y que tiene su fundamento en la cooperación entre individuos existente en los procesos evolutivos naturales.

Generación de grupos por afinidad. PBGA generalizado

Una de las principales características del algoritmo PBGA, es la capacidad de obtener de forma automática la descomposición por variables de salida de una función compleja. Esto nos ha permitido trabajar con modelos de mundo más reales en el MDB manteniendo tamaños de redes moderados con el consecuente ahorro en coste computacional.

Desde un punto de vista puramente evolutivo, se han aportado los medios para que en las poblaciones del algoritmo evolutivo se formen especies por salida. La definición de especie que más se adapta al MDB es la aportada por el naturalista G. G. Simpson (1961):

> *"Una especie evolutiva es una estirpe (una secuencia de poblaciones ancestro-descendiente) que evoluciona separadamente de otras y que tiene un papel y unas tendencias de evolución propios y de carácter unitario".*

Desde un punto de vista más biológico que evolutivo, la definición de especie está basada en la observación de que las poblaciones de diferentes especies coexisten, comparten el territorio pero no se cruzan, es decir, no existe reproducción entre individuos de distintas especies.

De estas definiciones se deduce que, si de una especie se derivan otras, es necesario que exista algún tipo de barrera para que evolucionen de forma independiente y no haya cruce. Estas barreras pueden estar dadas por la separación geográfica o porque existen las condiciones necesarias para que haya una separación reproductiva dentro del mismo territorio. Bien, en el caso del PBGA, la barreras son del segundo tipo y las condiciones para la separación reproductiva han sido impuestas por el diseño. La característica que utilizamos para evitar que individuos de distintas especies se crucen es la salida a la que afectan, es decir, la función que realizan.

Dado que conocemos que la única forma que tienen los individuos de manifestar su comportamiento es a través de la salida o salidas que tienen activas, utilizamos dicha propiedad para crear las especies. En el PBGA, por tanto, el número máximo de especies que se pueden crear coincide con el número de variables de salida de la función a aproximar. Es el número máximo porque dos especies pueden ser idénticas si los individuos que poseen esas dos salidas activas son mejores que los de una salida por separado. La propia evolución es la que, por medio de la función de calidad, establece la distribución de individuos en cada especie.

En este apartado pretendemos generalizar la formación de especies en la población sin imponer de forma explícita la característica que diferencia dichas especies, es decir, la barrera evolutiva. Es la realización de una tarea común, el logro de un mismo objetivo, el que da lugar a la formación de especies ya que en la población se forman *grupos* de individuos diferentes que se asocian para realizar dicha tarea. Esta asociación se realiza en función de algún tipo de criterio de *afinidad*, de modo que los individuos

más afines tienden a colaborar del mismo modo que ocurre en los procesos evolutivos naturales. Desde un punto de vista puramente funcional, la afinidad de un individuo respecto a otro o a otros depende únicamente del grado de satisfacción obtenido tras la colaboración en la tarea a realizar. Pero el concepto de afinidad es mucho más complejo que lo expuesto aquí, y a la hora de formar grupos de individuos que colaboren se puede tener en cuenta cualquier tipo de criterio subjetivo.

Con este planteamiento a base de grupos, las especies aparecen porque dentro de dichos grupos los individuos realizan diferentes funciones con lo que son diferentes unos de otros y, a la hora de cruzarse, tienden a hacerlo los que son más parecidos entre sí. Con el paso del tiempo, estos procesos de cruce conducen a la aparición de especies de individuos similares.

Como vemos, la formación de especies en el PBGA es un caso particular de la formación de grupos por afinidad, donde el grado de afinidad viene dado por la salida que está activa. Así, redes con distintas salidas activas son más afines que redes con las mismas salidas activas y, entre redes afines, lo serán más las que proporcionen un menor error en la predicción de la función global.

Por tanto, hemos desarrollado un proceso de “interrelación en vida" de los individuos que forman la población de forma que la supervivencia viene determinada por la calidad que obtienen los individuos al colaborar en un *grupo* y no por la calidad del individuo por separado. La parte más importante de este proceso de formación de grupos es el establecimiento de la *afinidad* entre individuos, que debe formar parte del proceso evolutivo e irse adaptando a las necesidades y virtudes de cada individuo de forma autónoma. Así, al igual que en un proceso evolutivo real, los grupos de individuos se forman porque se dan las condiciones de forma natural para satisfacer algún tipo de necesidad común. Como veremos más adelante, conceptualmente, dentro de un grupo pueden existir varios individuos de la misma especie.

Desde el punto de vista del algoritmo evolutivo, el objetivo de la creación de grupos es poder obtener individuos como composición de subindividuos por medio de la combinación de sus actuaciones individuales, permitiendo una reutilización mucho más versátil de los conocimientos adquiridos. De esta forma, no imponemos que las especies se tengan que formar necesariamente por salida ya que, dentro de cada salida, también se pueden formar especies. Por ejemplo, podría ocurrir que una de las salidas de un modelo de mundo sea composición de dos funciones simples, de modo que sería posible la existencia en la población de individuos especializados en cada una de estas dos funciones.

En el campo de la optimización multiobjetivo, es usual la búsqueda de una colaboración entre individuos de la población de los algoritmos evolutivos [Coello, 02]. Para ello se han desarrollado técnicas basadas en la compartición de la calidad (fitness sharing) entre todos los individuos, de modo que la calidad de un cierto individuo se escala mediante algún tipo de medida de similitud respecto a los demás [Goldberg, 87]. Pero, tal y como explicamos al hablar del algoritmo PBGA, este tipo de técnicas parten del conocimiento del frente de Pareto a obtener, es decir, parten del conocimiento de los distintos objetivos y buscan individuos especializados en estos objetivos que colaboren entre sí. Nuestro problema es muy diferente, ya que buscamos una descomposición automática en modelos lo más simples y básicos posible, pero sin conocimiento alguno de cómo llevarlo a cabo.

Un trabajo que debemos destacar es el llevado a cabo por Xin Yao y Paul Darwen en [Darwen, 97], donde proponen una aproximación al aprendizaje evolutivo para diseñar sistemas modulares de forma automática. Para ello, utilizan el concepto de especiación introducido por Goldberg [Goldberg, 89] mediante una técnica basada en compartición de calidad. La novedad de este trabajo reside en que, dentro de cada especie, utilizan técnicas de coevolución donde no existe el concepto de una única solución, en este caso, de un único individuo mejor que el resto. Los autores aplican esta técnica a un juego (el dilema del prisionero) donde este algoritmo se muestra muy eficiente.

Este trabajo es el más próximo que hemos encontrado a la filosofía que subyace tras la formación de grupos por afinidad que estamos presentando. Yao y Darwen no utilizan el concepto de afinidad, sino que heredan la compartición de calidad de los problemas multiobjetivo. En cualquier caso, es obvio que las investigaciones que tratan con entornos y problemas reales tienden a dedicar grandes esfuerzos a la modularización automática de las soluciones, buscando modelos lo más simples y reutilizables que sea posible.

En resumen, de cara a la formación de especies no condicionadas de individuos en la población del algoritmo evolutivo, planteamos una versión generalizada del PBGA: el GPBGA. Está basada en la creación de grupos de individuos que colaboran para llevar a cabo el objetivo final del algoritmo, utilizando para ello un parámetro de afinidad entre individuos que establece las preferencias a la hora de formar los grupos. La afinidad es una propiedad emergente de la interacción, no está predefinida. Esta generalización favorece el aprendizaje gradual de comportamientos en el MDB y, partiendo de casos simples, podemos ir complicando los ejemplos constructivamente para que se puedan llegar a generar modelos muy complejos. El algoritmo PBGA se verá así generalizado al permitir que las especies se formen de modo automático sin imponer el diseñador la condición que las diferencia. A continuación detallaremos las principales diferencias que existen entre este nuevo PBGA generalizado y el original.

Características de funcionamiento del PBGA generalizado

La principal característica del PBGA generalizado reside en que los individuos pueden formar parte de un grupo, en cuyo caso serán evaluados como miembros del grupo, y no individualmente. De hecho, la solución final que aportará el GPBGA tras la evolución será un grupo que ha de proporcionar todas las variables de salida del problema. Dicho grupo podrá estar formado por un único individuo o por varios que pueden colaborar de distintas formas. Por ejemplo, si el GPBGA debe obtener un modelo de mundo con 2 salidas, el grupo final podría estar formado por dos redes de una salida cada una, por una única red de dos salidas o incluso por más de dos redes. Este último caso es posible porque las redes pueden combinar sus salidas, de tal forma que cada una de ellas se especializa en partes de la función asignada a cada salida. La idea general es que en un grupo se juntan individuos que colaboran de todas las formas posibles para llegar a una solución mejor que la que obtendrían por separado. El GPBGA debe permitir ambos extremos y que sea la propia selección natural la que decida la necesidad de la formación de dichos grupos. Con este planteamiento, se favorece la aparición de especies de individuos en la población que realizan distintas funciones en los grupos.

Antes de analizar con más detalle la formación de grupos por afinidad, mostramos de forma esquemática, el funcionamiento del PBGA generalizado en comparación con el PBGA:

1. Se crea una población aleatoria inicial de redes con genes promotores, al igual que en el PBGA.
2. Calculamos la calidad de dichas redes. Para ello, se llevan a cabo los siguientes pasos:
   a) Formamos grupos de individuos utilizando un valor de afinidad que refleja las preferencias de cada individuo.
   b) Cada grupo formado tiene una calidad de grupo, que resulta de la aplicación conjunta de los individuos del grupo al problema a tratar.
   c) Realizamos un proceso de recombinación de grupos, donde se prueban nuevos a partir de los anteriores.
   d) Se calcula de nuevo la calidad de los grupos formados manteniendo siempre los mejores.
   e) Se repiten los pasos c) y d) un número dado de veces. Tras finalizar, la calidad de cada individuo es su calidad de grupo.
3. Aplicamos el operador de selección sobre la población utilizando la afinidad y la calidad de cada individuo como criterios.
4. Aplicamos los operadores de cruce y mutación al igual que en el PBGA.
5. Calculamos la calidad como se explica en el punto 2.
6. Repetimos los pasos 3, 4 y 5 un número de generaciones dado al igual que en el PBGA.

Los principales cambios residen en el cálculo de la calidad y en el operador de selección, por lo que los trataremos con detalle en los próximos apartados.

Calidad de grupo

Dado que el algoritmo evolutivo trabaja ahora con grupos en lugar de individuos, el cálculo de la calidad se verá afectado ya que un individuo no tiene relevancia de forma independiente, sino como parte del grupo. Por este motivo, la calidad de cada individuo es la calidad del grupo en el que se encuentra.

El primer paso a la hora de calcular la calidad de grupo es, obviamente, la formación de los grupos. Para ello se pueden utilizar distintas alternativas, pero nosotros hemos optado por la siguiente: cada individuo posee un vector de afinidad que almacena una serie de valores numéricos que representan sus características, de modo que el acompañante ideal de cada individuo es otro con su vector de afinidad complementario. Por tanto, el vector de afinidad representa tanto las propias características del individuo como las que desearía poseer, es decir, las características que busca en otro u otros para, en conjunto, convertirse en un individuo ideal. Hemos optado porque cada individuo pueda formar su propio grupo escogiendo de entre una ventana de candidatos (como en todo algoritmo evolutivo, la selección no se realiza sobre toda la población sino sobre una ventana de tamaño menor con objeto de permitir la ampliación del tamaño de población con un coste computacional lineal y no exponencial), aquel o aquellos que sean más afines. No existe un límite conceptual en el tamaño de un grupo, simplemente cada individuo que forma grupo busca un vector de afinidad complementario al suyo, de tal forma que si no lo consigue uniéndose a un único individuo, buscará otro y así sucesivamente. En el apartado siguiente estudiaremos el vector de afinidades, y veremos

cómo se establece el grado de afinidad entre vectores y como se puede implementar la complementariedad.

Con este esquema de formación de grupos, tras esta primera etapa, tendremos tantos como individuos pero teniendo en cuenta que cada individuo puede aparecer en más de un grupo. De forma simplificada, podemos decir que estamos utilizando individuos, pero más complejos que en un algoritmo evolutivo tradicional.

La calidad de un grupo se calcula combinando mediante algún tipo de operación aritmética, normalmente una suma o un producto, las calidades individuales de los miembros del grupo calculadas sobre todas las muestras almacenadas en la MCP. Entonces, los miembros de un grupo combinan sus calidades individuales teniendo en cuenta que el grupo debe proporcionar todas las variables de salida del problema. Por ejemplo, en la figura 36 mostramos un esquema para el caso en el que tenemos un grupo con cuatro individuos en un problema con dos variables de salida. La calidad de cada individuo se combina con la del resto para cada una de las salidas de forma independiente al igual que en el PBGA, obteniendo una calidad de grupo por salida. Pero ahora el grupo es la unidad funcional única del algoritmo, por lo que la calidad del grupo se debe calcular como la combinación de las calidades por salida. Desde un punto de vista conceptual, pero que escapa al propósito del presente trabajo, los grupos podrían contener individuos que no afecten directamente a las salidas, sino que regulen las combinaciones entre el resto de miembros del grupo.

Una vez que cada grupo ha sido formado y tiene una calidad asociada, llevamos a cabo una etapa de recombinación de grupos que permite que cada individuo cree de nuevo su grupo más afín. En cada paso de esta nueva etapa, los individuos forman grupos y se calcula la calidad de dichos grupos. Si ésta es mayor que la del grupo anterior, el nuevo es el que se mantiene y si es menor, el grupo que ya existía continúa.

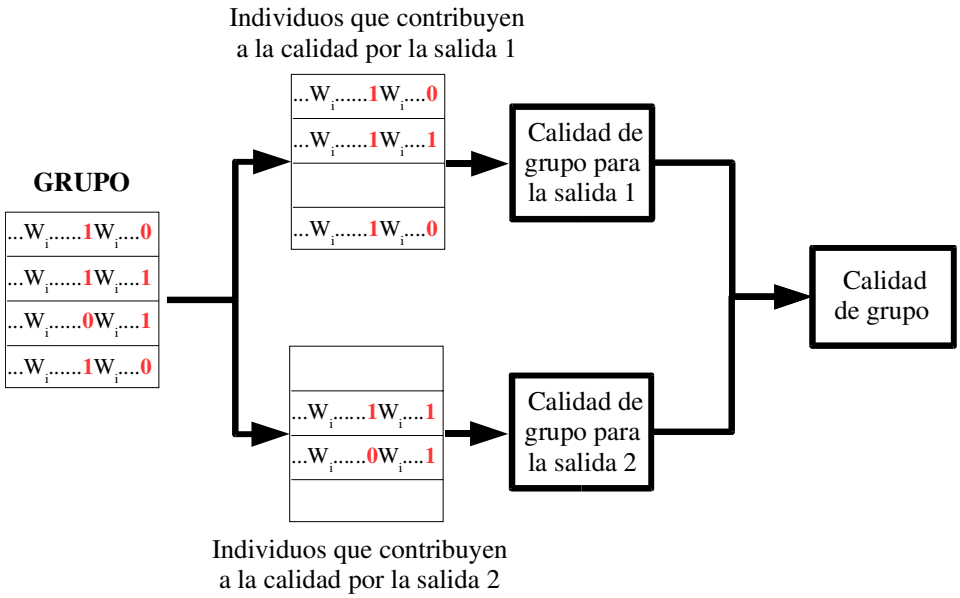


Figura 36. Esquema que representa el proceso de cálculo de la calidad de grupo para el caso un grupo de 4 individuos y un problema con dos variables de salida

Tras un número fijado de recombinaciones, cada individuo ha formado el grupo más afín y de mejor calidad posible, por lo que dicha calidad de grupo se establece como calidad individual. Así, desde el punto de vista del PBGA, la formación y recombinación de grupos es una fase del algoritmo donde se calcula la calidad de cada individuo de forma más compleja.

La parte fundamental de este proceso es la creación y modificación del vector de afinidad, que pasamos a estudiar a continuación.

Vector de afinidad

El vector de afinidad representa las preferencias de un individuo a la hora escoger miembros para su grupo. Es un concepto que nos permite razonamientos de más alto nivel desde el punto de vista inteligente, ya que nos otorga la posibilidad de incluir distintos criterios para establecer las preferencias. Podríamos utilizar un criterio puramente funcional, de modo que la afinidad dependa únicamente del resultado final del grupo o bien podríamos incluir criterios de tipo subjetivo (que reflejen los "gustos" de los individuos). En este sentido, un individuo podría escoger a otro para formar grupo porque le atrae como "amigo", y no sólo como "compañero de trabajo".

Cada elemento del vector de afinidad se representa mediante un número real en el caso más general. Cuando un individuo forma un grupo, busca otro u otros individuos con vectores de afinidad complementarios. Por ejemplo, si cada elemento se representa mediante un número real entre $-V$ y V, podemos definir la complementariedad como la obtención de un valor cero en dicho elemento tras la suma de los valores correspondientes. De esta forma, un valor $-V_1$ es el complemento de un valor V_1 y un valor V_2 del $-V_2$. Este criterio no implica que los valores de afinidad sean opuestos en signo, ya que depende de si el rango es positivo y negativo como en este ejemplo o del mismo signo. Además, cuando un individuo forma grupo, busca otro individuo lo más complementario posible a su afinidad y, en caso de necesitarlo, buscará otro a continuación que sea el más complementario respecto a la afinidad de los dos que ya están en el grupo. Este proceso de selección por afinidad se repite mientras el individuo que forma grupo no consiga un vector de afinidad complementario al suyo. Como se observa a partir de este razonamiento, existen distintas formas de definir complementariedad entre vectores dependiendo de la nomenclatura que se utilice.

En cuanto a la dimensión del vector de afinidad, ésta depende únicamente de la complejidad que queramos en los grupos. Así, cuanto mayor sea la dimensión, más difícil será que un individuo encuentre su vector complementario uniéndose únicamente a otro individuo. En general, cuanto mayor sea la dimensión del vector de afinidad, mayor será la complejidad del grupo formado. Si utilizamos la definición de complementariedad entre vectores del ejemplo anterior, un individuo con vector de afinidad de la forma (V_1, V_2, $-V_3$) podría escoger como único individuo para formar grupo el que tuviese un vector de afinidad lo más cercano a ($-V_1$, $-V_2$, V_3), ya que el resultado de sumar ambos es el vector nulo. Pero resultaría numéricamente más simple el encontrar dos individuos de la forma ($-V_4$, $-V_5$, V_6) y ($-V_7$, $-V_8$, V_9) ya que se verifica que $-V_4 - V_7 = V_1$, $-V_5 - V_8 = V_2$ y $V_6 + V_9 = V_3$. Lógicamente existen más posibilidades de que se de el segundo caso que el primero. En este ejemplo, también se podrían formar grupos de tres individuos.

Una vez definidos los vectores en cuanto a su dimensión y rango de los valores de cada elemento, debemos establecer cómo se modifican dichos valores adaptándose así a

las preferencias de cada individuo. Inicialmente, los valores de afinidad son aleatorios y cuando se forman grupos por primera vez, utilizan estas afinidades aleatorias. Por tanto, los primeros grupos son también aleatorios. Tras el cálculo de la calidad de grupo, se inicia el proceso de recombinación y es aquí donde reside la etapa más relevante para los vectores de afinidad. Cada vez que se prueba un nuevo grupo, se obtiene un nuevo valor de calidad que podemos comparar con la calidad del grupo anterior. Esto nos da información sobre lo afines que son los individuos de ambos grupos, tanto si la calidad mejora como si empeora y el nuevo grupo no se forma.

Si la calidad del nuevo grupo es mejor, los vectores de afinidad de los individuos del nuevo grupo se separan entre sí en el espacio n-dimensional que definen dichos vectores. Esta separación es proporcional a la diferencia de calidad entre los grupos, de modo que cuanto mayor sea la mejora más se separan los vectores. Si calidad del nuevo grupo es peor, los que se separan son los vectores de afinidad de los individuos del grupo ya existente. Esta metodología favorece la formación de especies, es decir, de "clusters" de individuos con vectores de afinidad similares, ya que la separación está limitada por los rangos que utilizamos en los vectores. Por ejemplo, en el caso de utilizar un vector de dimensión dos con los elementos limitados al rango [-1, 1], la tendencia del proceso lleva a la formación de 4 especies de valores [-1, 0], [-1, 1], [1, 0] y [1, 1] por ser las más separadas espacialmente. Obviamente, se formarán estas cuatro especies en caso de ser necesarios cuatro individuos en los grupos.

Vemos que, tras cada etapa de recombinación, todos los individuos han ajustado su vector de afinidad automáticamente dependiendo de la función que realizan en el grupo. Además, los individuos que pertenecen a más de un grupo ajustan su vector en cada prueba, no sólo en su grupo propio maximizando así la información disponible.

Con el paso del tiempo, cada individuo tendrá un vector de afinidad que representará sus preferencias y, por tanto, sus deficiencias y que ha sido obtenido de manera totalmente autónoma. Es decir, hemos desarrollado un mecanismo autónomo para la formación de grupos de individuos que colaboran en una tarea común, y que conlleva la formación de especies en la población. La característica que distingue ahora a las especies no está impuesta por el diseñador, sino que se ajusta en "tiempo de vida" por la propia evolución y es el vector de afinidad. Cuanto mayor sea la dimensión del vector de afinidad mayor es el número posible de distintos vectores y, en consecuencia, mayor es el número de especies que se pueden formar en la población.

En resumen, tras la formación y recombinación de los grupos tenemos una calidad para cada individuo que lo cataloga en cuanto a su funcionalidad en la población y un vector de afinidades que lo cataloga en cuanto a sus características. Pero este proceso no cambia los individuos, las redes, que forman parte de la población del PBGA y que son el sustrato de las soluciones. Para ello debemos continuar con la evolución de la misma forma que en cualquier algoritmo evolutivo tras el cálculo de la calidad. El siguiente paso es el proceso de selección, que pasamos a estudiar a continuación.

Operador de selección

En el PBGA desarrollamos un operador de selección que utilizaba una subpoblación por salida, de tal forma que las redes con distintas salidas activas no se pudiesen cruzar por pertenecer a especies radicalmente diferentes. Como hemos venido diciendo desde el comienzo del estudio del PBGA generalizado, ahora las especies las crea el propio proceso evolutivo mediante los vectores de afinidad y el diseñador no impone que los

individuos se agrupen por salidas activas. Por tanto, el planteamiento del operador de selección es el mismo que en el caso del PBGA pero generalizado y se puede esquematizar en 3 pasos:

1. Un individuo de la población es seleccionado para cruzarse según una probabilidad de cruce en un proceso de torneo clásico.
2. De entre una ventana de torneo escoge al individuo con el que se va a reproducir en función de dos criterios:
   a) El que posea un vector de afinidad más similar.
   b) Entre los más afines, el que posea mayor calidad (que es la calidad de su grupo).
3. Los dos individuos seleccionados se cruzan dando lugar a un único descendiente.
4. Repetimos estos tres pasos tantas veces como individuos tenga la población.

En caso de que la probabilidad del operador indique que un individuo no debe ser cruzado, éste pasa directamente a la siguiente generación. Este planteamiento engloba al utilizado en el PBGA. Ahora no es posible el crear subpoblaciones porque el número de vectores de afinidad distintos no es una cantidad discreta. En el caso del PBGA, sabemos que debe existir una subpoblación por salida y así optimizamos el proceso de reproducción asegurando que nunca se van a juntar individuos de especies diferentes. Al utilizar el vector de afinidad, ya no es posible asegurar esto y lo único que podemos hacer es favorecer que los individuos con vectores más parecidos se reproduzcan porque los individuos con vectores de afinidad similares están realizando funciones similares en los grupos en los que participan, es decir, son de la misma especie. Y si queremos que el algoritmo converja, hemos de favorecer la reproducción de individuos de la misma especie.

Con el paso de las generaciones, los vectores de afinidad tenderán a distribuirse dando lugar al número mínimo de ellos que son útiles para la evolución, es decir, al número mínimo de individuos diferentes (especies). Por este motivo, la selección de individuos con vectores de afinidad similares es más efectiva con el paso de las generaciones. Inicialmente, la evolución en el GPBGA será más lenta que en el PBGA porque los individuos que se reproducen no serán realmente afines hasta que los vectores de afinidad se hayan ajustado. Una vez que dichos vectores estén distribuidos de forma estable, es la calidad la que guía la evolución como en cualquier algoritmo evolutivo.

El operador de cruce que se les aplica a los individuos tras la selección es idéntico al expuesto en el apartado dedicado al PBGA. Asimismo, el operador de mutación se aplica sobre la población de individuos tras el cruce como en el caso del PBGA. Estos dos operadores son independientes de la utilización de grupos o de la creación de especies, ya que se aplican a nivel de individuo (red).

La única decisión que hemos tenido que adoptar tras la aplicación del operador de cruce, es cómo se ve afectado el vector de afinidad. Si un individuo pasa de una generación a la siguiente sin cruzar, lógicamente su vector debe ser el mismo pero, cuando hay cruce, no se puede establecer el grado de variación del vector en función de lo que aporta cada individuo al hijo. La solución que hemos adoptado consiste en asignar un vector de afinidad al descendiente que sea la media aritmética de los vectores de los padres en cada elemento. Esto no conlleva prácticamente ningún cambio si los vectores de afinidad de los padres son similares y, en caso de que sean muy diferentes

(algo que no estamos favoreciendo), el descendiente partirá de un caso intermedio. Dado que los vectores de afinidad representan la especie a la que pertenece o se parece más un individuo, en caso de que dichas especies sean diferentes, no podemos hacer ninguna suposición acerca de la especie del descendiente, por lo que le asignamos un valor intermedio que sí resulta adecuado en el caso de que sean de la misma especie.

En los siguientes apartados, mostraremos los resultados obtenidos con el GPBGA centrándonos en el estudio de los grupos que se forman y, sobre todo, del vector de afinidad.

Grupos de dos individuos

En esta primera aproximación a la formación de especies a través de grupos por afinidad, nos planteamos la obtención de un modelo de mundo teórico utilizando el PBGA generalizado, de forma similar a lo que hicimos en los apartados 6.2.2 y 6.2.3 al estudiar las estrategias de reemplazo para la MCP y la MLP. Es decir, no utilizamos el MDB completo sino simplemente la parte relacionada con la obtención de los modelos de mundo. La función a obtener se muestra en la figura 37 y es el resultado de sumar la ya conocida campana de Gauss y una función senoidal 3D que se utilizaron en el ejemplo del apartado 6.2.3 dedicado a la MLP. Es una función con 2 variables de entrada (x e y) y una única variable de salida (z). De nuevo, los parámetros concretos de cada evolución pueden ser consultados en el Apéndice B.

El objetivo es simplemente obtener la red neuronal que mejor aproxime la función de la figura 37. Si utilizásemos el PBGA en este problema, al tener una única salida, nunca se formarían especies de individuos independientes y la solución obtenida sería un

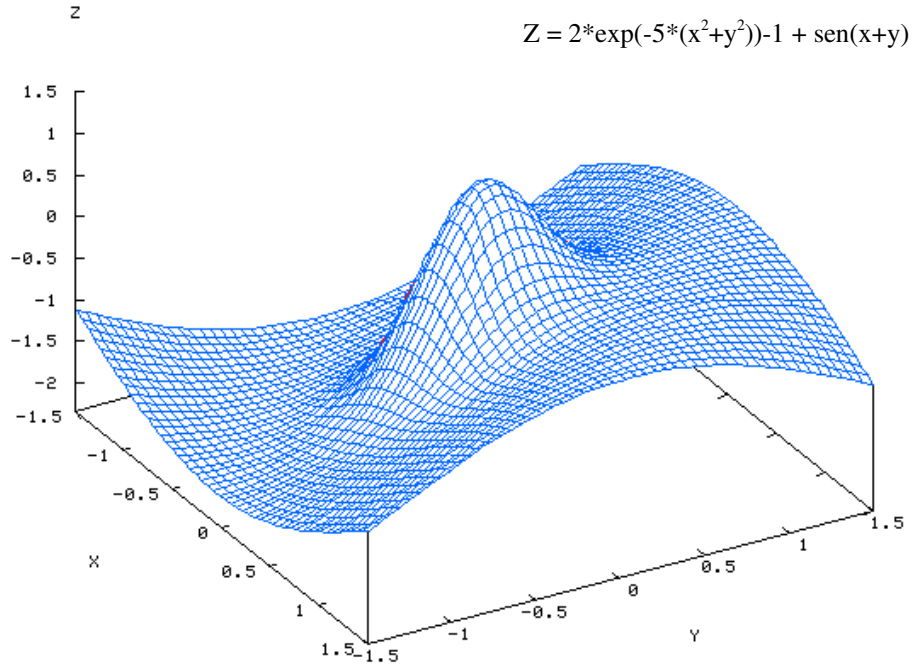


*Figura 37. Función que se debe aproximar utilizando el PBGA generalizado. Es el resultado de sumar las funciones Gaussiana: z = 2*exp(-5*(x²+y²)) - 1 y senoidal 3D: z = sen (x+y)*

individuo simple. Pero al utilizar el GPBGA, la solución vendrá dada por un grupo que, en función de como se defina el vector de afinidad, será más o menos complejo

En este ejemplo, hemos fijado el tamaño máximo de los grupos a dos, y el vector de afinidad es de dimensión uno, es decir, es un valor numérico. Lo hemos hecho así para comprobar si el GPBGA es capaz de obtener dos especies que aproximen las dos funciones básicas que componen la función de la figura 37, esto es, una función gaussiana y una senoidal u otras similares. Con todo este desarrollo del tema de grupos, pretendemos establecer una metodología para extraer subfunciones de una función compleja de forma automática. Las funciones simples que hemos utilizado para crear la función compleja son conocidas, por lo que podremos analizar qué especies se forman y si realmente son distintas y relevantes.

Cuando se calcula la calidad de cada individuo, formamos y recombinamos grupos tal y como se ha explicado en los apartados anteriores. Para ello utilizamos un único valor de afinidad en el rango [-1, 1], de tal forma que el individuo que debe formar grupo debe escoger otro que lo acompañe y que posea una afinidad complementaria. En este caso, buscamos que la suma de ambos valores de afinidad sea lo más próxima a cero posible. Por ejemplo, para un vector de afinidad de dimensión dos [-1, 0.5], el individuo ideal sería [1, -0.5]. El cálculo de la calidad de grupo se realiza aplicando las dos redes de cada grupo a cada una de las muestras de la MCP y sumando sus salidas individuales. De esta forma sabemos el comportamiento del grupo en conjunto, como una única entidad sobre toda la MCP.

La parte más relevante de esta implementación es la modificación de la afinidad. Como establecimos con anterioridad, cada vez que se calcula la calidad de grupo tenemos nuevos datos para calcular la afinidad de los individuos. Siempre que en una de estas pruebas un individuo aumente su calidad de grupo, separamos su valor de afinidad del valor del otro miembro del grupo de modo proporcional a la mejora en dicha calidad. Teniendo en cuenta los rangos utilizados para la afinidad en este caso, hemos limitado la separación a un valor máximo de 0.2.

Los resultados obtenidos se muestran en la figura 38, donde hemos representado la predicción (superficie) de un muestreo de la función objetivo (puntos) que realiza el mejor los grupos obtenidos tras 5000 iteraciones de evolución. Como se observa, el seguimiento es satisfactorio aunque lo realmente interesante de este ejemplo ha sido la distribución final de los valores de afinidad de todos los individuos de la población y que podemos ver en la figura 39. Llama la atención la claridad con la que se forman dos agrupaciones de individuos prácticamente iguales, unos con afinidad 1 y otros con afinidad -1. Al utilizar grupos de tamaño dos, estamos favoreciendo este tipo de separación entre individuos ya que siempre que un grupo funciona bien, separamos las afinidades hasta un límite superior de 1 y otro inferior de -1. Si los grupos fuesen de tres individuos como máximo, la máxima separación entre afinidades se produciría en los valores -1, 0, 1 que podría dar lugar a tres especies como máximo. Pero la formación de una especie de afinidad 0 sería más lenta que las otras dos porque el 0 no es un valor límite donde las asignaciones de afinidad se tiendan a agrupar.

En el caso que nos ocupa, se forman dos especies, una de ellas se caracteriza por un valor de afinidad 1 y la otra por un valor -1, como muestra la figura 39. El haber limitado el problema a dos individuos por grupo permite la aparición de, como máximo, dos especies. Al trabajar con un único valor de afinidad, no existen problemas para que los individuos que forman grupo encuentren otro que sea complementario. Con un

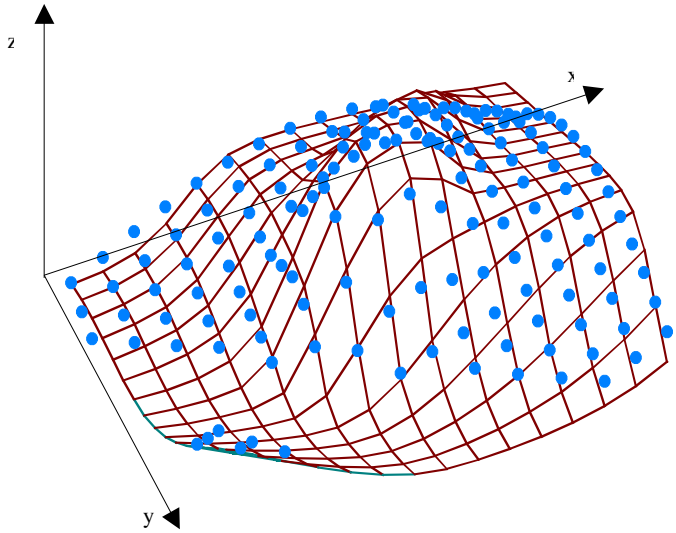


Figura 38. La superficie representa la predicción de un muestreo de la función objetivo (puntos) que realiza el mejor grupo obtenido mediante en GPBGA.

vector de afinidad de mayor dimensión, en cambio, esta tarea sería más compleja y deberíamos permitir un grado de libertad mayor a los grupos aumentando su tamaño.

En la gráfica superior de la figura 40, hemos representado la salida z que proporcionan 4 individuos que poseen afinidad -1 y en la gráfica inferior 4 individuos que poseen afinidad 1 para el mismo muestreo de la variables de entrada. Los 8 individuos han sido tomados de forma aleatoria de la población, es decir, no son los mejores y simplemente queremos comprobar que realmente pertenecen a las mismas

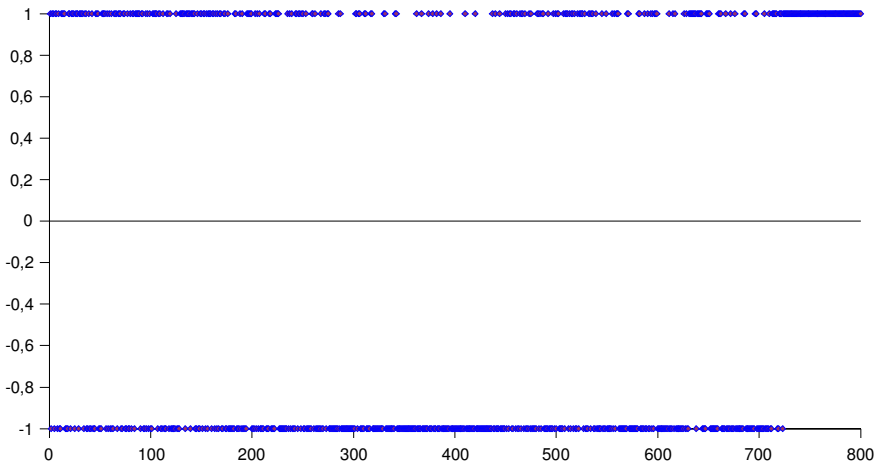


Figura 39. Distribución de los valores de afinidad de los 800 individuos de la población

especies. Este hecho queda confirmado al analizar las gráficas porque los 4 casos superiores son análogos y los 4 inferiores también. Por tanto, las dos afinidades que obtenemos realmente corresponden a individuos diferentes aunque, lógicamente, no todos son igual de buenos de cara a la formación del grupo final.

Nos falta por comprobar la parte más interesante de este experimento, y es comparar estas dos funciones que obtienen cada una de las especies con las dos funciones originales (gaussiana y senoidal) que componen la función objetivo. En la figura 41, hemos representado la salida z que proporcionan los dos individuos que forman el mejor grupo obtenido, en la gráfica superior el individuo con afinidad -1 y en la inferior con afinidad 1. Estos dos individuos unidos (sumando sus salidas) obtienen la predicción de la función objetivo mostrada en la figura 38. Además, hemos representado en la gráfica superior la salida z correspondiente a la función senoidal y en la gráfica inferior la salida z para la función gaussiana. Lo hemos hecho así después de comprobar la similitud entre unas y otras, tal y como se observa en ambas gráficas. Por tanto, el

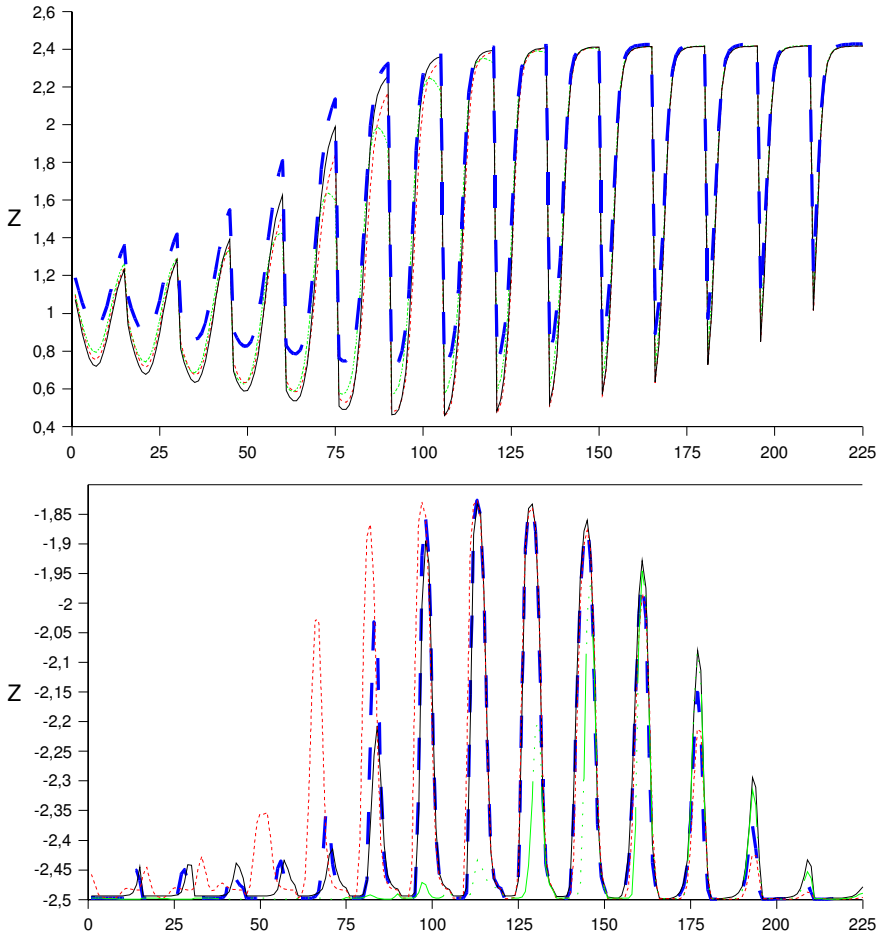


Figura 40. Salida z que proporcionan 4 individuos que poseen afinidad -1 tomados de forma aleatoria de la población (gráfica superior) y 4 individuos que poseen afinidad 1 tomados también de forma aleatoria (gráfica inferior) para el mismo muestreo de la variables de entrada.

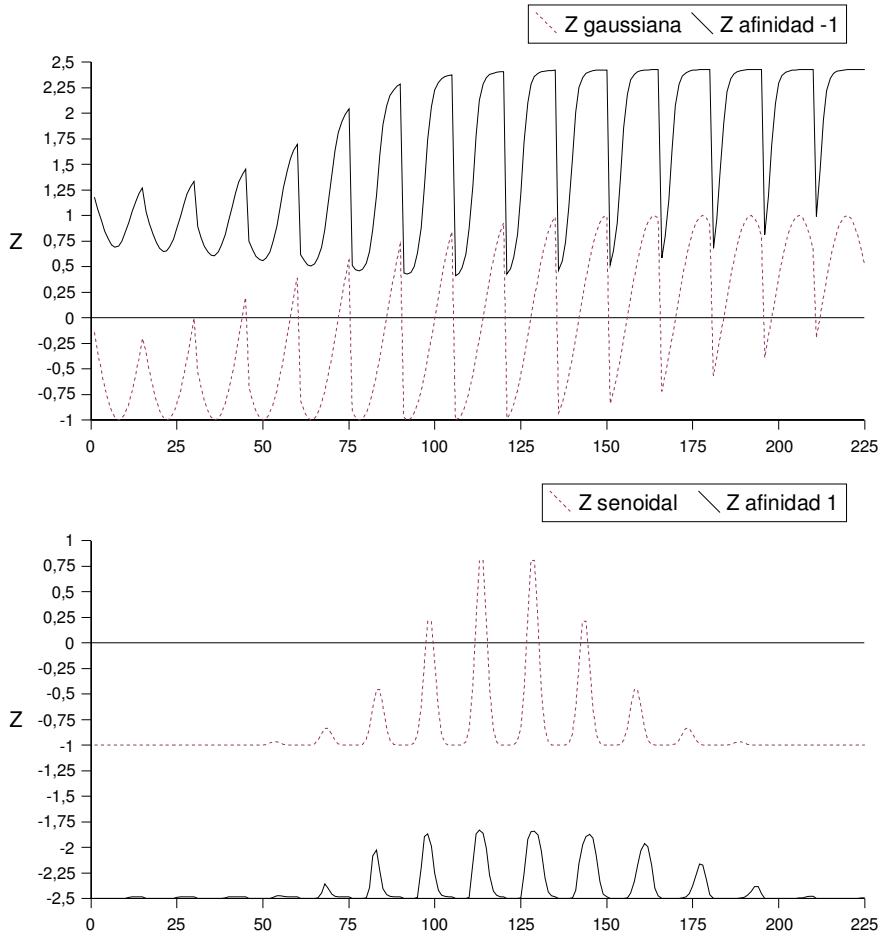


Figura 41. En la gráfica superior se representa la salida z que proporciona el individuo con afinidad -1 del mejor grupo frente a la salida z que proporciona la función senoidal teórica z = sen(x+y). En la gráfica inferior se representa la salida z del individuo con afinidad 1 del mejor grupo frente a la salida z de la función gaussiana $z = 2*exp(-5*(x^2+y^2)) - 1$.

GPBGA ha obtenido de forma autónoma las dos subfunciones que forman la global y, además, son muy similares a las funciones teóricas que hemos utilizado. Lógicamente esto no tiene por qué ser así en todos los casos dado que las redes neuronales pueden aproximar cualquier función mediante distintas combinaciones lineales.

Vemos, además, en la figura 41 como la función asociada al individuo de afinidad -1 está desplazada hacia arriba en el eje Z mientras que la asociada al individuo con afinidad 1 está desplazada hacia abajo. El resultado de la suma de ambas en el grupo estará centrado en el mismo rango que la función objetivo (figura 37). Este resultado profundiza de nuevo en el hecho de que el GPBGA posee total libertad para llegar a la función objetivo, tanto en los rangos utilizados como en las propias funciones que puede obtener.

Conclusiones al estudio de la generación de grupos por afinidad

En este apartado, hemos planteado una generalización para el algoritmo evolutivo PBGA basada en la formación de especies de individuos en la población usando para ello un nuevo concepto: la afinidad. La afinidad de un individuo representa sus preferencias a la hora de unirse a otro u otros individuos en grupos y así colaborar para lograr la meta común. Hemos desarrollado un método para la formación automática de grupos concretando el concepto de afinidad en un vector asociado a cada individuo.

Desde un punto de vista evolutivo, ahora el PBGA está más cercano a lo que ocurre en la evolución natural, donde se forman agrupaciones de individuos para lograr un bien comunitario. Existen unos criterios funcionales y subjetivos que controlan el modo en que se forman los grupos en la naturaleza, criterios que hemos tratado de reproducir con el vector de afinidad. Así, aparecen especies de individuos que se especializan en ciertas tareas o funciones en su comunidad o población, pero de una forman natural, sin imposiciones externas como ocurría, por ejemplo, en la selección por salidas del PBGA.

Al extender este algoritmo generalizado al MDB, se producen dos cambios fundamentales. Dado que los individuos son ahora grupos de individuos, la Memoria a Largo Plazo que los almacena será más compleja y el mecanismo de gestión deberá ser ajustado al nuevo tipo de información que almacena (no son simples modelos). Pero lo más interesante es que los grupos están formados por submodelos de los modelos de mundo e internos, es decir, por elementos básicos que conforman una parte del entorno y del propio agente. En general, tras un proceso de aprendizaje con el MDB para un agente autónomo, su MLP almacenará las partes básicas en las que se fundamenta su entorno y su estado interno. De este modo, será posible el aprendizaje de nuevos entornos y motivaciones por combinación de los elementos básicos de los que se dispone en la MLP. Estamos reincidiendo en la idea de que aprender es muy costoso, de tal forma sería ideal poder almacenar lo aprendido en su forma más simple y distribuida, de tal manera que en futuros problemas se pueda recurrir a esa información y combinarla para lograr comportamientos complejos.

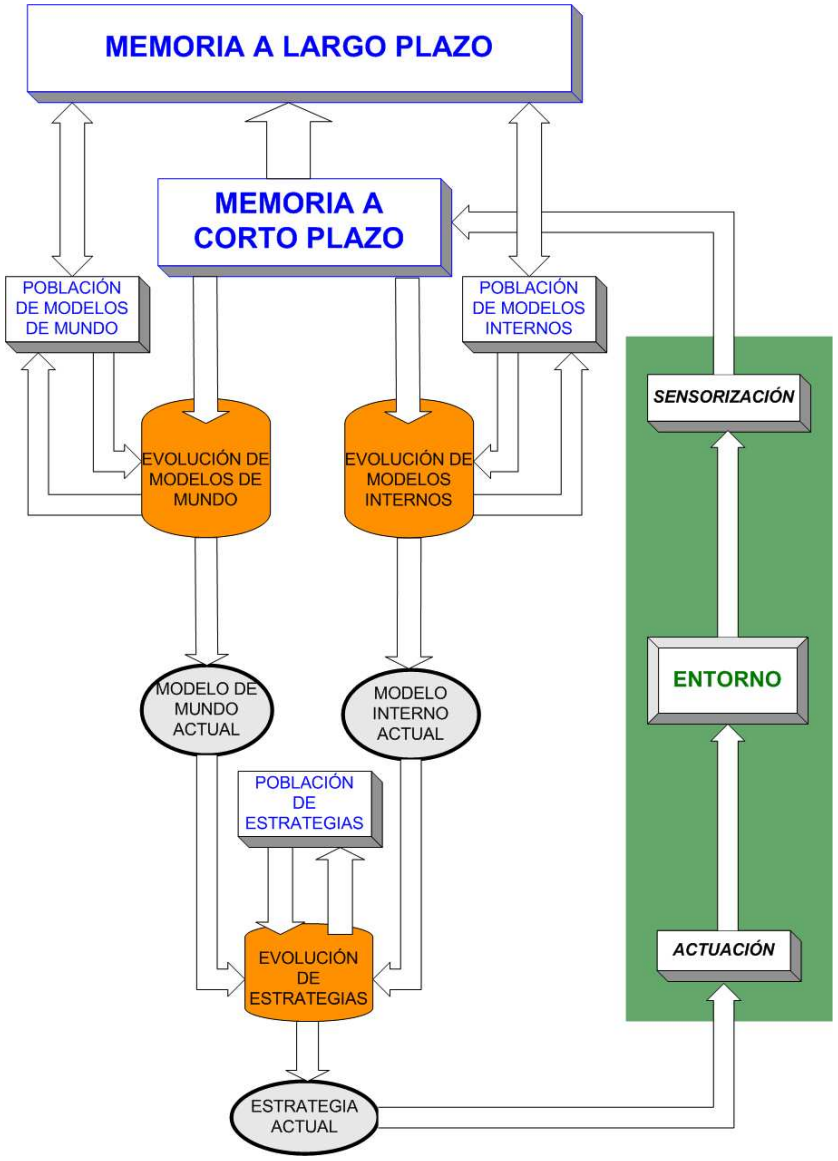
Por tanto, los cambios que introduce el GPBGA en la obtención de los modelos, nos permiten plantear las bases de una versión más avanzada de todo el mecanismo que se fundamenta en la obtención y combinación automática de información aprendida lo más básica posible.

Como se puede observar, el tema de la formación de grupos y sus implicaciones es muy interesante y profundo. De hecho, en el presente estudio nos hemos quedado en una primera implementación que continuaremos mejorando en el futuro. La combinación de calidades en los grupos con distintos operadores que la suma, la posibilidad de obtener grupos de tamaño uno, la eliminación de un tamaño máximo en los grupos, etc son problemas que tendremos que resolver. Pero también aquí reside el principal interés de este tema, y es que aún quedan muchos interrogantes por contestar hasta lograr el objetivo final de obtener un algoritmo evolutivo que se comporte de forma idéntica a los procesos naturales, guiado únicamente por la función de calidad.

A continuación realizaremos un breve resumen de los resultados obtenidos mediante el diagrama de bloques final del mecanismo.

6.4 Diagrama de bloques final del MDB

A lo largo de este apartado, hemos ido analizando los elementos básicos del MDB así como su funcionamiento. Finalmente, el diagrama de bloques que resume dicho funcionamiento es el siguiente:



Los bloques *Sensorización*, *Entorno* y *Actuadores* conforman la operación externa del mecanismo, y no dependen para nada de lo que pueda ocurrir internamente. Por este motivo, para separar visualmente la operación interna de la externa, hemos subrayado la operación externa resaltando la existencia de dos procesos fundamentales en el mecanismo:

1. **Proceso reflexivo:** en el que se prueban internamente las posibles estrategias tras buscar los mejores modelos. Se "piensan" las posibles acciones a realizar en función del conocimiento interno y externo del mundo en forma de modelos, paralelamente a un proceso de mejora de dicho conocimiento del mundo.
2. **Proceso interactivo:** en el que se interacciona con el mundo aplicando estrategias y percibiendo sensorialmente sus consecuencias.

Recordamos de nuevo el ciclo de funcionamiento básico del MDB teniendo en cuenta las mejoras que le hemos añadido en nuestro estudio:

1. La *Estrategia* seleccionada es aplicada al *Entorno* a través de los actuadores del agente, representados en el bloque *Actuación*, obteniendo una *Sensorización* como consecuencia.
2. Los datos de actuación realizada y sensorización obtenida proporcionan un par acción-percepción que es almacenado en la *Memoria a Corto Plazo*.
3. Sobre el nuevo par acción-percepción se aplica el criterio de inestabilidad que detecta cambios relevantes en las funciones que los modelos deben aproximar y actúa sobre los contenidos de la MCP sin variar el funcionamiento conceptual del mecanismo.
4. Los modelos contenidos en la *Memoria a Largo Plazo* son inyectados en las poblaciones de los algoritmos evolutivos.
5. Con los pares contenidos en esta memoria se llevan a cabo los procesos de *Evolución del Modelo de Mundo* y *Evolución del Modelo Interno* tratando de maximizar las funciones reales representadas por los pares acción-percepción que almacena la *Memoria a Corto Plazo*. Para ello, en este trabajo, se proponen los algoritmos PBGA y su extensión, el GPBGA.
6. Los mejores individuos de las poblaciones de los algoritmos evolutivos se marcan como *Modelo de Mundo Actual* y *Modelo Interno Actual* y se utilizan en la *Evolución de Estrategias* (en general, optimización de estrategias). Además, se comprueba si cumplen el criterio de estabilidad para ser almacenados en la MLP. En caso afirmativo se produce un proceso de reemplazo en dicha memoria.
7. La mejor de las estrategias, *Estrategia Actual*, es aplicada al *Entorno* en una etapa de *Actuación* obteniendo una nueva *Sensorización*.

Damos así por finalizado este extenso apartado 6 dedicado a la implementación práctica del MDB y al estudio detallado de sus principales elementos. Pasamos a continuación a presentar los ejemplos de aplicación del mismo sobre robots reales.

Aplicación

7 Aplicación

El Mecanismo Cognitivo MDB ha sido desarrollado para ser aplicado a cualquier tipo de agente autónomo, tanto software como hardware. Como indicamos en el apartado 4.3, como plataforma de implementación hemos utilizado agentes hardware, es decir, robots autónomos, por sus especiales características de autonomía computacional, carencias sensoriales, etc. Por este motivo, al tratar los detalles prácticos del MDB, hemos recurrido a ejemplos relacionados con robots.

De cualquier forma, la aplicación del mecanismo a agentes software es inmediata, es más, nos evitamos muchos de los problemas derivados de la sensorización deficiente que tienen los robots y, sobre todo, pasamos a tratar con entornos más simples (tanto virtuales como reales, pero simplificados). La única parte del MDB que depende del tipo de agente es la operación externa, donde se aplican las acciones y se obtienen nuevos valores de sensorización. En cuanto a la operación interna, dados unos pares acción-percepción, es la misma para cualquier tipo de agente y situación.

Teniendo esto en cuenta, volvemos sobre los agentes hardware para presentar tres ejemplos prácticos de aplicación del MDB en casos reales. Los dos primeros se centran en el funcionamiento general del mecanismo de cara a mostrar si el concepto desarrollado es correcto. Por este motivo son ejemplos simples desde el punto de vista de la tarea a realizar y de los algoritmos utilizados, ya que hemos aplicado un algoritmo genético clásico y una estrategia de reemplazo puramente temporal de tipo FIFO para la Memoria a Corto Plazo. La realización de estos experimentos nos ayudó a la hora de encontrar deficiencias en las primeras versiones y así surgieron tanto el algoritmo PBGA como la estrategia de reemplazo para la Memoria a Corto Plazo y la utilización de la Memoria a Largo Plazo. Estos dos primeros ejemplos fueron realizados sobre el robot hexápodo Hermes II, que será analizado en el siguiente apartado.

El tercer ejemplo incluye todos los elementos presentados en este trabajo (excepto la creación de grupos por afinidad), es decir, utiliza el modelo de mecanismo presentado en el apartado 6.4. Es un ejemplo de mayor complejidad conceptual y de más alto nivel deliberativo. Para llevarlo a cabo utilizamos el Pioneer 2, que es un robot rodado.

7.1 Aplicación del MDB en comportamientos primarios

En el primer ejemplo pretendemos que un robot hexápodo aprenda de forma autónoma una combinación adecuada de fases iniciales en sus patas de modo que pueda caminar correctamente; en segundo lugar utilizaremos la forma de caminar obtenida para llevar a cabo de forma autónoma una tarea de aprender a girar en busca de un objetivo. Ambos son comportamientos primarios desde un punto de vista intelectual.

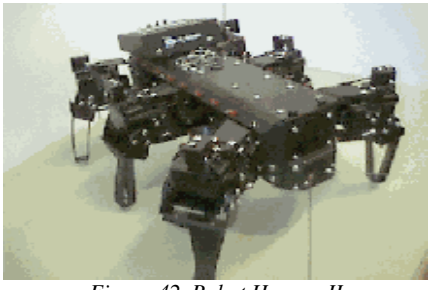


Figura 42. Robot Hermes II

7.1.1 Hermes II

El robot que utilizamos es un modelo hexápodo llamado Hermes II, fabricado por IS Robotics y que se muestra en la figura 42. Se caracteriza fundamentalmente por su robustez mecánica y por el gran número de sensores de que está dotado: seis sensores infrarrojos situados encima de cada pata, un sensor que indica el grado de inclinación horizontal y otro vertical, un sensor de fuerza en cada pata y dos sensores de contacto

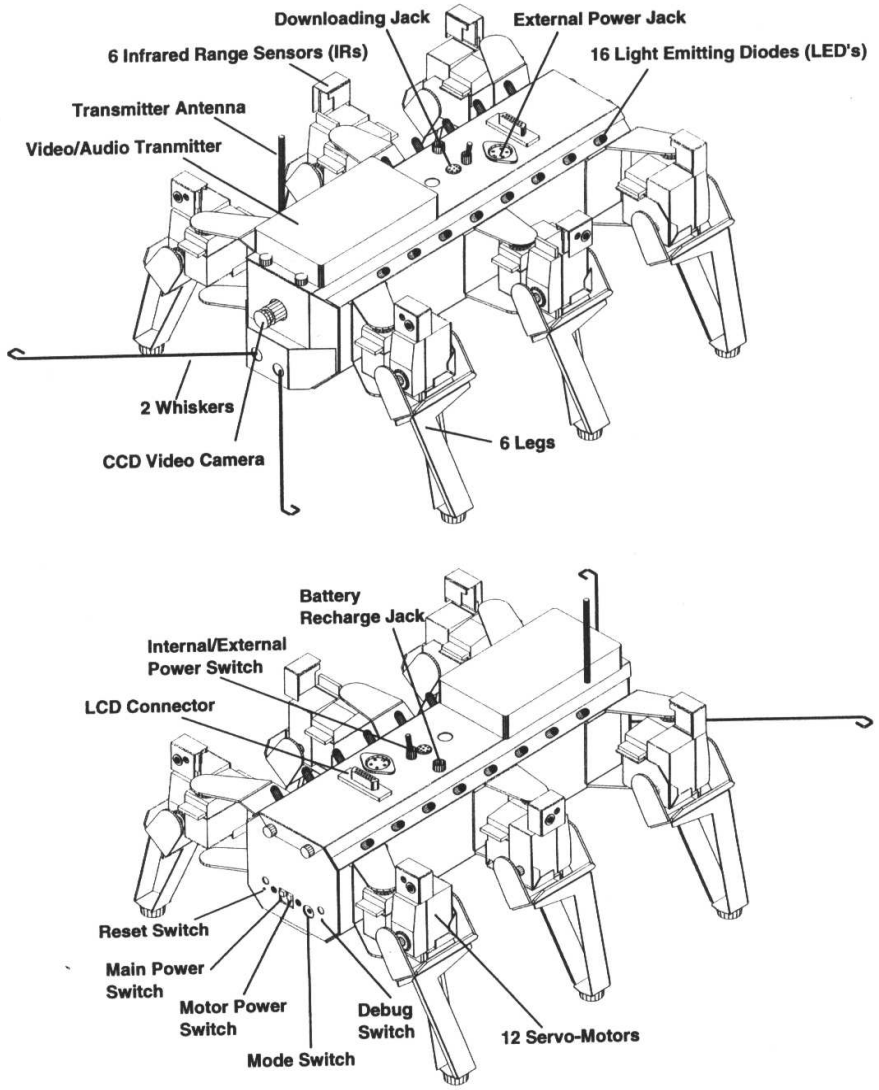


Figura 43. Esquema de la distintas partes del robot Hermes II

(whiskers). La figura 43 muestra un esquema de las principales características que posee el robot. Su principal problema es que presenta deficiencias en la sensorización a la hora de llevar a cabo tareas de navegación por entornos reales, ya que los sensores infrarrojos resultan poco precisos. Esto limita enormemente la dificultad de las tareas que podemos realizar.

La aplicación directa del MDB en el propio Hermes II no es posible dadas las limitaciones computacionales del microcontrolador que posee, de forma que debemos separar la operación externa, en el robot, de la interna, en una computadora. Así, cada vez que el mecanismo selecciona una estrategia, esta es aplicada al entorno mediante el Hermes II obteniendo nuevos valores de sensorización. Esta estrategia experimental tiene el problema fundamental de que los movimientos iniciales son erráticos e impredecibles (mientras se están aprendiendo los modelos) y dado que el robot está unido a la computadora por un cable, tenemos muy limitada la operación. Por esta razón, hemos decidido trabajar con un modelo, lo más real posible, en simulación. Debemos recalcar que el modelado dinámico y mecánico de un robot hexápodo no es una tarea simple, y gran parte del trabajo de este ejemplo ha consistido en alcanzar el modelo óptimo. Dicho modelo ha sido generado en un simulador mecánico CAD en 3D llamado DADS, del que haremos una breve introducción.

7.1.2 El simulador DADS

Es una herramienta informática de simulación que estudia el comportamiento de sistemas mecánicos. El proceso de modelado se realiza por grupos de elementos a través de una interfaz gráfica denominada DADSModel donde se crean modelos físicos completos incluyendo todas las fuerzas, pares, tensiones, etc a las que va a estar sometido el sistema mecánico.

Las ecuaciones matemáticas se forman internamente y son resueltas usando un algoritmo exacto, eficiente y estable. Las posiciones, velocidades, aceleraciones y fuerzas de reacción son calculadas para cada una de las partes constitutivas del modelo. Los resultados del análisis pueden ser analizados con la ayuda de gráficos construidos por el propio módulo de DADS DADSGraph o, en su defecto, apoyándonos en la animación que DADSModel proporciona para la visualización del comportamiento del sistema.

En la figura 44 mostramos el simulador con el modelo del Hermes cargado. A la hora de utilizar el simulador integrado con el mecanismo, utilizamos el lenguaje de programación y control Matlab, que es capaz de manejar los modelos generados en DADS y simularlos dadas unas ciertas condiciones iniciales. Cada vez que el mecanismo selecciona una estrategia a aplicar en el mundo real, a través de un programa construido en Matlab, simulamos el modelo generado en DADS. Ha sido muy costoso conseguir un modelo mecánico completo del Hermes debido al gran número de articulaciones que posee y a la escasa simetría de su estructura, pero los resultados son muy satisfactorios y refleja a la perfección el comportamiento real como veremos más adelante.

En la figura 45 podemos ver, en la imagen de la izquierda, el modelo en simulación del Hermes II que hemos utilizado en este ejemplo. Los resultados que obtengamos en el simulador serán cotejados en el robot real que también se muestra en la imagen de la derecha de dicha figura 45. Para una explicación más profunda del simulador y de sus posibles aplicaciones recomendamos [Lamas, 99].

Figura 44. Entorno gráfico del simulador utilizado para el Hermes II

Pasamos ahora al primer ejemplo de aplicación del MDB donde pretendemos que el Hermes II aprenda por sí mismo a caminar.

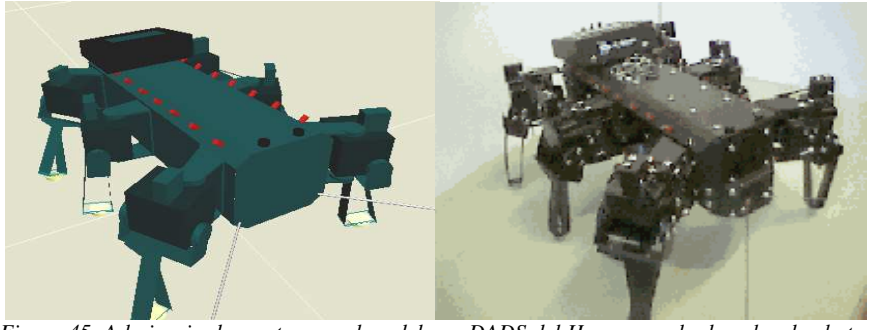


Figura 45. A la izquierda mostramos el modelo en DADS del Hermes y a la derecha el robot real en el que se inspira.

7.1.3 Aprendiendo a caminar

En este experimento, buscamos que un robot hexápodo encuentre la combinación de fases iniciales en el movimiento de sus patas para poder avanzar y alcanzar un objeto (un bloque), de modo que la motivación del comportamiento consiste en maximizar la percepción de sus sensores infrarrojos.

Cada pata del Hermes II posee dos grados de libertad, uno realiza un movimiento de oscilación (swing) y el otro uno de elevación (lift) como se muestra en la figura 46.

Debido a la amplia difusión del término en la bibliografía especializada y por compacidad, nos referiremos de aquí en adelante a la forma de caminar, entendida como la combinación de fases y amplitudes en los motores de las patas, como *gait*. El movimiento que proporciona cada uno de los motores se puede aproximar por una función senoidal ($x = Asen(\omega t + \phi)$) entre los límites amplitud A, con una frecuencia ω y una fase inicial en cada pata ϕ.

En este primer ejemplo, fijamos todos los parámetros en el motor de elevación y fijamos amplitud y frecuencia en el de oscilación, con lo que la única variable que utilizamos es la fase inicial ϕ. Ésta es la que diferencia un gait de otro, ya que las patas inician sus movimientos antes o después en el tiempo. Utilizamos, por tanto, estrategias formadas por una única acción.

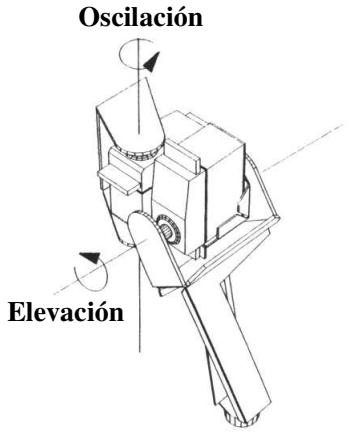


Figura 46. Giros posibles en cada pata del Hermes II, Oscilación y Elevación

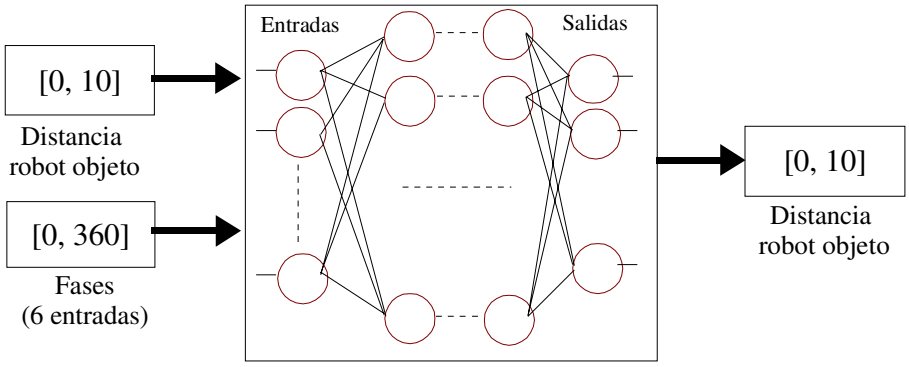
Entre los gaits comunes que podemos hallar en la bibliografía cuando se trata de analizar la forma de caminar de los hexápodos, destacan el gait trípode y el gait onda [Beer, 92]. Ambos están basados en la forma que tienen de caminar los insectos y, en el caso del trípode, consiste en que las patas consecutivas se muevan en contrafase, de forma que siempre hay 3 en el suelo a la vez asegurando la estabilidad del hexápodo (las dos de un lado del cuerpo y la central del lado contrario). En el caso del onda, las patas se mueven como una onda en cada lado del cuerpo, subiendo y bajando de forma consecutiva.

Volviendo al ejemplo en concreto, el robot parte de una posición fija y tiene el bloque a un metro de distancia delante de él. Dejamos que la ejecución de cada gait dure 13 segundos en simulación, que se traducen en unos 6 segundos reales. En función de cuán buena sea la elección de las fases iniciales, se acercará más o menos al bloque objetivo y eso le sirve como función de calidad para las estrategias.

Modelos y estrategias

El modelo de mundo utilizado en este ejemplo tiene 7 entradas y una única salida. La primera entrada corresponde a la distancia del cuerpo del robot al objeto (bloque). El robot utiliza los 6 sensores infrarrojos de que dispone para detectar el bloque, pero son

poco precisos. Para solventar esta deficiencia, hemos desarrollado un sensor virtual que a partir de las lecturas de un sensor en un barrido de la pata, devuelve la distancia y el ángulo de objeto sensado. Los detalles sobre la implementación de este sensor están explicados en el Apéndice A. El resultado final es que disponemos de la distancia y el ángulo del objeto respecto al robot, y con suficiente precisión. Hemos codificado la distancia entre 0 y 10 de forma que distancias inferiores a 30 cm toman el valor 10 y distancias superiores a 100 cm toman el valor 0 (tenemos una especie de sensor de proximidad). Estos límites de 30 y 100 cm los impone el robot real ya que por debajo y por encima no detecta nada. Las otras seis entradas al modelo de mundo son las fases iniciales, una por pata, que están codificadas en un rango entre 0 y 360 (entre 0 y 2Π realmente, por ser el argumento de la función $x = Asen(\omega t + \phi)$). La salida del modelo de mundo es la distancia predicha y se mueve en un rango de 0 a 10, al igual que en la entrada. Podemos ver cómo queda el modelo de mundo de forma esquemática con los rangos de entradas y salidas:



El modelo interno es trivial en este caso y utilizamos como estado interno la propia distancia predicha, por lo que su rango varía entre 0 y 10 de nuevo. La maximización del estado interno consiste, pues, en minimizar la distancia al objeto (de acuerdo con la motivación establecida de maximizar la sensorización de los infrarrojos) sin necesidad de un modelo interno explícito. Esta simplificación no afecta a la estructura ni al funcionamiento del mecanismo.

El algoritmo evolutivo utilizado para la obtención de los modelos de mundo y de las estrategias es un algoritmo genético simple que posee características básicas que a continuación se muestran. Para los **modelos de mundo**:

- Tamaño de la red que representa el modelo de mundo: 7 neuronas de entrada, dos capas ocultas de 4 neuronas cada una y 1 neurona de salida. Evolucionamos los pesos de la red (7x4 + 4x4 + 4x1 = 48 genes) además de un bias por cada neurona de las capas de entrada e intermedias (4 + 4 + 1 = 9 genes). Por tanto, los cromosomas están formados por 57 genes (números reales).
- Escogemos una población de un orden de magnitud por encima del número de genes, en este caso de 700 individuos.
- El cruce se realiza por dos puntos con una probabilidad del 60%.

- La mutación consiste en sumar al gen correspondiente un número aleatorio entre -1 y 1 elevado al cubo. Obtuvimos los mejores resultados para una probabilidad del 2%.
- Conservamos el mejor individuo de una generación a la siguiente (elitismo).
- Utilizamos selección por torneo con una ventana de tamaño 2 (baja presión selectiva).

Los parámetros del algoritmo genético que mejor resultado nos han dado en la evolución de las **estrategias** son los siguientes:

- Las estrategias son conjuntos de 6 valores que se evolucionan directamente, por tanto cromosomas de 6 genes.
- La poblaciones utilizadas fueron de 120 individuos.
- El cruce se realiza por dos puntos con una probabilidad del 60%.
- La mutación es idéntica al caso de los modelos de mundo, por lo que antes de aplicar este operador normalizamos los genes entre -1 y 1. Obtuvimos los mejores resultados con probabilidad del 6%.
- Conservamos el mejor individuo de una generación a la siguiente.
- Utilizamos selección por torneo con una ventana de tamaño 2 (baja presión selectiva).

Para llegar a establecer estos parámetros fue necesaria una etapa de pruebas donde, sobre todo, tuvimos que utilizar distintos tamaños de red hasta dar con el más adecuado en relación al error que proporcionaba y a su tamaño. Esta etapa de pruebas se verá reducida al mínimo con la inclusión del algoritmo PBGA, como veremos en el ejemplo del apartado 7.2. En cuanto a la Memoria a Corto Plazo, utilizamos una estrategia de reemplazo temporal pura de tipo FIFO con 40 muestras.

Por último, en este ejemplo no hemos utilizado Memoria a Largo Plazo porque no existe cambio en el entorno ni en el estado interno, de modo que no existe ningún modelo que reutilizar.

Resultados

En la figura 47 representamos la evolución del error cuadrático medio entre la predicción que realiza el modelo de mundo actual en cada iteración y los datos reales almacenados en la MCP. Vemos como, a partir de la iteración 110, el error cuadrático medio es inferior a 0.5. Hemos comprobado que este valor es el límite para que el robot escoja gaits buenos o malos. Los sensores infrarrojos reales no detectan un objeto si se encuentra por debajo de 30 cm y, hasta alrededor de la iteración 110, las combinaciones de fases que aplica no le llevan tan cerca. Esta distancia mínima es la que utilizamos para decidir si un gait es bueno o malo, y lo tomamos como valor máximo del estado interno.

A la hora de aplicar las fases al robot real hemos utilizado la nomenclatura que se muestra en la figura 48, de modo que la fase 0 se aplica a la pata 0 y así sucesivamente.

En la figura 49 vemos la función senoidal que representa el movimiento del motor de oscilación de cada pata para el gait trípode donde, como se observa, el parámetro diferenciador es la fase inicial. Utilizaremos este tipo de diagrama para representar los

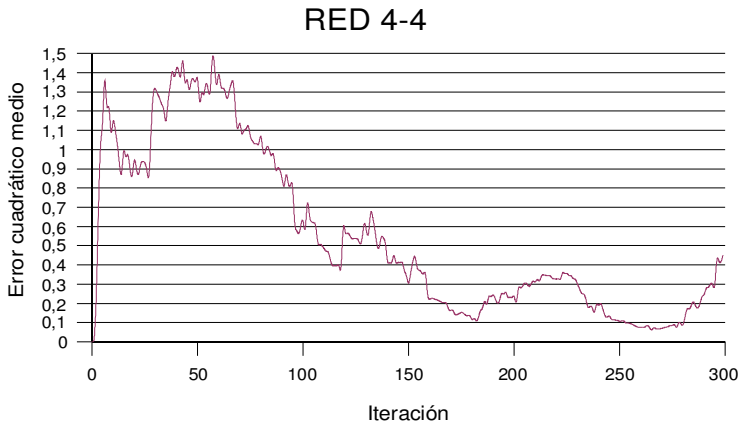


Figura 47. Evolución del error cuadrático medio entre la predicción que realiza el mejor modelo de mundo y los datos reales almacenados en la MCP en cada iteración del MDB

distintos gaits obtenidos por el mecanismo. El simulador aplica estas funciones al motor de oscilación, de modo que está permitida cualquier combinación de fases. La única limitación que nos impone el simulador es que si las diferencias entre fases son iguales, el gait obtenido va a ser análogo independientemente del valor, es decir, una

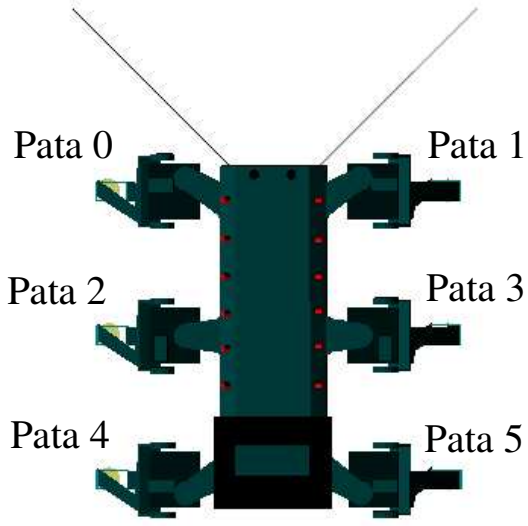


Figura 48. Nomenclatura utilizada en el ejemplo a la hora de asignar las fases a las patas del robot Hermes II.

combinación (0-180-180-0-0-180, gait trípode) es equivalente a una (180-0-0-180-180-0), que mantiene la misma diferencia entre fases aunque cada valor ha aumentado en 180.

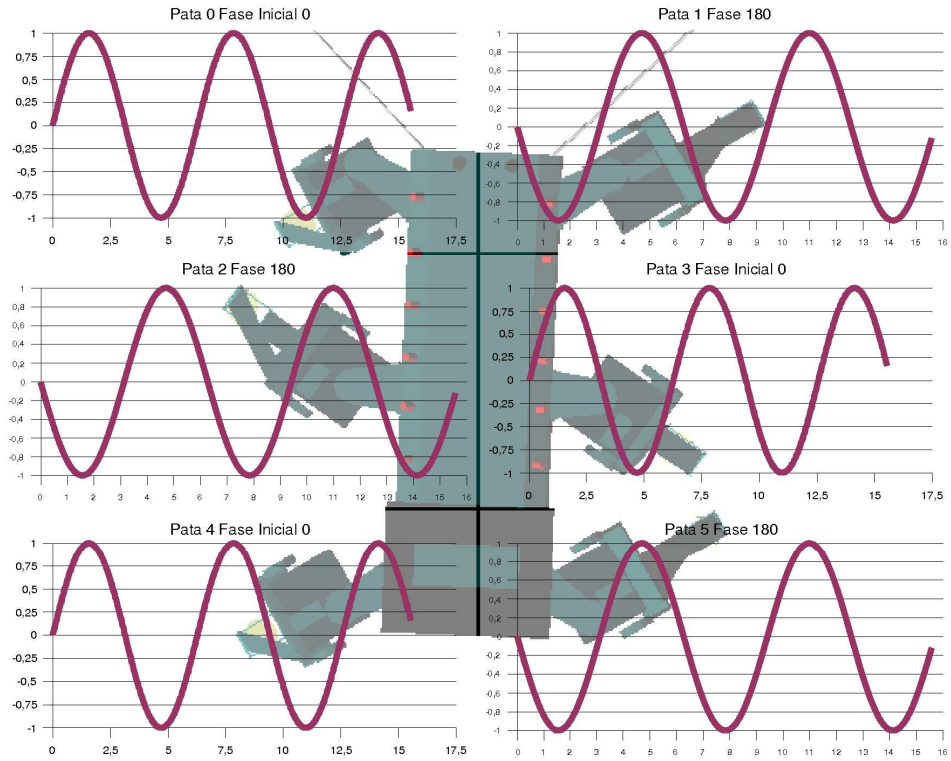


Figura 49 Función senoidal que representa el movimiento de oscilación de cada pata para el gait trípode. Las patas del robot empiezan como muestra el esquema.

En la figura 50 se representa un dato real, esto es, el gait correspondiente a la iteración 6, (270-76-63-143-152-90) que lleva al robot a una distancia de 90 cm del bloque (recordamos que el robot parte de una distancia de 100 cm). En esta iteración, el error cuadrático medio que proporciona el modelo de mundo es 1.36, lo que significa que puede fallar hasta en 136 cm su predicción. La segunda estrategia que tomamos como ejemplo (131-310-90-158-184-90), se ejecuta en la iteración 82 donde el error cuadrático medio es de 0.98 y se muestra en la figura 51. Su ejecución lleva al robot a una distancia de 51 cm del bloque, mejorando mucho los resultados iniciales. Por último, en la iteración 148 el error cuadrático medio es de 0.37 y se aplica la estrategia (270-90-110-270-270-90), que es casi un gait trípode (sólo cambia la fase de la pata 2 que debería ser 90). Este gait lleva al robot a una distancia de 28 cm del bloque, que es el objetivo buscado y la hemos representado en la figura 52. Como se observa, una vez que la evolución llega a un error cuadrático medio para los modelos de mundo del orden de 0.2, las estrategias llevan siempre al objetivo, confirmando el hecho de que la selección de acciones correcta depende básicamente del grado de fiabilidad y aprendizaje de los modelos y, en menor medida, del propio proceso de selección de estrategias (en general más simple).

Para obtener una estimación más exacta de la mejora en los gaits, definimos la *eficiencia* de un gait como la distancia normalizada que recorre en línea recta el robot ponderada por la distancia que se desvía horizontalmente. En el mejor de los casos, con el tiempo de ejecución que hemos fijado, el robot recorre 80 cm en línea recta y alcanza

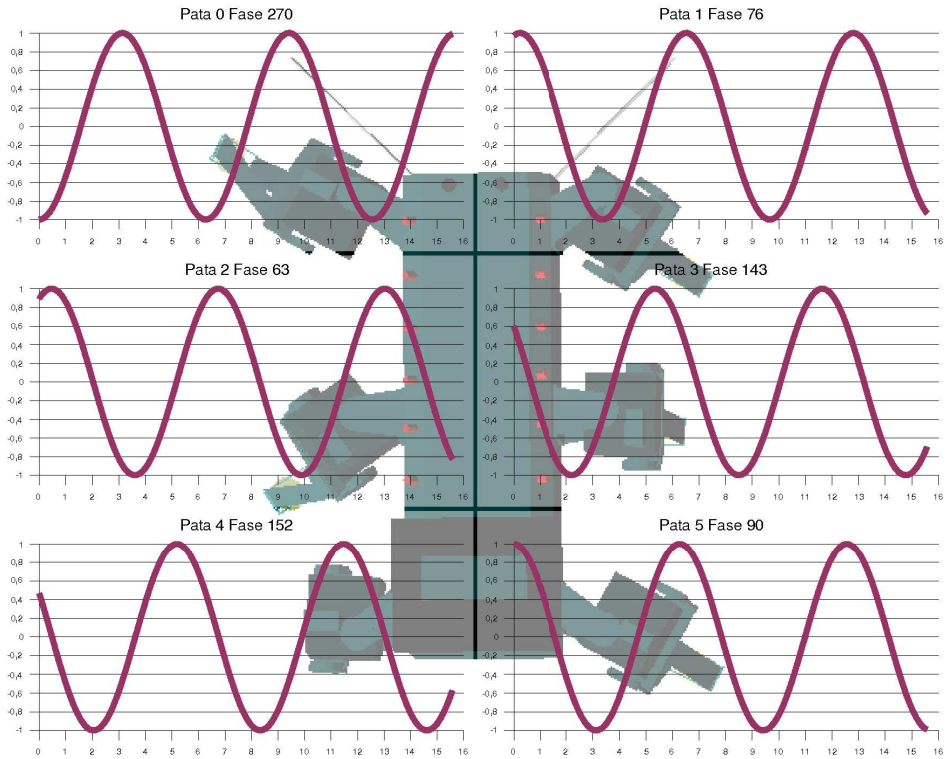


Figura 50. Función senoidal que representa el movimiento de oscilación de cada pata para la estrategia 6: 270-76-63-143-152-90. Las patas del robot empiezan como muestra el esquema.

el bloque (realmente se queda a 20 cm de él) sin desviarse horizontalmente. Este gait obtendría un valor de eficiencia 1. Cualquier otro gait que no llegue a 80 cm tendrá un valor inferior y además será menor todavía si se desplaza hacia los lados.

Con estos datos datos hemos realizado la gráfica de la figura 53, donde vemos como la eficiencia tiende a 1 al aumentar las iteraciones. Hemos trazado una línea de tendencia para ilustrar con más claridad el comportamiento global de la gráfica. Este resultado confirma lo que venimos diciendo y, a medida que la interacción con el mundo es mayor, los gaits que ejecuta el robot son más eficientes.

Pasamos a implementar en el robot real estos resultados obtenidos en simulación. Para ello, hemos tomado las 86 primeras estrategias aplicadas y las hemos ejecutado en el Hermes II. Esto refleja el proceso de aprendizaje que ha seguido el robot, de modo que, en estas 86 iteraciones, aplica estrategias erróneas. También hemos implementado estrategias en torno a la iteración 300 donde el gait aplicado ya es un trípode perfecto.

En la figura 54 vemos dos secuencias de cuatro imágenes que corresponden a la estrategia aplicada en la iteración 27. Las imágenes de la izquierda están sacadas del simulador y las de la derecha de la ejecución en el robot real. Hemos tomado 4 instantes repartidos proporcionalmente en un periodo completo de movimiento, es decir, cada pata realiza un ciclo completo con la fase inicial que le corresponda. En esta figura vemos cómo, en las primeras iteraciones, las patas no guardan ningún tipo de sincronía y el gait es malo. El robot gira debido a esa mala elección de fases en las patas y, como

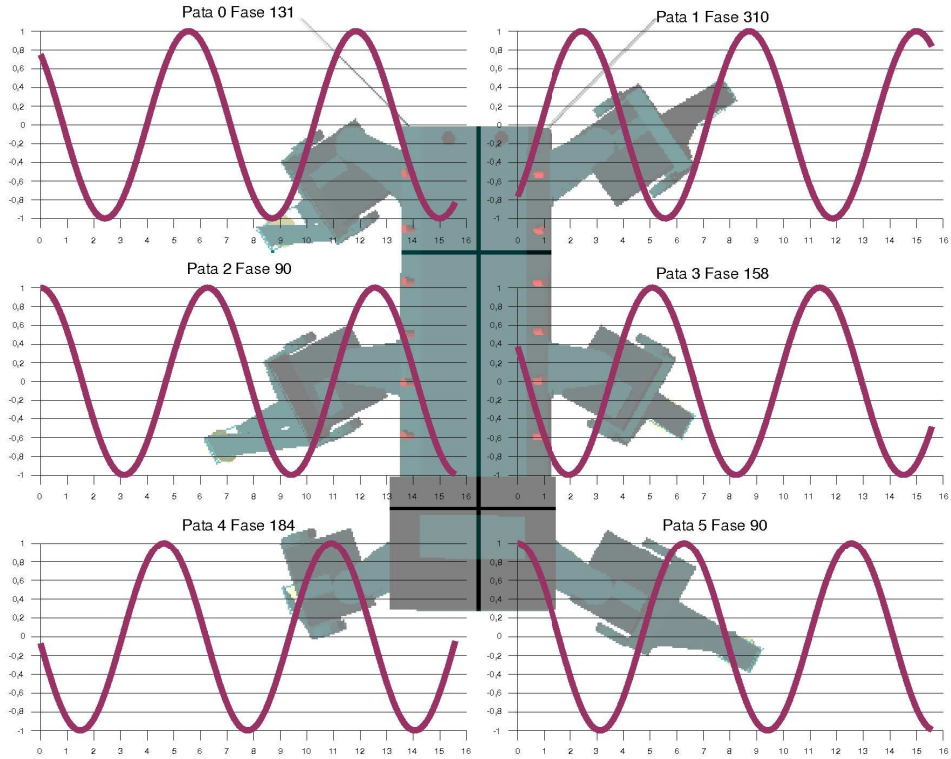


Figura 51. Función senoidal que representa el movimiento de"oscilación de cada pata para la estrategia 82: 131-310-90-158-184-90. Las patas del robot empiezan como muestra el esquema.

es lógico, la eficiencia es cero (distancia recorrida = 0, desplazamiento lateral = 0.364 cm a la izquierda, eficiencia = 0).

En las figuras 55 y 56 mostramos el gait aplicado en las iteraciones 71 y 300. En la 71, el gait no es bueno del todo porque, aunque avanza mucho (distancia recorrida = 0.73, desplazamiento lateral = 0.014 cm a la izquierda, eficiencia = 0.9), lo hace "trastabillándose" y esto implica oscilaciones laterales serias. Tal y como hemos visto en simulación, en la iteración 300 el gait es un trípode perfecto y la eficiencia es 1 (distancia recorrida = 0.8, desplazamiento lateral = 0.02 cm a la derecha, eficiencia = 1).

Como conclusión a este primer ejemplo, diremos que el funcionamiento general del mecanismo es correcto aplicado a un problema basado en un robot real donde la complejidad del modelo de mundo es alta. El comportamiento obtenido es útil en el mundo real para el robot Hermes II, ya que hemos obtenido un gait muy eficiente. El simulador utilizado resulta una herramienta muy potente por su enorme similitud con el robot real y, aunque las simulaciones son lentas, el resultado compensa el gasto computacional.

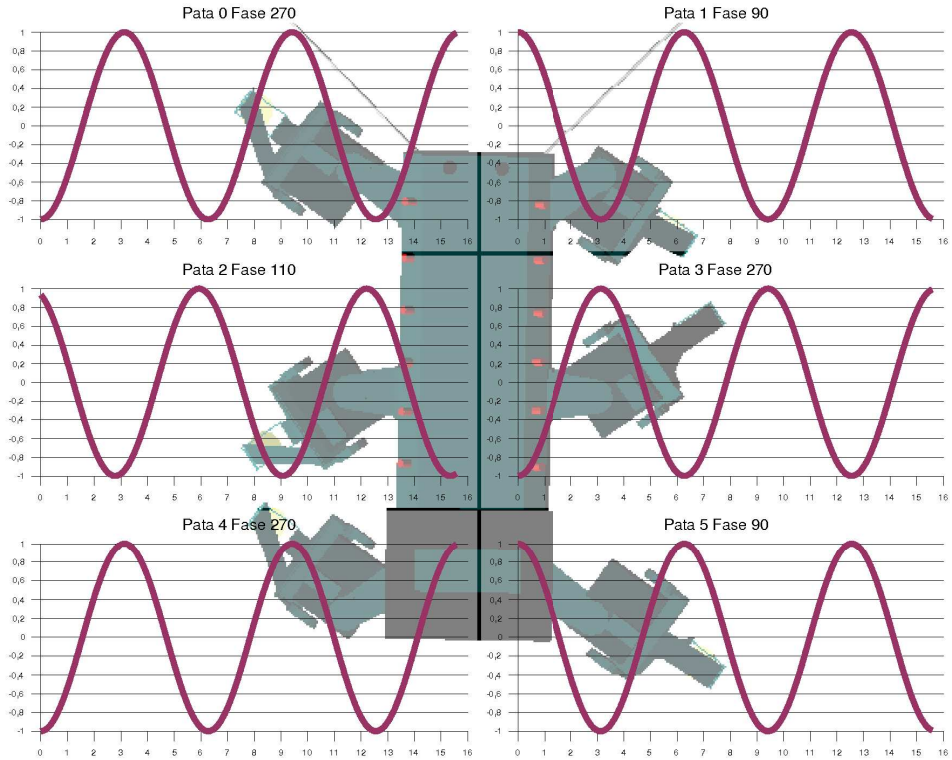


Figura 52. Función senoidal que representa el movimiento de oscilación de cada pata para la estrategia 148: 270-90-110-270-270-90. Las patas del robot empiezan como muestra el esquema.

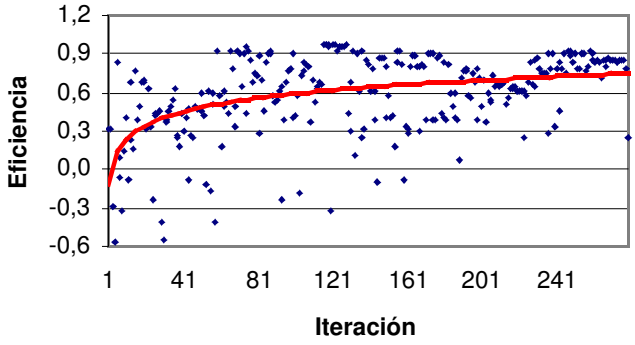


Figura 53. Eficiencia de los gaits aplicados

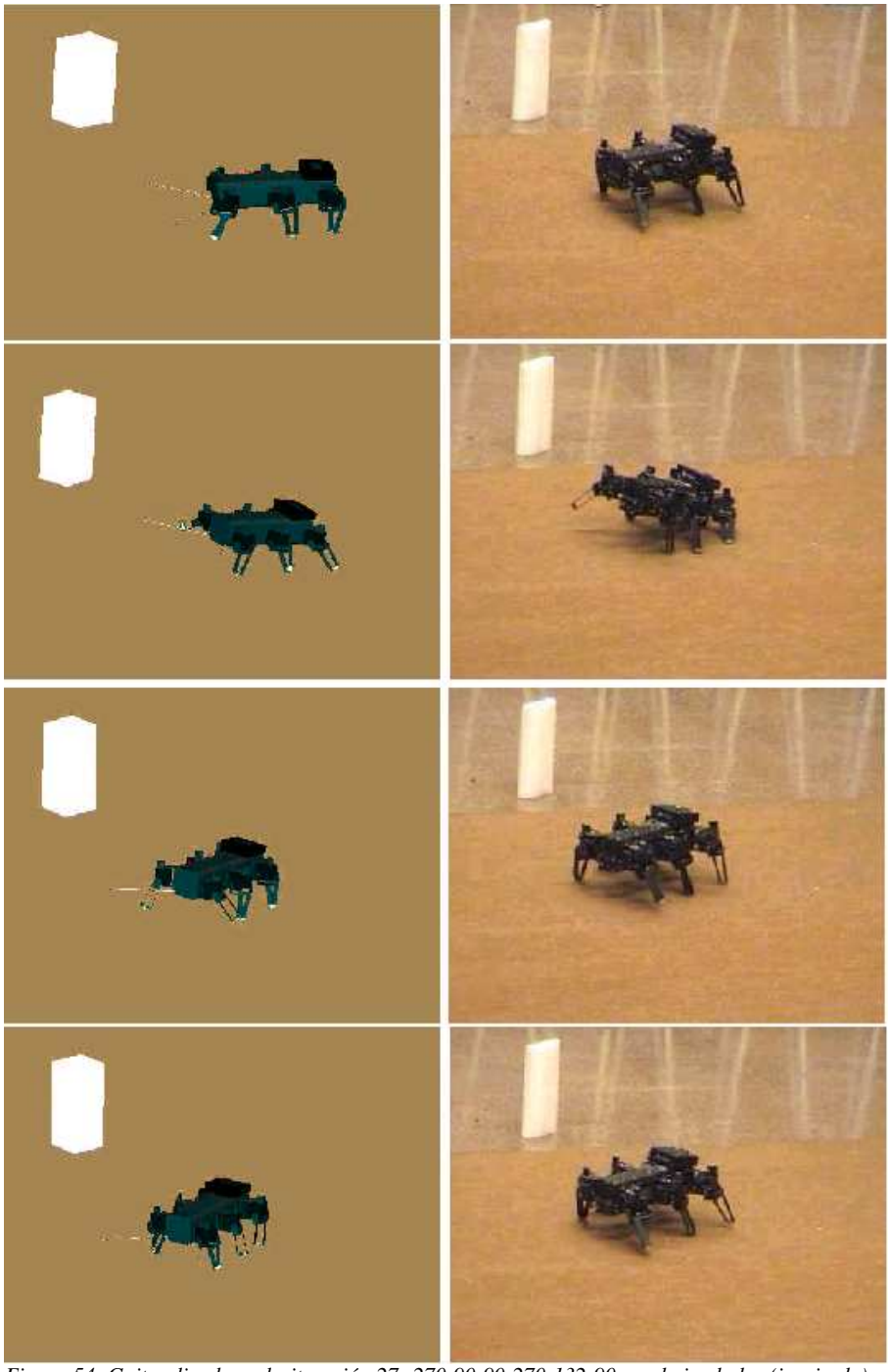


Figura 54. Gait aplicado en la iteración 27: 270-90-90-270-132-90 en el simulador (izquierda) y en el robot real (derecha). Eficiencia = 0.

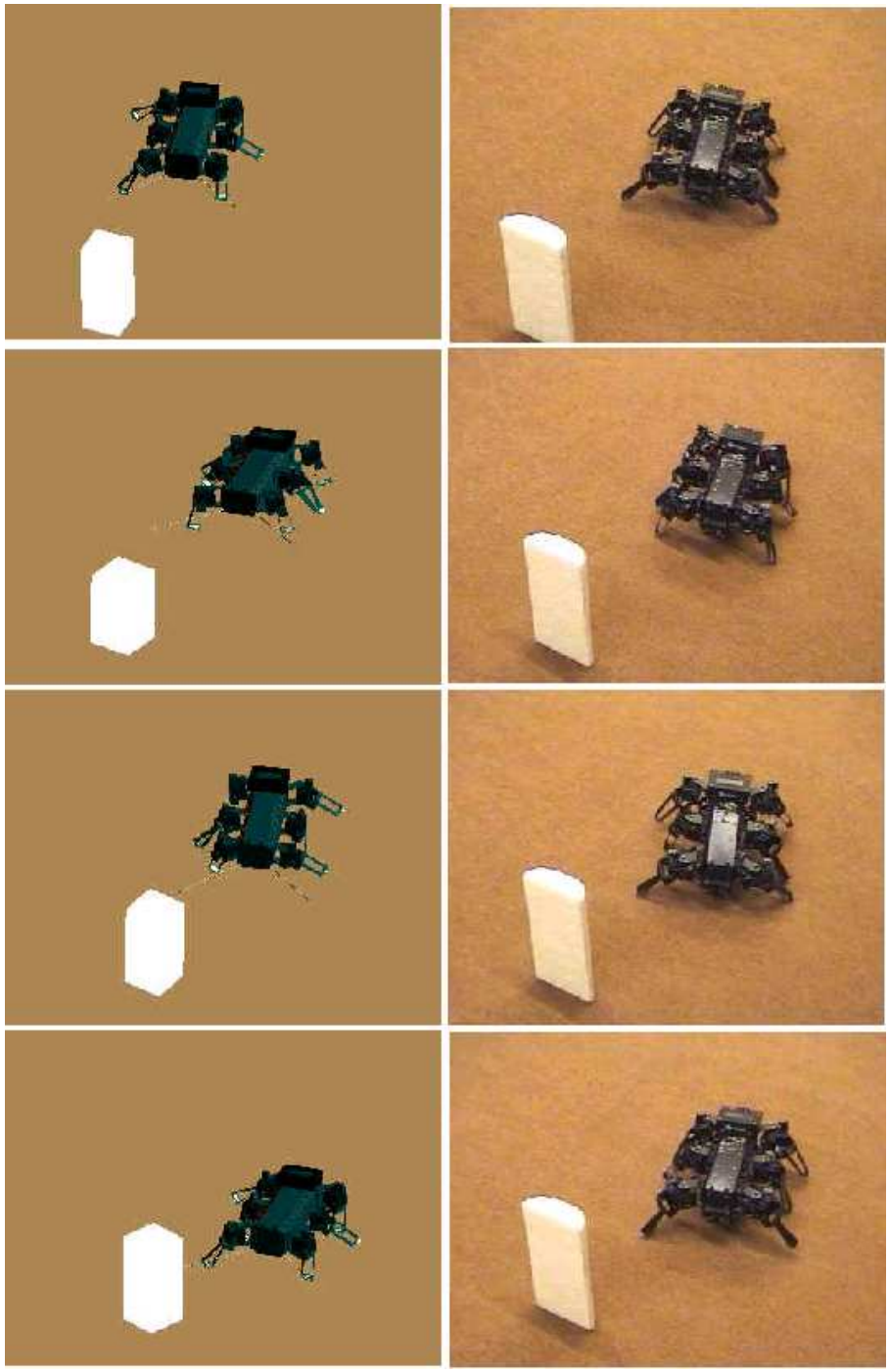


Figura 55. Gait aplicado en la iteración 71: 270-8-90-209-270-90 en el simulador (izquierda) y en el robot real (derecha). Eficiencia = 0.9.

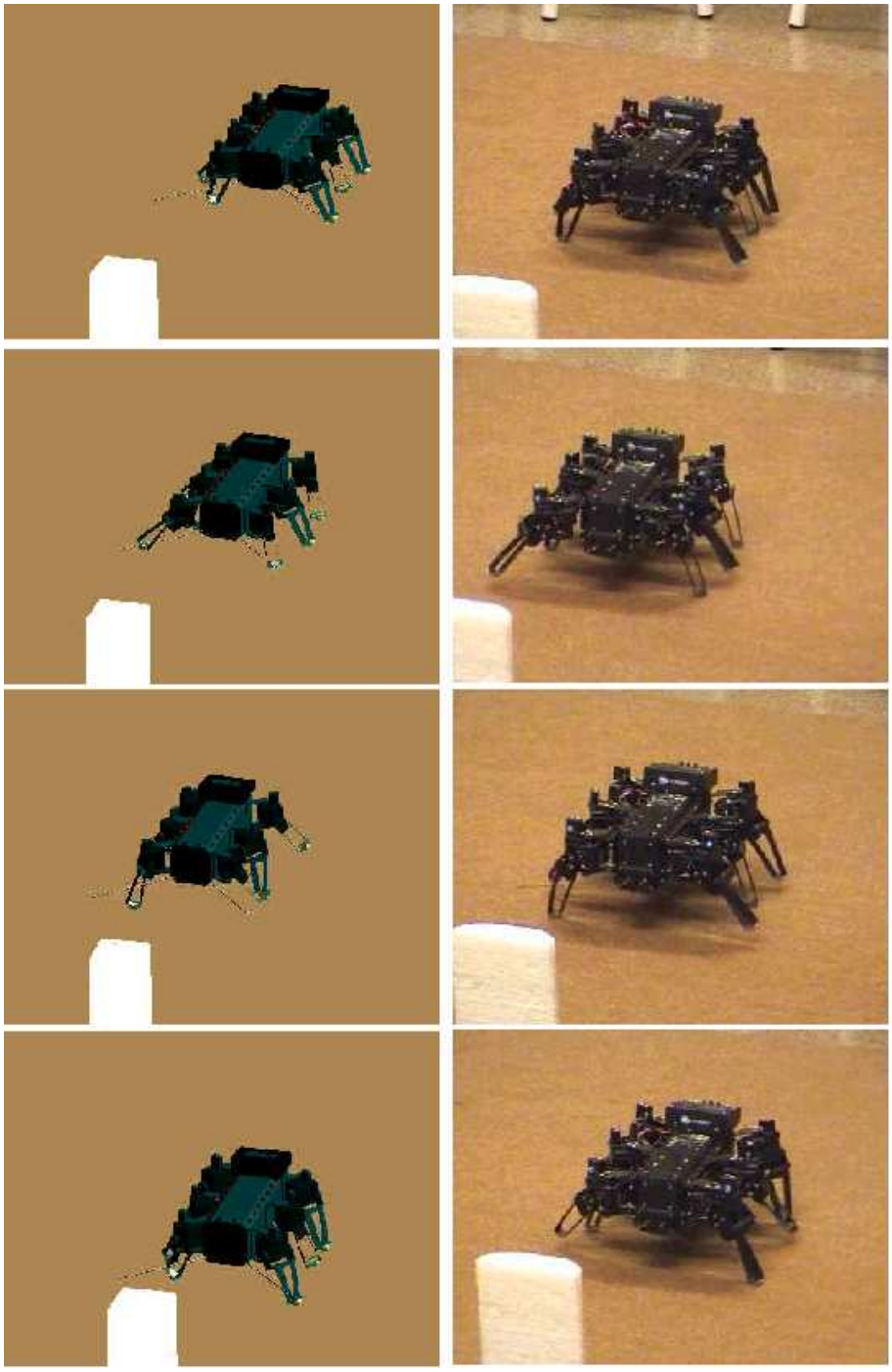


Figura 56. Gait aplicado en la iteración 300: 270-90-90-27-270-90 en el simulador (izquierda) y en el robot real (derecha). Eficiencia = 1.

7.1.4 Aprendiendo a girar

Del apartado anterior hemos obtenido que la forma más adecuada de caminar, en función de los condicionantes del simulador, es un gait trípode. Ahora fijamos esta combinación de fases en el Hermes II y cambiamos el problema: el robot debe escoger una combinación de amplitudes A en el motor de oscilación de cada pata para alcanzar un objeto que situamos en su cercanía. De nuevo usamos estrategias de longitud uno.

Hemos desarrollado un método de enseñanza que consiste en colocar un objeto con forma de bloque en una semicircunferencia enfrente al robot y a una distancia aleatoria entre 50 y 100 cm, como se muestra en la figura 57, dejar que sense dicho objeto y ejecutar una iteración del mecanismo. Pretendemos que no sólo aprenda a girar a un lado o a otro, sino a girar más o menos de forma que termine lo más cerca y de cara al objeto (distancia mínima, ángulo cero). La motivación del comportamiento es, por tanto, la maximización de la sensorización en los dos infrarrojos de las patas delanteras.

La estrategia aplicada llevará al robot más o menos lejos del objeto, de forma que si lo alcanza (distancia al bloque menor de 30 cm) o bien si lo pierde (distancia al bloque mayor de 120 cm) volvemos a colocarlo en su entorno. Hemos utilizado estas distancias límite porque, como explicamos en la primera parte de este ejemplo, los sensores infrarrojos del Hermes tienen dicho rango de alcance, es decir, por debajo de 30 cm devuelven siempre 0 y por encima de 100 cm siempre 20. Este método de enseñanza es

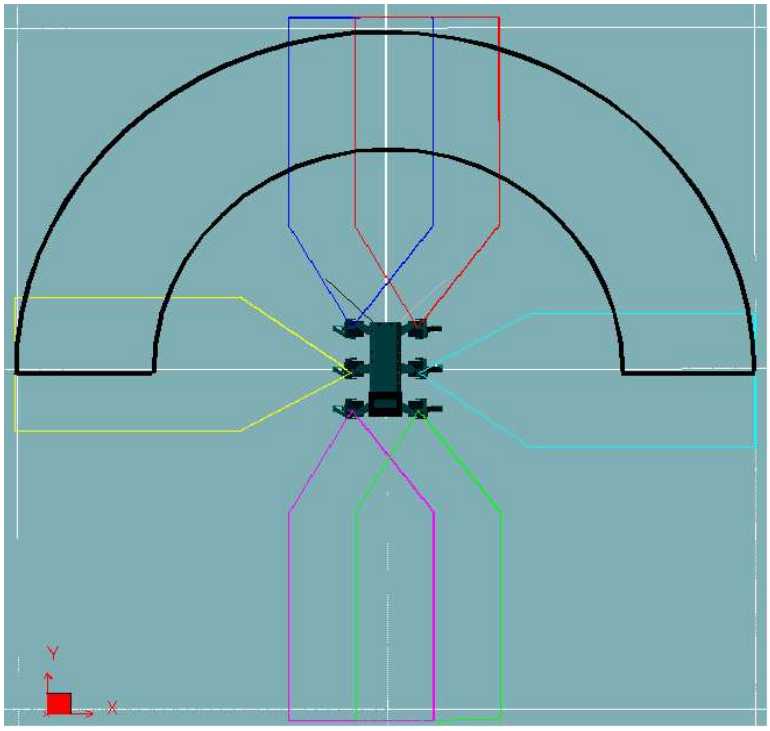
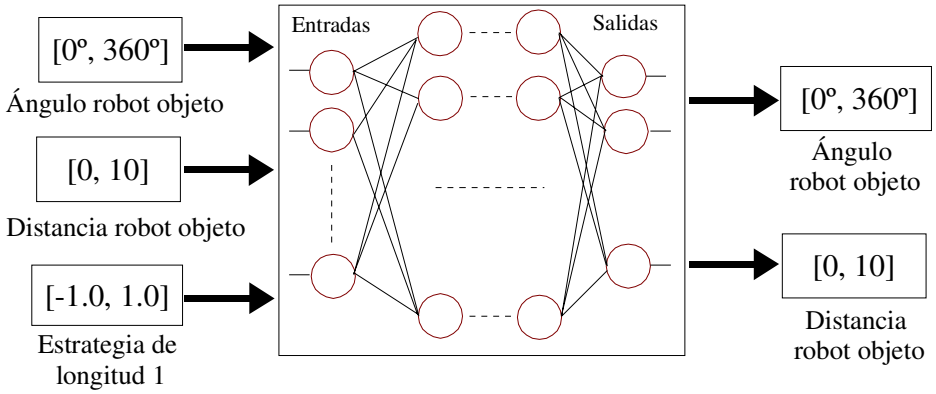


Figura 57. Imagen del Hermes II en el simulador donde la semicircunferencia engloba las posibles posiciones para el bloque que debe ser alcanzado.

idéntico al que utilizaríamos con un animal o un niño, es decir, se muestra el estímulo y se premia la actuación adecuada.

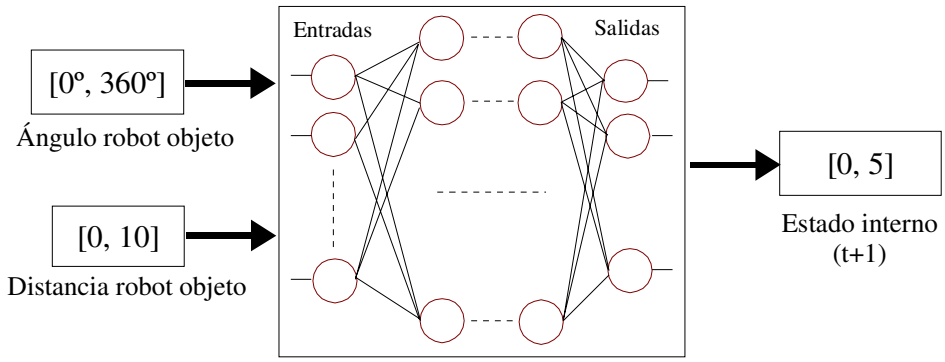
Modelos y estrategias

Los modelos de mundo son redes neuronales con 3 entradas y 2 salidas. Las entradas corresponden a la distancia y ángulo del robot con respecto al bloque (dadas por el sensor virtual tal y como explicamos en la primera parte de este ejemplo y en el Apéndice A) y a la amplitud de giro. Este último valor lo hemos codificado entre -1 y 1 de forma que un 0 se traduce en que el robot se mueve recto con una amplitud base (fijada para que la velocidad de movimiento sea media). Un valor 1 hace que las 3 patas del lado derecho del robot no giren mientras que las de la izquierda giran con amplitud base y, en consecuencia, el robot gira 90º hacia la derecha en el sitio, mientras que un valor de, por ejemplo, -0.5 hace que las 3 patas de la zona izquierda del robot giren la mitad de amplitud que las de la derecha, obteniendo un giro neto a la izquierda de un cierto radio. Las 2 salidas corresponden a las entradas sensoriales en t+1 y son, por tanto, el ángulo y la distancia del robot respecto al bloque. A continuación mostramos los modelos de mundo de forma esquemática:



El modelo interno tiene ahora 2 entradas que corresponden a las predicciones que realiza el modelo de mundo y una única salida que es el estado interno. La motivación del comportamiento del robot consiste en acercarse lo máximo posible al objeto y, además, llegar lo más de frente posible. Teniendo esto en cuenta, hemos codificado el estado interno entre 0 y 5 en función únicamente de la distancia y el ángulo del robot respecto al bloque. Así, un valor 0 corresponde a una distancia grande y a un ángulo distinto de cero, mientras que un valor 5 corresponde a una distancia pequeña y un ángulo de cero grados. Como hemos comentado en varias ocasiones a lo largo del planteamiento teórico del mecanismo y como refleja la codificación utilizada para el estado interno, ésta es más subjetiva que la utilizada en el modelo de mundo. Así, los modelos internos son más flexibles a la hora de ser definidos y, por este motivo, su complejidad puede ser reducida con mayor facilidad.

El modelo interno quedaría, pues, de forma esquemática:



De nuevo hemos aplicado algoritmos genéticos para la obtención de los modelos y estrategias con los siguiente parámetros:

- El cruce se realiza por dos puntos con una probabilidad del 60%.
- La mutación consiste en sumar al gen correspondiente un número aleatorio entre -1 y 1 elevado al cubo, como ya hicimos en el ejemplo anterior. Obtuvimos los mejores resultados con probabilidad del 2% en los modelos y del 10% en las estrategias.
- Conservamos el mejor individuo de una generación a la siguiente.
- Utilizamos selección por torneo con una ventana de tamaño 2 (baja presión selectiva).

Los parámetros propios de la evolución de los **modelos de mundo** que mejor resultado nos han dado son los siguientes:

- Tamaño de la red: 3 neuronas de entrada, dos capas ocultas de 4 neuronas cada una y 2 neuronas de salida. Evolucionamos los pesos de la red (3x4 + 4x4 + 4x2 = 36 genes) además de un bias por cada neurona de las capas intermedias y de salida (4 + 4 + 2 = 10 genes). Por tanto, los cromosomas están formados por 46 genes (números reales).
- Escogemos una población alrededor de un orden de magnitud por encima del número de genes, en este caso de 600 individuos.

En los **modelos internos**, los parámetros particulares que mejor resultado nos han dado son:

- Tamaño de la red que representa el modelo interno: 2 neuronas de entrada, dos capas ocultas de 3 neuronas cada una y 1 neurona de salida. Evolucionamos los pesos de la red (2x3 + 3x3 + 3x1 = 18 genes) además de un bias por cada neurona de las capas intermedias y de salida (3 + 3 + 1 = 7 genes). Por tanto, los cromosomas están formados por 25 genes (números reales).
- Escogemos una población alrededor de un orden de magnitud por encima del número de genes, en este caso de 600 individuos.

Antes de mostrar los parámetros utilizados en la evolución de las **estrategias**, hemos de explicar la metodología seguida al intentar evolucionar cromosomas con un único gen (recordamos que la estrategia está formada por una única acción que varía entre -1 y 1): hemos optado por dividir, a la hora de aplicar los operadores de cruce y mutación, el

valor real de cada gen en 5 partes, de modo que su suma resulte el valor original. Estamos trabajando internamente con cromosomas de 5 genes. El cruce y la mutación se realizan sobre estos genes y, tras estas operaciones, se restaura el valor de la estrategias entre -1 y 1. En la evolución hemos utilizado los siguientes parámetros:

- Como en los ejemplos anteriores, evolucionamos directamente las estrategias por lo que tenemos cromosomas de un sólo gen.
- Utilizamos una población de 20 estrategias.

Por último, en cuanto al funcionamiento global del mecanismo, cada iteración implica 4 generaciones de evolución para los modelos de mundo e interno y 100 generaciones para las estrategias. De hecho, partimos de una población nueva de estrategias en cada iteración. La MCP utiliza una estrategia de reemplazo de tipo FIFO y su tamaño es de 40 muestras.

Resultados

La evolución del error cuadrático medio de la salida que proporciona el modelo de mundo respecto a la real se muestra en la figura 58. Como ocurría en los dos ejemplos anteriores, el error cuadrático medio va decreciendo con oscilaciones hasta que se llega a una zona de estabilidad. Este error es muy pequeño por lo que la predicción de distancia y ángulo que realiza el modelo de mundo es adecuada.

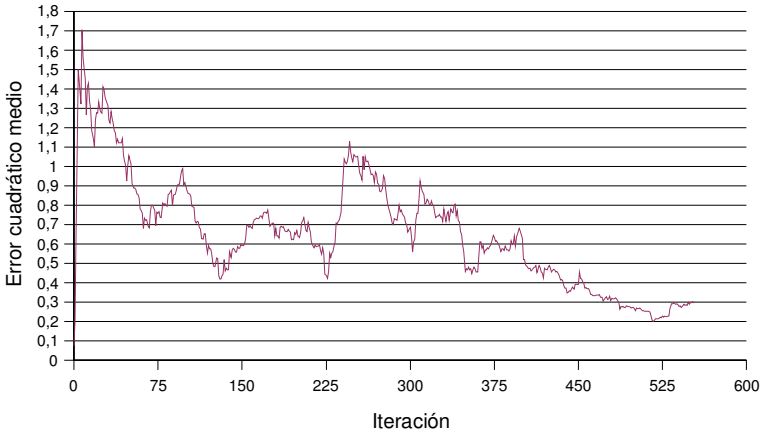


Figura 58. Evolución del error cuadrático medio entre la predicción que realiza el mejor modelo de mundo y los datos reales almacenados en la MCP en cada iteración del MDB

En la figura 59 representamos la evolución del error cuadrático medio de la salida que proporciona el modelo interno respecto a la real. Vemos que es más fácil de aprender que el modelo de mundo, y sólo tarda unas 10 iteraciones en oscilar en torno a un valor de 0.4. Alrededor de la iteración 350 oscila entorno a 0.25 y al final termina oscilando en torno a 0.08, que implica una predicción perfecta del estado interno. Durante la ejecución del mecanismo, guardábamos el número de iteración en la que el robot alcanzaba el bloque (distancia menor de 30 cm). Con estos datos hemos hecho la gráfica 60 que muestra la distancia entre dos aciertos consecutivos. Inicialmente, el robot tardaba entre 20 y 40 iteraciones en alcanzar el bloque dos veces consecutivas. A medida que los modelos van mejorando, las estrategias que se aplican son cada vez

mejores y el robot tarda menos iteraciones en alcanzar el bloque. Como vemos en la figura, hacia la iteración 300 el número de iteraciones necesarias se ha reducido a 2, 3 y esta tendencia ya no la pierde. El número mínimo de iteraciones necesarias para alcanzar el objeto era 2, impuesto por el diseño, por lo que el mecanismo llega a la solución óptima. De hecho, en las primeras etapas del mecanismo, la tendencia del robot era girar de forma aleatoria sin que pareciese importar la posición del objeto. A partir de la iteración 300 podemos afirmar que ya ha aprendido a girar porque, aunque no alcance el objeto en dos iteraciones, lo hace en 3 o 4 con una clara tendencia a corregir su posición en función de la del bloque.

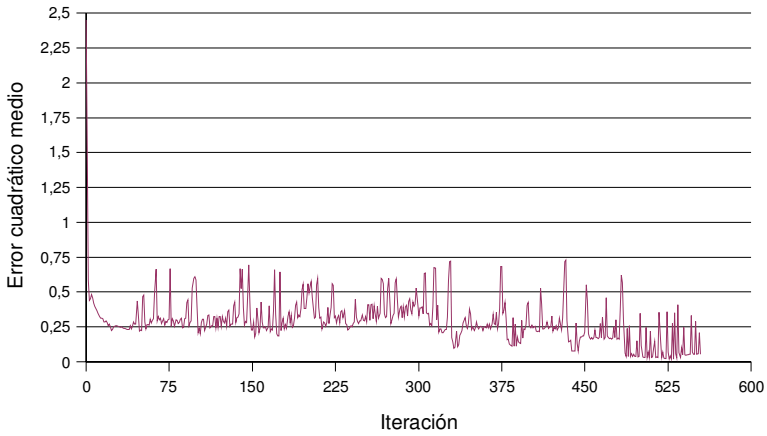


Figura 59. Evolución del error cuadrático medio entre la predicción que realiza el mejor modelo interno y los datos reales almacenados en la MCP en cada iteración del MDB.

Este mismo comportamiento es el que queremos resaltar con las imágenes tomadas del robot real, que se muestran en las figuras 61 y 62. En la figura 61 se muestra la trayectoria seguida por el robot en simulación y en el mundo real tras aplicar las estrategias 53 (estrategia -0.9; recordamos que implicaría que las tres patas de la izquierda prácticamente no girasen), 54(-0.9), 55(-0.9), 56(-0.6), 57(0.9) y 58(-0.4). Las tres primeras imágenes corresponden a la misma posición del bloque, mientras que en las dos últimas lo hemos movido a una nueva posición, como se indica. En las iteraciones 53, 54, 55 y 56 el robot efectúa un giro sobre sí mismo alejándose del objeto por lo que en la iteración 57 se lo acercamos pero, como muestran la dos últimas imágenes, lo vuelve a perder. De nuevo se refleja el hecho de que los modelos de mundo no son adecuados todavía (de acuerdo con la figura 58) y las estrategias aplicadas tampoco.

En la figura 62, las estrategias corresponden a las iteraciones 421(0.98), 422(-0.32), 423(-0.12) y 424(-0.9). Para las dos primeras, el bloque estaba en la misma posición, es decir, el robot necesitó dos movimientos para alcanzar el bloque (dos primeras imágenes de la figura 62). En la iteración 423 el robot continuó prácticamente recto, alcanzó el objeto y lo movimos a una nueva posición (tercera imagen de la figura). La estrategia 424 implica un giro a la izquierda para volver a alcanzar el bloque (dos últimas imágenes de la figura). Como vemos, la similitud entre el resultado del simulador y lo que el Hermes realiza en la realidad es alta. La figura 62 refuerza la

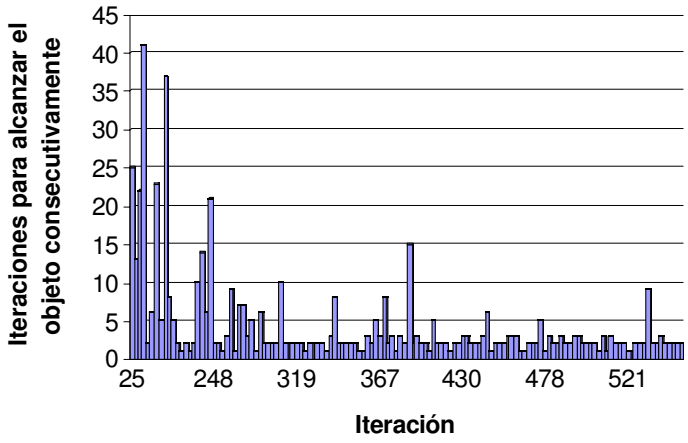


Figura 60. Iteraciones transcurridas entre dos capturas consecutivas del objeto

filosofía del mecanismo y, tras interactuar con su entorno, termina aprendiendo lo que de él necesita a la perfección.

Como conclusión, debemos decir que una vez el gait ha sido escogido con acierto, el hecho de obtener una combinación de amplitudes de giro resulta sencillo. Originalmente, planteamos este ejemplo de otra forma, ya que pretendíamos que el robot aprendiese a la vez las fases y las amplitudes de las patas. El problema de este planteamiento es que era demasiado complicado discriminar si un giro se debía a un gait inadecuado o a una amplitud errónea. La evolución de los modelos oscilaba mucho y no daba señales de convergencia porque las funciones a aproximar eran excesivamente complejas (sobre todo en el caso de los modelos de mundo). Al dividir el problema en dos, fases y amplitudes, hemos obtenido un mecanismo que permite al Hermes real alcanzar un objeto con un modo de caminar muy eficiente. Es necesario destacar que el robot llega a estas soluciones sin que el diseñador imponga restricciones en las posibles estrategias a realizar. En el caso de las amplitudes, la única simplificación que hemos hecho ha consistido en utilizar un actuador virtual, es decir, un único valor en la evolución y selección de las estrategias que se traducía en una amplitud sobre cada pata. Esto no disminuye el rango de posibles amplitudes y simplifica el modelo de mundo. En el caso de las fases, no encontramos el modo de aplicar un actuador virtual sin perder generalidad por lo que el robot podía probar cualquier combinación posible de valores, y el gait obtenido ha sido el trípode.

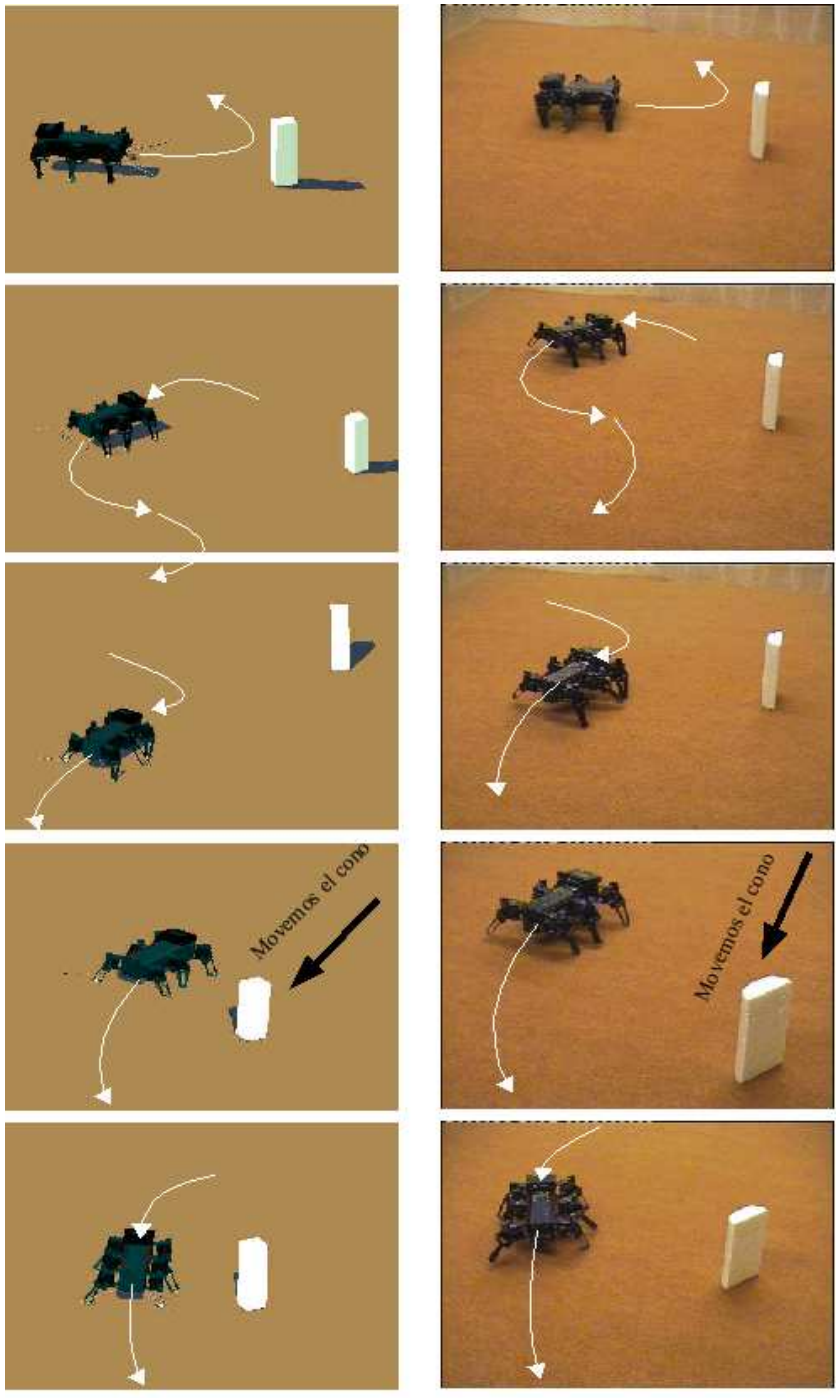


Figura 61. Estrategias aplicadas en las iteraciones 53, 54, 55, 56, 57 y 58. A la izquierda mostramos los resultados del simulador y a la derecha los del robot real.

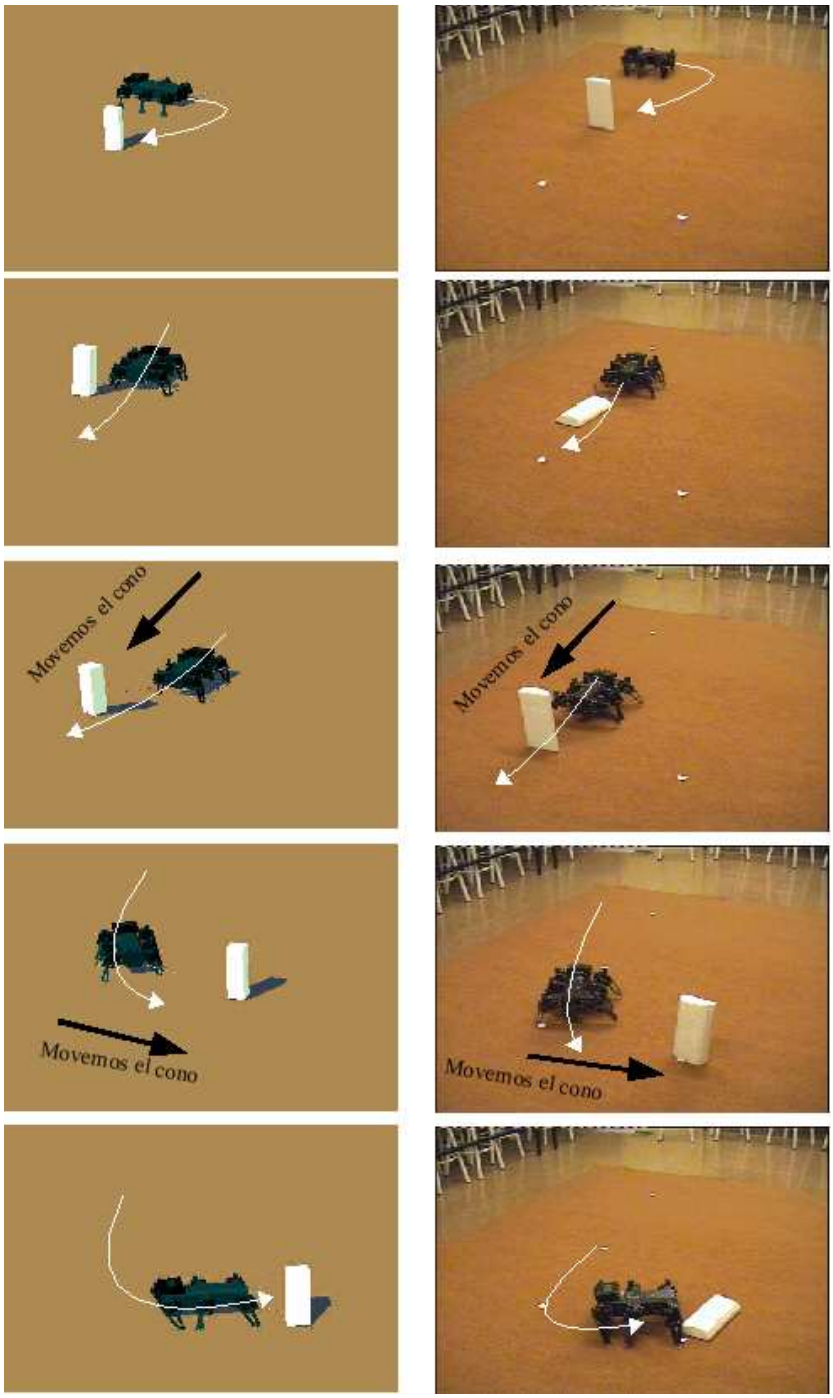


Figura 62. Estrategias aplicadas en las iteraciones 421, 422, 423 y 424. A la izquierda mostramos los resultados del simulador y a la derecha los del robot real.

7.2 Aplicación del MDB en comportamientos de alto nivel

A lo largo del presente trabajo, hemos hecho énfasis en que el MDB ha sido diseñado para ser aplicado a agentes reales en tareas que requieran aprendizaje y adaptabilidad. A continuación abordamos un ejemplo de aplicación de más alto nivel deliberativo que los dos anteriores, donde nos habíamos centrado en la obtención de comportamientos primarios como aprender a caminar o aprender a girar hacia un objetivo.

Para ello, usaremos un robot rodado, el Pioneer 2, en una tarea donde, de nuevo debe alcanzar un objeto siguiendo las órdenes de un profesor. El comportamiento de alto nivel que pretendemos demostrar consiste, desde el punto de vista de un observador externo, en el aprendizaje por parte del robot de las órdenes que le da el profesor de modo que, tras una etapa de enseñanza donde se le dan premios o castigos, el robot sea capaz de continuar con el mismo comportamiento aunque el profesor no esté presente. En este ejemplo, el lenguaje que utiliza el profesor para comunicarse con el robot también debe ser aprendido permitiendo que, en caso de que el profesor cambie dicho lenguaje, el robot posea la capacidad de aprender el nuevo y continuar obedeciendo. Esto facilita enormemente la interacción hombre-máquina. Finalmente, se demostrará cómo el MDB permite aprender comportamientos diversos en función de las instrucciones del profesor por medio de un cambio en la relación órdenes-premios que le llevará a alejarse del objeto.

Antes de entrar con el ejemplo en sí, haremos una breve presentación del robot Pioneer 2 resaltando las razones que nos han llevado a su selección.

7.2.1 Robot Pioneer 2

Como plataforma de aplicación de este ejemplo, utilizaremos el robot rodado Pioneer 2-DX que mostramos en la figura 63 y que se caracteriza por una gran capacidad de carga (de hasta 20 kg) y por la fiabilidad de sus sensores. En este sentido, el Pioneer 2 posee un anillo de 16 sensores de ultrasonido (sónar) dispuestos de forma análoga a la mostrada en la figura 64 para los 8 sensores frontales. Los actuadores son dos motores conectados a sendas ruedas existiendo una tercera rueda de apoyo, más pequeña y libre, situada en la parte trasera. En la figura 65 vemos un esquema de las partes más importantes del Pioneer 2.

Figura 63. Robot Pioneer 2-DX que utilizaremos en este ejemplo.

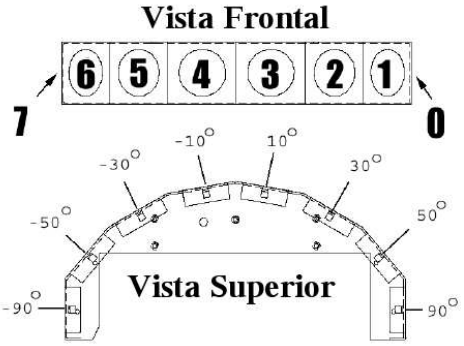


Figura 64. Vista esquemática del anillo frontal de 8 sónares que posee el Pioneer 2.

El sistema operativo se ejecuta en el microcontrolador del robot, un Siemens 88C166 de 20 Mhz, que realiza únicamente las tareas más básicas, como recibir los datos de los sensores y proporcionarlos a las aplicaciones que los necesiten y viceversa, ejecutando sobre los actuadores las órdenes que le llegan. Tanto la sensorización como la actuación son procesadas por parte del sistema operativo. En el primer caso, éste devuelve la distancia a la que se encuentra el objeto detectado y en el segundo requiere como entradas las velocidades lineal y angular. Como hemos dicho, estos datos no coinciden con la codificación interna que realmente utiliza el robot.

La filosofía de funcionamiento del Pioneer 2 lo hace mucho más potente que el Hermes II utilizado en el apartado anterior, ya que el microcontrolador se encarga únicamente de aplicar los comandos a los actuadores y de proporcionar información a una computadora más potente que realiza las tareas de cálculo de los controladores o acciones que serán aplicadas (en nuestro caso, la ejecución del MDB). En este ejemplo, hemos utilizado un ordenador portátil que situamos en la plataforma superior conectado al robot mediante un puerto serie, como se muestra en el figura 66.

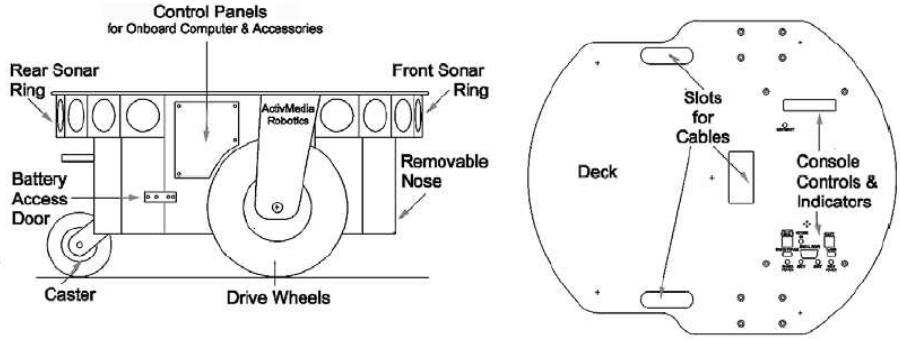


Figura 65. Esquema de las partes más importantes del robot Pioneer 2.

Además de las características básicas que acabamos de comentar, de cara a poder llevar a cabo comportamientos más complejos, hemos añadido nuevos sensores externos

Figura 66. Robot Pioneer 2 llevando el ordenador portátil donde se ejecuta el MDB.

al robot asociados al ordenador portátil donde se ejecuta el MDB: un micrófono, un ratón y un teclado. Cualquiera de ellos facilita la comunicación de forma directa con el robot. También hemos añadido un nuevo actuador, una pinza o gripper, que mostramos en la figura 67 ya montada en el robot y que nos permite marcar cuando el Pioneer logra su objetivo al alcanzar el objeto.

Al igual que en los ejemplos con el Hermes II, las primeras pruebas de ejecución del MDB se llevaron a cabo en un entorno simulado que utiliza un modelo muy realista del Pioneer 2. Dicho entorno ha sido bautizado como SEVEN (Simulation and EVolution Environment) y en la figura 68 vemos una imagen del mismo. Los detalles sobre la implementación del simulador y su grado de fiabilidad pueden ser consultados en [Becerra, 03]. SEVEN se compone, básicamente, de dos módulos: un evolucionador de comportamientos y un simulador. El simulador dispone de un interfaz gráfico que puede

Figura 67. Imagen del robot Pioneer 2 con la pinza (gripper) que utilizaremos en el ejemplo.

ser activado o desactivado y permite diseñar entornos que posteriormente serán usados en la evolución o en el testeo.

El punto de unión entre el mecanismo MDB y el simulador de SEVEN se encuentra en la fase de actuación, es decir, cada vez que una estrategia seleccionada debe ser aplicada al entorno para obtener nuevos valores de sensorización.

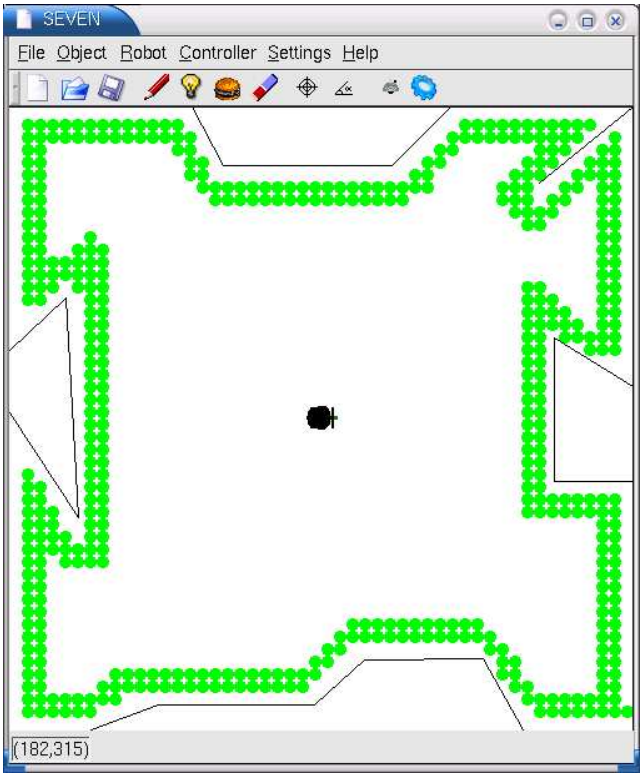


Figura 68. Imagen del simulador para el Pioneer 2 de la herramienta SEVEN.

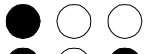
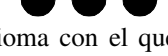
Como vemos, las diferencias con el robot Hermes II utilizado anteriormente son notables, tanto a nivel de capacidad sensorial como computacional. El hecho de que la comunicación con el robot sea directa sin la necesidad de cables, nos proporciona una gran libertad de movimiento y de reacción antes posibles fallos. Aún así, cada uno de los comportamientos ha sido obtenido previamente en el simulador, por lo que mostraremos tanto los resultados en simulación como los reales.

7.2.2 Modelo interno inducido

Pavlov (1927), médico ruso, observó casualmente que a los perros que tenía en su laboratorio, les bastaba oír los pasos de la persona que les traía la comida para comenzar a salivar y a segregar jugos gástricos, es decir, parecía que los perros habían aprendido a anticipar la comida. Pavlov comenzó a estudiar este intrigante fenómeno y se preguntó si cualquier otro estímulo, por ejemplo el sonido de una campana, podía provocar la salivación si se unía a la presentación de la comida.

Bien, pues estos experimentos darían lugar a una teoría conocida como *Condicionamiento Clásico* que posteriormente aportaría parte del fundamento teórico del *Conductismo*, escuela psicológica que pretende explicar y predecir la conducta y que aporta luz sobre muchos de los aprendizajes no sólo en los perros, también en los humanos.

Gran parte de nuestras conductas son aprendidas y en numerosas ocasiones se aprenden por condicionamiento, es decir, descubrimos que lo que hacemos tiene consecuencias positivas o negativas y somos capaces de anticipar dichas consecuencias y, de acuerdo con ello, modelar nuestra forma de comportarnos. Este tipo de aprendizaje también se denomina *aprendizaje inducido* y es el que pretendemos lograr en este ejemplo para los modelos internos del MDB. Para ello, hemos planteado un experimento que se puede resumir en los siguientes puntos básicos:

- El robot Pioneer 2 está situado en un entorno real simple, sin obstáculos, donde el único objeto existente es un cilindro de 24 cm de diámetro y 50 cm de alto (podríamos utlizar cualquier otro objeto detectable por sus sónares).
- Situamos al robot en el punto central del entorno y el objeto en una zona entre dos circunferencias de radios 1.8 m y 2.4 m respecto a dicho objeto. En la figura 69 mostramos un esquema de este montaje en el simulador SEVEN. La posición y la orientación inicial del robot es siempre la misma, en cambio el objeto puede estar en cualquier punto de la zona señalada en color oscuro.
- Realizaremos un proceso de enseñanza supervisada, de modo que un profesor externo al MDB guiará al robot de frente hacia el objeto. Es fundamental para este experimento que el profesor sea coherente en sus indicaciones, de modo que no engañe al robot.
- Las órdenes posibles han sido reducidas a siete, con objeto de centrar el ejemplo en la tarea cognitiva y no en la precisión a la hora de alcanzar el objeto. Es decir, con un conjunto reducido de órdenes, serán necesarias varias acciones antes de lograr el objetivo al estar limitados los movimientos posibles. A continuación mostramos, a título indicativo, una posible codificación discreta para las órdenes utilizadas, que podrá cambiar dependiendo del profesor:
  - SIN GIRO : 
  - GIRO PEQUEÑO A LA DERECHA: 
  - GIRO MEDIO A LA DERECHA: 
  - GIRO GRANDE A LA DERECHA: 
  - GIRO PEQUEÑO A LA IZQUIERDA: 
  - GIRO MEDIO A LA IZQUIERDA: 
  - GIRO GRANDE A LA IZQUIERDA: 
- Estas órdenes representan el lenguaje o idioma con el que el profesor se dirige al robot. Es, como vemos, un *lenguaje basado en símbolos* (no gramatical) que relaciona pulsos (sonidos, pulsaciones mecánicas, etc) con acciones de forma directa. Así, en la aplicación real podemos utilizar algún tipo de fuente sonora, el teclado o el ratón del ordenador portátil con los que se codifican dichas órdenes y que el robot percibirá.

- Tras recibir una orden, el robot actúa y realiza una de las siguientes 7 acciones:
  - SIN GIRO
  - GIRO PEQUEÑO A LA DERECHA
  - GIRO MEDIO A LA DERECHA
  - GIRO GRANDE A LA DERECHA
  - GIRO PEQUEÑO A LA IZQUIERDA
  - GIRO MEDIO A LA IZQUIERDA
  - GIRO GRANDE A LA IZQUIERDA

  Es necesario que todas las órdenes posean una acción asociada para evitar ambigüedades en el proceso de enseñanza.
- *En función del grado de cumplimiento de la orden, el profesor debe premiar o castigar al robot*. Para ello, como señal de recompensa o castigo, utilizamos un valor numérico introducido directamente con el teclado del ordenador portátil, de tal forma que existe un rango continuo entre el castigo y el premio que va desde 0 hasta 5. Además, en caso de obediencia, cuanto mayor sea la distancia recorrida el premio podrá aumentar hasta 8 y cuanto más de frente se encuentre el robot respecto al objeto (menor ángulo), mayor es el premio con un valor máximo de 11.

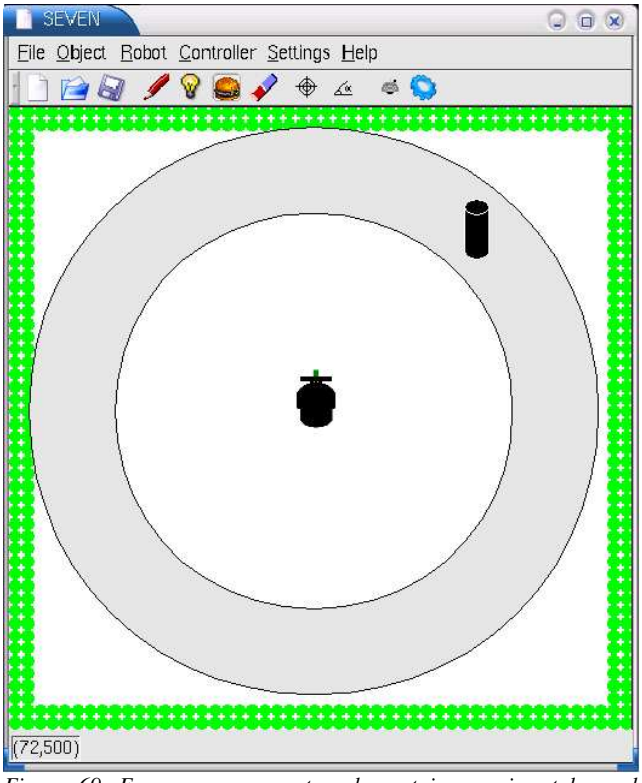


Figura 69. Esquema que muestra el montaje experimental en el simulador.

- Cada vez que el robot alcanza el objeto con la pinza nos indica que se ha logrado la meta y, en este punto, devolvemos el robot a la posición inicial (centro del entorno) y colocamos de nuevo el objeto en su alrededor.

Estas son las características más relevantes del montaje experimental. En cuanto al funcionamiento del mecanismo, hemos establecido que la motivación que guía el comportamiento consiste en *obtener el mayor premio posible* y, en consecuencia, el menor castigo. No se ha concretado más el tipo de premio o castigo, aunque podemos asociar directamente este premio, por ejemplo, a comida de tal forma que la motivación consista en comer lo máximo posible. La motivación está implícita en el agente y se mantendrá durante todo el ejemplo.

Para lograr un aprendizaje inducido en los modelos internos, hemos realizado una implementación particular de los modelos y su interrelación. Para empezar, los modelos de mundo poseen 4 entradas: distancia al objeto, ángulo al objeto, bit de continuidad y acción. La distancia y el ángulo se obtienen directamente de los sensores sónar del Pioneer 2 y el bit de continuidad simplemente indica si dicho ángulo está en un lado u otro del cuerpo del robot. Tal y como explicamos en el ejemplo del apartado 6.2.3, este bit es necesario para evitar ambigüedades en el ángulo, dada la simetría del robot. Las acciones posibles han sido discretizadas a siete valores tal y como se comentó anteriormente. Las salidas de los modelos de mundo son la distancia al objeto, el ángulo al objeto y el bit de continuidad predichos tras la aplicación de la acción.

La novedad de este ejemplo reside en que utilizamos otro modelo de mundo separado del anterior, que se encarga de las sensorizaciones relacionadas con la comunicación profesor-robot, y que denominaremos *modelo de comunicación*. Posee 2 entradas: orden y acción, y proporciona una salida: el premio o castigo predicho, es decir, su salida está directamente relacionada con la motivación. Como vemos, es un modelo de mundo según nuestra definición del apartado 5.2, porque relaciona entradas sensoriales en el instante de tiempo t y la acción aplicada en dicho instante, con entradas sensoriales en t+1. El propio diseño del experimento ha motivado la separación en dos modelos de mundo ya que, como veremos a continuación, al cesar las órdenes este modelo de comunicación no es necesario, pero en cambio el modelo de mundo sí. Además, las entradas sensoriales que afectan al modelo de comunicación son de naturaleza muy distinta a las del modelo de mundo ya que dependen del profesor, es decir, están asociadas a la interacción hombre-máquina.

Por último, el modelo interno que será aprendido por inducción posee dos entradas: distancia relativa recorrida y ángulo tras la aplicación de la acción, y una salida: el estado interno. Como hemos dicho, el estado interno debe reflejar el grado de satisfacción del agente, por lo que coincide con los premios y castigos que proporciona el profesor.

El proceso de ejecución del experimento es el siguiente:

1. El profesor comienza a dar órdenes al robot mediante el lenguaje de símbolos que hemos creado, guiándolo siempre hacia el objeto. Cada orden que proporciona el profesor, es seguida de una acción que lleva a cabo el Pioneer 2. El tiempo de ejecución de cada acción ha sido limitado de modo que, en caso de desobediencia, exista un margen de corrección por parte del profesor. Si la acción coincide con la orden, tras finalizar su aplicación, el robot recibe un premio. En cambio, si ha desobedecido, recibe un castigo.

2. Si el robot se sale de los límites del entorno o bien si alcanza el objeto, lo devolvemos a la posición inicial.
3. La interacción con el profesor y con el entorno proporciona los pares acción-percepción necesarios para el aprendizaje de todos los modelos. En esta etapa, los modelos de mundo, de comunicación e interno evolucionan, aunque la acción se escoge utilizando únicamente el modelo de comunicación, porque buscamos que el robot nos obedezca sin que deba realizar ninguna predicción de otro tipo. Tenemos por tanto, dos procesos evolutivos en segundo plano, ya que los modelos de mundo e interno son aprendidos de acuerdo con el funcionamiento normal del MDB.
4. Llegado un cierto instante, el profesor deja de dar órdenes al robot y éste detecta la nueva situación. A partir de ese instante, el Pioneer 2 utiliza en la selección de la acción el modelo interno que ha ido aprendiendo en la etapa de interacción con el profesor. Debemos aclarar que no existen dos modos de funcionamiento diferentes que deban ser activados en función de la presencia o ausencia de orden, sino que la operación del MDB es la típica con la salvedad de que si se detectan órdenes, no se utilizan los modelos de mundo e interno para seleccionar la acción, sino el de comunicación. En esta etapa, al no haber órdenes, no existe evolución en los modelos internos ni en los modelos de comunicación, pero sí en los modelos de mundo.
5. El modelo interno ha sido aprendido a base de observar las consecuencias en premios y castigos que tienen los aumentos y disminuciones en la distancia tras aplicar una acción. Por este motivo, es fundamental que el profesor sea coherente en su

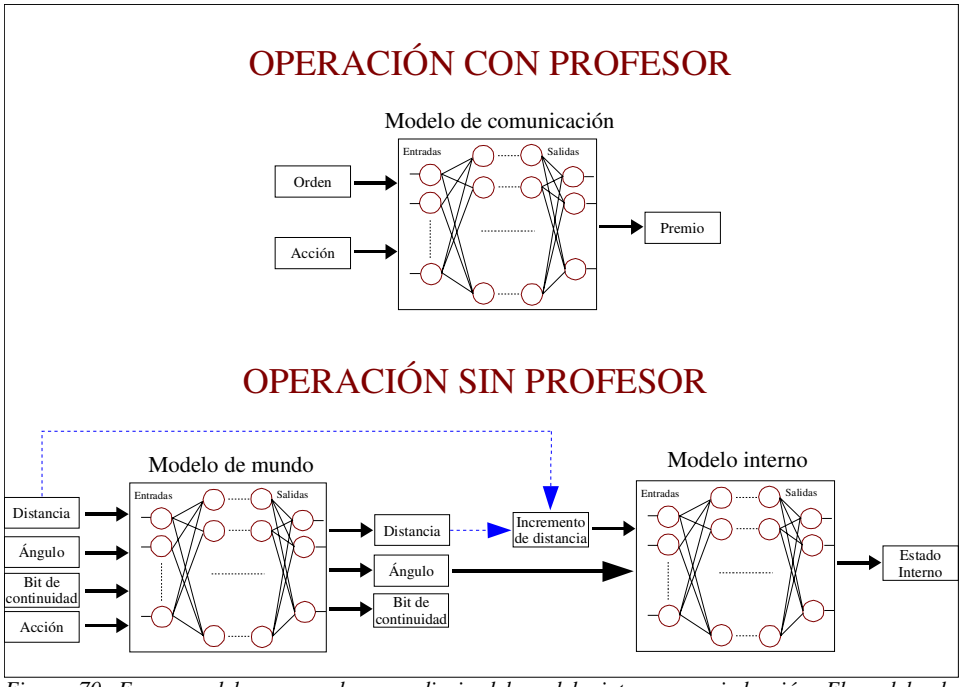


Figura 70. Esquema del proceso de aprendizaje del modelo interno por inducción. El modelo de comunicación se utiliza en presencia del profesor, mientras que en ausencia se utilizan los modelos de mundo e interno inducido de acuerdo con la operación usual del MDB.

enseñanza, dado que los aumentos de distancia deben estar asociados a obediencia y las disminuciones a desobediencia.

En la figura 70 representamos un esquema de los modelos utilizados y de su interrelación en el proceso de aplicación de las acciones. Así, mientras el profesor está presente, las acciones son seleccionadas mediante el modelo de comunicación actual buscando aquella que maximice el premio. Cuando el profesor desaparece, la selección de acciones se realiza aplicando el modelo de mundo actual y el modelo interno actual, como en la operación usual del MDB. Esto en cuanto al proceso de optimización de la acción, porque el aprendizaje de los modelos se produce de manera distinta en unos y otros. Así, los modelos de mundo evolucionan siempre, tanto si hay profesor como si no, debido a que los datos sensoriales que manejan siempre están presentes. En cambio, los modelos de comunicación e internos sólo evolucionan cuando existen órdenes.

Se observa en dicha figura 70 que una de las entradas al modelo interno es la distancia recorrida tras la aplicación de la acción, es decir, la distancia en t+1 menos la distancia en t. Ambos datos son siempre conocidos y permiten que el modelo interno se guíe por incrementos positivos o negativos en la distancia.

Resultados

El primer resultado que comentaremos se basa en la figura 71, donde hemos representado el número de capturas del objeto que se producen a lo largo de las iteraciones, teniendo en cuenta que hasta la iteración 500 existe profesor y desde la 500 hasta la 1000, no. Por tanto, en torno a la iteración 500 se produce la transición que marca si el aprendizaje ha tenido lugar. Como vemos en la figura, el robot sigue capturando al objeto del mismo modo que le había enseñado el profesor, por lo que el aprendizaje por inducción del modelo interno ha resultado satisfactorio. El cambio de pendiente que se produce, refleja que el tiempo entre capturas aumenta, de modo que el modelo inducido no es tan eficiente como el profesor debido, fundamentalmente, a que no se basa en la obediencia, sino en aumentos de la distancia. Al estar trabajando con un robot real en un entorno real, estos aumentos no son siempre iguales y, como comentaremos más adelante, existe una cierta ambigüedad en los modelos de mundo.

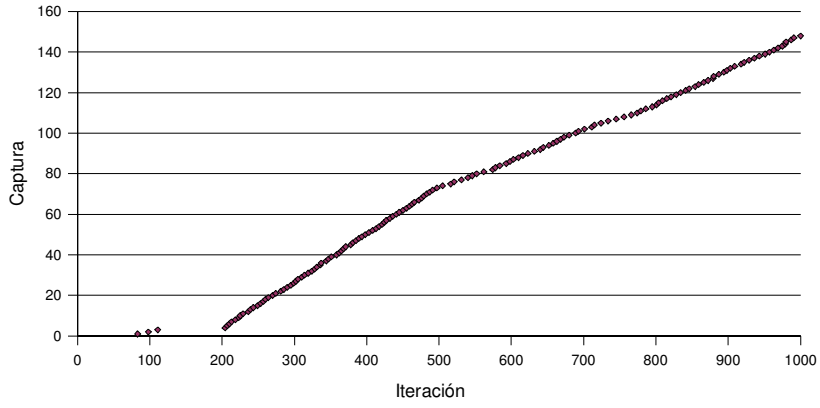


Figura 71. Capturas del objeto por parte del Pioneer 2 frente a la iteración en la que tienen lugar. Desde la iteración 500 hasta la 1000 cesan las órdenes por parte del profesor y se utiliza el modelo interno inducido.

De cualquier forma, la conclusión más importante de este resultado es que el robot continúa con la tarea tras la ausencia del profesor.

En todo este ejemplo se ha utilizado el algoritmo PBGA en el aprendizaje de los modelos y la estrategia de reemplazo para la MCP presentada en el apartado 6.2.2 (sin término de antigüedad). No hemos utilizado MLP por no existir cambio en ninguno de los modelos, algo que sí haremos más adelante. Los parámetros de cada uno de los procesos evolutivos se resumen en la siguiente tabla:

PARÁMETROS DEL PBGA			
	MODELOS DE MUNDO	**MODELOS DE COMUNICACIÓN**	**MODELOS INTERNOS**
Iteraciones	1000	1000	1000
Generaciones por iteración	4	4	4
Tamaño de la población	1400	600	600
Probabilidad de cruce paramétrico	70%	70%	70%
Probabilidad de cruce por capa	50%	50%	50%
Probabilidad de mutación estructural	2%	2%	2%
Probabilidad de mutación paramétrica	10%	10%	10%
Tamaño máximo de la red	4-6-6-3	2-4-4-1	2-4-4-1
Tamaño de la MCP	60	60	60

En las 200 primeras iteraciones que se muestran en la figura 71, no hay prácticamente capturas. Esto es debido a que, previa a la etapa de enseñanza, hemos utilizado una etapa de exploración aleatoria del mundo de cara a que las muestras que almacena la MCP de los modelos de mundo sean lo más heterogéneas posible. En este sentido, debemos resaltar la complejidad del modelo de mundo con el que estamos trabajando debido, por un lado, a la dimensión de la función utilizada, con 4 entradas y 3 salidas y, sobre todo, por la especial disposición de los sensores de sónar en el Pioneer 2. Cada sensor barre un ángulo de 30º con precisión hasta una distancia de unos 3 m, pero existen lagunas de percepción cuando el objeto es estrecho y está cerca del robot, ya que cae en una zona que no es cubierta por ningún sensor. Este hecho puede parecer poco probable, pero nos ocurría con demasiada frecuencia. Para solucionarlo, hemos utilizado un objeto ancho, de 24 cm de diámetro, con lo que en muchas ocasiones es detectado por más de un sónar a la vez. Esta detección múltiple trae consigo que el modelo a aprender sea ambiguo, y tengamos que trabajar con rangos de sensorización y

no con sensorizaciones puntuales unívocas. Este tipo de problemas es algo común en robots reales (se suele hablar de ruido en los sensores) e implica que el modelo de mundo requiere una red mayor que los otros dos modelos.

De cualquier forma, el PBGA nos proporciona tres redes independientes una para cada salida, con un error cuadrático medio tras 1000 iteraciones de un 1% en la distancia, un 1.5% en el ángulo y un 2% en el bit de continuidad, precisión más que suficiente para llevar a cabo la tarea. Debemos recalcar, por tanto, que el PBGA cumple con el objetivo para el que fue diseñado y permite modelar funciones complejas de forma automática reduciendo al mínimo el número de parámetros que debe prefijar el diseñador.

	ORDEN	ACCIÓN	PREMIO
A	○ ○ ●	SIN GIRO	9,3
B	○ ○ ●	GIRO GRANDE IZDA	2,5
C	○ ● ●	GIRO GRANDE DCHA	4,2
D	○ ○ ●	SIN GIRO	8,8
E	○ ● ○	SIN GIRO	4,2
F	○ ● ○	GIRO PEQ DCHA	8,6

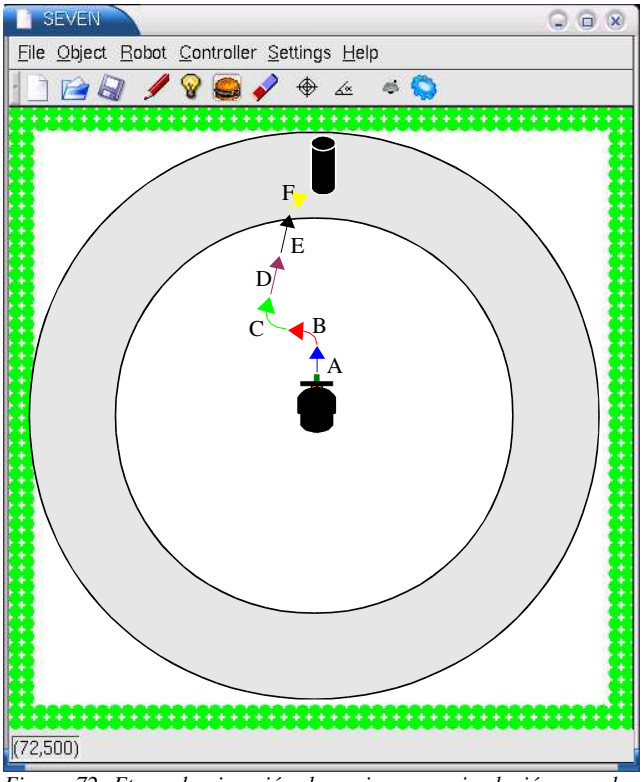


Figura 72. Etapa de ejecución de acciones en simulación para las iteraciones 199-204 donde la selección de acciones se basa en las órdenes del profesor.

La evolución de los modelos de comunicación es muy rápida dada su simplicidad y el PBGA consigue un error del 2% para el mejor modelo. En cuanto a los modelos internos, éstos son más complejos que los anteriores, pero aún así son aproximados con un error del 3% que, como concluimos de la figura 71, es adecuado para llevar a cabo la tarea de forma satisfactoria.

En la figura 72 vemos el resultado en simulación de una etapa de aprendizaje, donde el profesor da órdenes al robot y éste utiliza el modelo de comunicación para seleccionar la acción aplicada. Las acciones mostradas corresponden a las iteraciones 199 a 204, que son las primeras tras la etapa de exploración del entorno, y las únicas donde el modelo de comunicación todavía falla y podemos ver premios y castigos.

La tabla adjunta a la figura indica los valores de orden, acción y premio concretos para las acciones que se muestran en el simulador. De acuerdo con los rangos numéricos establecidos, todas las acciones que obedezcan al profesor proporcionan un valor

	ORDEN	ACCIÓN	PREMIO
A	○ ● ●	GIRO MEDIO DCHA	8,2
B	○ ● ○	GIRO PEQ DCHA	8,6
C	○ ○ ●	SIN GIRO	9,3
D	○ ○ ●	SIN GIRO	9,3

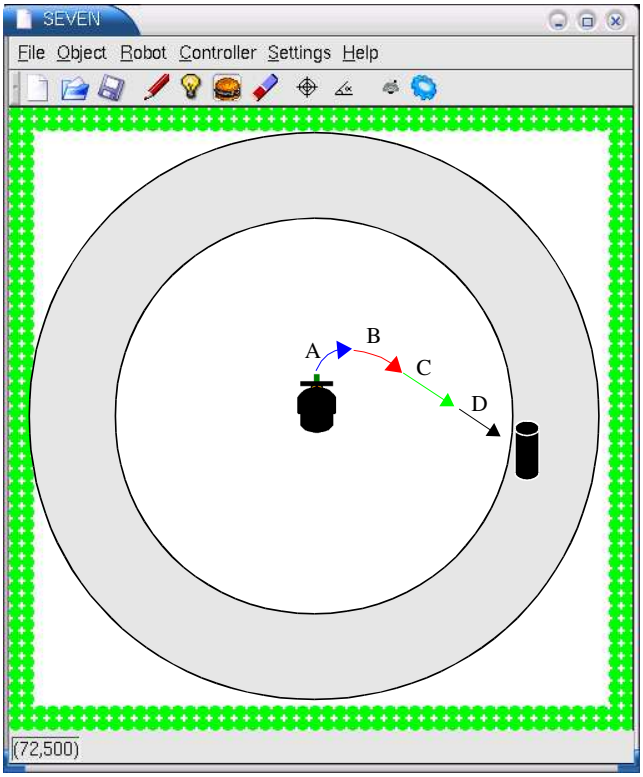


Figura 73. Etapa de ejecución de acciones en simulación para las iteraciones 414-417 donde la selección de acciones se basa en las órdenes del profesor.

mínimo de premio de 5, aumentando hasta 11 en función de la distancia relativa y del ángulo final que se obtengan. Por este motivo, los premios obtenidos en las etapas marcadas con B,C y E (desobediencia) tienen un valor inferior a 5.

De igual forma, en la figura 73 mostramos el resultado en simulación de otra etapa de aprendizaje para el mismo experimento pero en las iteraciones 414 a 417, donde el modelo de comunicación es muy preciso y el robot sigue sin fallo las órdenes del profesor, por lo cual siempre recibe premios. Estas figuras simplemente reflejan visualmente los resultados que se intuían en la figura 71, la más relevante de este apartado.

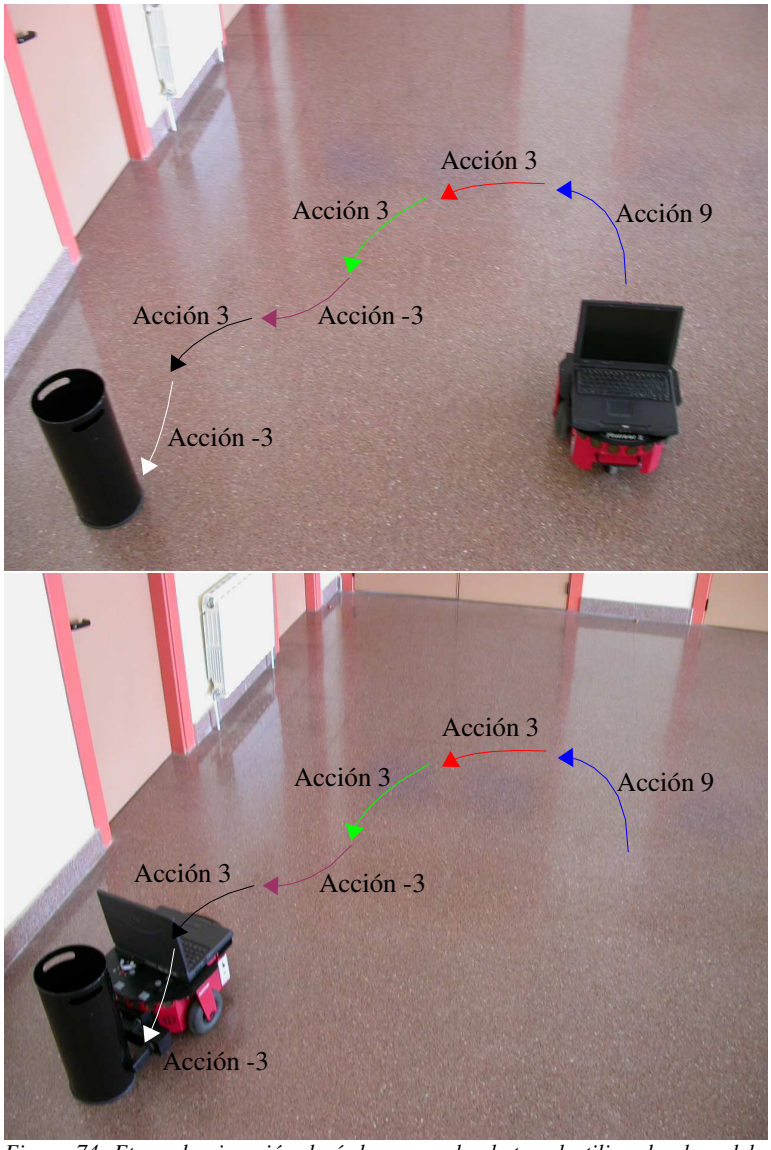


Figura 74. Etapa de ejecución de órdenes en el robot real utilizando el modelo interno inducido.

Una vez comprobado el funcionamiento en el simulador, pasamos a ejecutar el MDB en el mismo experimento pero en el robot real, obteniendo unos resultados análogos debido a que las particularidades sensoriales del problema real ya habían sido tenidas en cuenta en el simulador. La figura 74 muestra una etapa de aprendizaje idéntica a las anteriores en simulación, pero implementada en el Pioneer 2. La secuencia de acciones corresponde a la fase del ejemplo donde se utiliza el modelo interno inducido, motivo por el cual la captura del objeto es menos directa. Vemos en dicha figura como el robot avanza poco a poco, mediante pequeños giros a derecha e izquierda que siempre le llevan a disminuir la distancia al objeto. Obviamente, el profesor no daría tales órdenes, sino que el robot debería seguir recto la mayor parte de las acciones, pero el modelo interno ha aprendido a base de aumentos de distancia y ángulos pequeños, que es lo que logra con tales giros.

La implementación del MDB en el robot real tiene una serie de particularidades. Por una parte, hemos tenido que ajustar los tiempos de ejecución de cada iteración dado que las inercias del propio controlador del Pioneer 2 impiden el arranque y el frenado inmediato para evitar un daño sobre los motores. Además, debemos tener en cuenta que la secuencia de acciones mostrada en la figura 74 se obtiene tras una etapa de aprendizaje en el mundo real (no se parte de aprendizaje en simulación) que es un proceso computacionalmente costoso y se alarga hasta una media hora en el procesador Intel Celeron 400 Mhz del ordenador portátil utilizado. En los resultados concretos de esta figura, el profesor proporcionaba tanto las órdenes como los premios mediante el teclado del ordenador. Llevamos a cabo también una ejecución utilizando una fuente sonora para las órdenes y el teclado para los premios.

En conclusión a esta sección, debemos remarcar que la interacción profesor-robot ha dado unos resultados muy satisfactorios, logrando un comportamiento inducido a base de seguir las órdenes. Esta filosofía permitiría que el profesor fuese otro robot, ya que el único requisito es utilizar una codificación en lenguaje y premios que pueda ser entendida por el agente (captada) y, lo que es más importante, una forma coherente de presentar dichos premios o castigos.

7.2.3 Interacción hombre-máquina

Una vez comprobado el funcionamiento del MDB como generador de modelos de inducción, pasamos a estudiar con más profundidad la interacción hombre-máquina que se produce en este ejemplo. Por una parte, abordaremos el caso en que el profesor cambia de repente la relación entre la obediencia y los premios recibidos, de forma que el robot se debe adaptar a la nueva situación para satisfacer su motivación. Por otra parte, trataremos el caso en el que lo que cambia es el lenguaje utilizado por el profesor, de tal modo que el robot se tiene que adaptar al nuevo lenguaje con objeto de seguir recibiendo premios. Estos dos nuevos ejemplos nos sirven para probar el funcionamiento de la Memoria a Largo Plazo al alternar los distintos modelos.

Cambio de estado interno

En esta primera parte, en un instante dado el profesor cambia la relación entre obediencia y premios, es decir, el estado interno. La ejecución del experimento se puede resumir en cuatro etapas:

1. Al igual que en el apartado anterior, durante una primera fase, el profesor enseña al robot a alcanzar el objeto premiando la obediencia.

2. A partir de un cierto instante, el profesor desaparece y el MDB utiliza el modelo interno inducido en la etapa anterior para continuar con el mismo comportamiento.
3. Tras estas dos etapas vuelve el profesor, pero cambia la relación entre obediencia y premios, de modo que ahora se castiga el seguir las órdenes. Durante este periodo, el modelo interno inducido deberá cambiar para adaptarse al nuevo estado interno, ya que la motivación sigue siendo recibir premios. Es de esperar, que en esta etapa el robot no alcance el objeto prácticamente nunca.
4. Del mismo modo que con la primera fase de enseñanza, tras una serie de iteraciones de aprendizaje de la nueva tarea, el profesor deja de dar órdenes y se pasa a utilizar el modelo interno inducido.

Estas cuatro etapas serán repetidas dos veces, de modo que podremos comprobar el funcionamiento de la Memoria a Largo Plazo para los modelos de comunicación y para los modelos internos que han de cambiar. En los procesos evolutivos para los modelos, hemos aplicado el algoritmo PBGA con los mismos parámetros que en el apartado anterior, a excepción del número de iteraciones que ha aumentado hasta 4000.

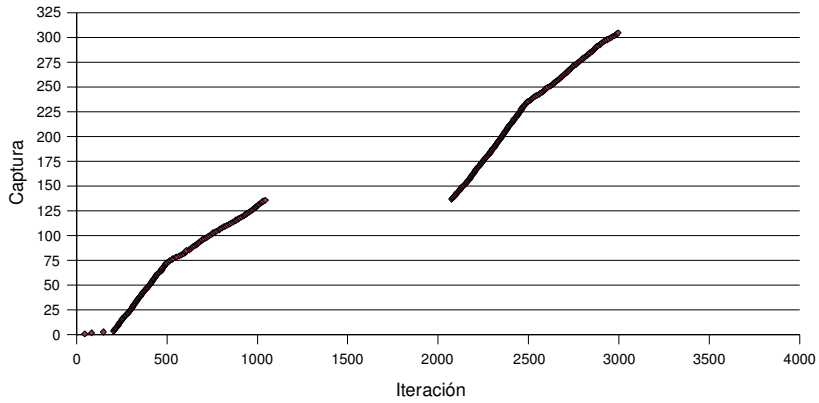


Figura 75. Capturas del objeto por parte del Pioneer 2 frente a la iteración en la que tienen lugar con cambios de estado interno cada 1000 iteraciones.

Representamos de nuevo (figura 75) las capturas del objeto que el robot realiza frente a la iteración en la que tienen lugar y se observa, como esperábamos, que sólo alcanza el objeto en las etapas donde el estado interno premia la obediencia. A partir de la iteración 200, donde cesa la fase de exploración aleatoria del entorno, el profesor cambia el estado interno cada 1000 iteraciones. Así, entre las iteraciones 200 y 500 se produce la etapa 1 explicada en el esquema anterior, entre la 500 y la 1000 la etapa 2, entre la 1000 y la 1500 la 3 y entre la 1500 y la 2000 la 4, comenzando de nuevo la etapa 1 en la iteración 2000.

Este resultado se ha obtenido tras varios cambios de profesor, con lo que se deberían haber almacenado en la Memoria a Largo Plazo los mejores modelos de comunicación e interno en cada caso. Hemos representado en la figura 76 la evolución del error cuadrático medio que proporciona el mejor modelo comunicación y, como vemos, se producen grandes aumentos en dicho error que dan lugar a detecciones de inestabilidad claras cada vez que cambiamos el estado interno. El resultado final es que la MLP almacena únicamente dos modelos de comunicación tras las 4000 iteraciones, uno para

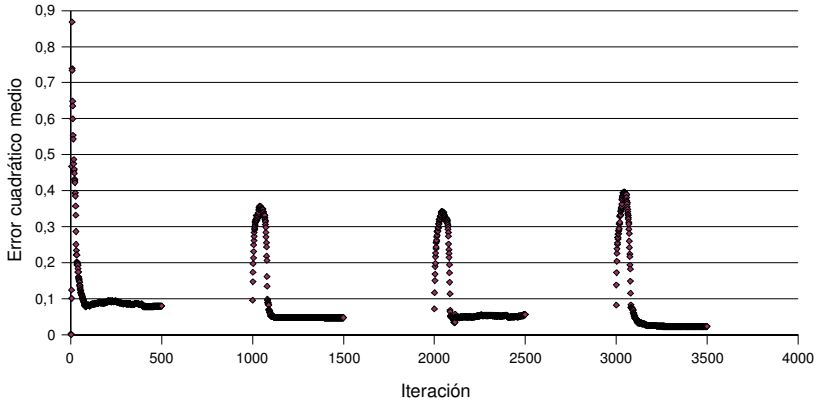


Figura 76. Evolución del error cuadrático medio que proporciona el mejor modelo de comunicación con cambio de estado interno cada 1000 iteraciones.

cada profesor, como era de esperar. Esto implica, como vemos en la figura 76, que cada vez que un profesor repite enseñanza, la adaptación del MDB es inmediata.

El comportamiento evolutivo de los modelos internos inducidos es muy similar y lo hemos representado en la figura 77. Si analizamos esta figura, veremos que, en la iteración 3000 (etapa con cambio de estado interno), no se recupera el nivel de error obtenido en la iteración 1500, algo que deberían hacer los modelos que son inyectados desde la MLP. Lo que ocurre en realidad es que el único modelo que es estable en la zona 1000-1500 tiene un nivel de error en torno a 0.3, de modo que el error mostrado en la iteración 3000 es correcto.

En ambos casos, las zonas donde no aparecen datos corresponden a la ausencia de profesor y, por tanto, no existe evolución en los modelos de comunicación ni en los modelos internos.

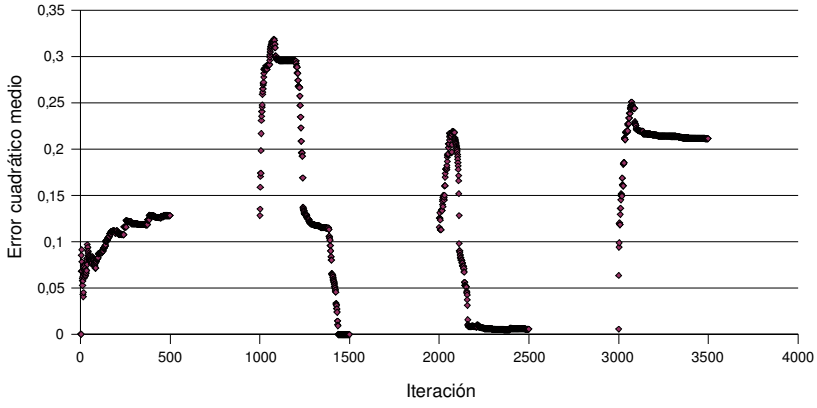


Figura 77. Evolución del error cuadrático medio que proporciona el mejor modelo interno con cambio de estado interno cada 1000 iteraciones.

En la figura 78 mostramos el resultado en simulación de una etapa de aprendizaje para este nuevo experimento de forma similar al apartado anterior, donde ahora el profesor ha cambiado el estado interno y el robot tiende a desobedecer de acuerdo con

	ORDEN	ACCIÓN	PREMIO
A	○ ○ ●	GIRO MEDIO IZDA	3,7
B	○ ● ●	GIRO PEQ IZDA	5,3
C	● ○ ○	GIRO PEQ IZDA	6,8
D	● ○ ○	GIRO MEDIO IZDA	7,9

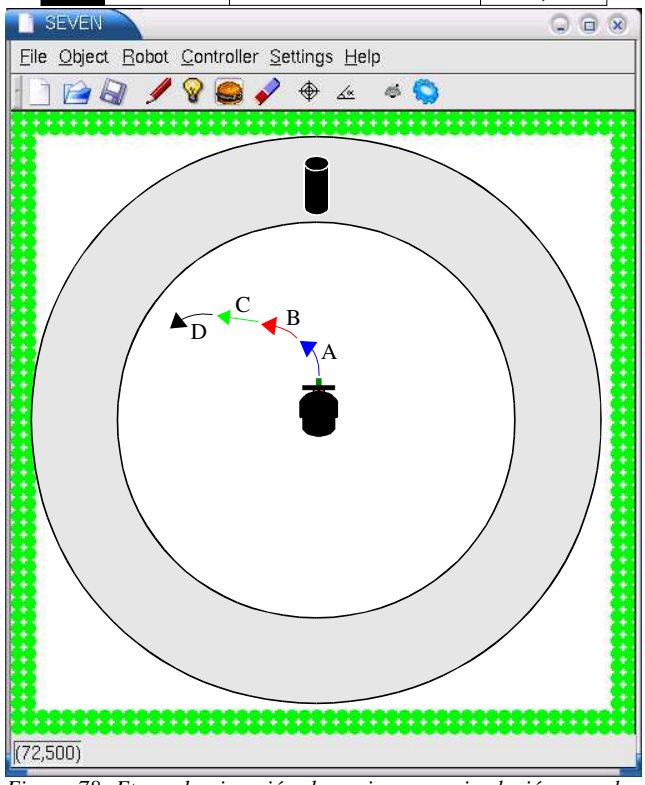


Figura 78. Etapa de ejecución de acciones en simulación para las iteraciones 1108-1111 donde la selección de acciones se basa en las órdenes del profesor tras el cambio en el estado interno.

el nuevo modelo de comunicación. Las acciones corresponden a las iteraciones 1108 a 1111, y observamos que la tendencia clara es a desobedecer logrando un premio mayor al aumentar la discrepancia. Los datos concretos de orden, acción y premio se muestran en la tabla adjunta a la figura.

Al igual que en el apartado anterior, la obtención de este nuevo comportamiento en el robot real ha sido satisfactoria y, a modo de ejemplo, mostramos la figura 79 donde las acciones corresponden a la utilización del modelo interno inducido. Obtuvimos este resultado a base de cambiar la relación entre las señales de orden y las de premio que utilizamos como sensorización de estado interno.

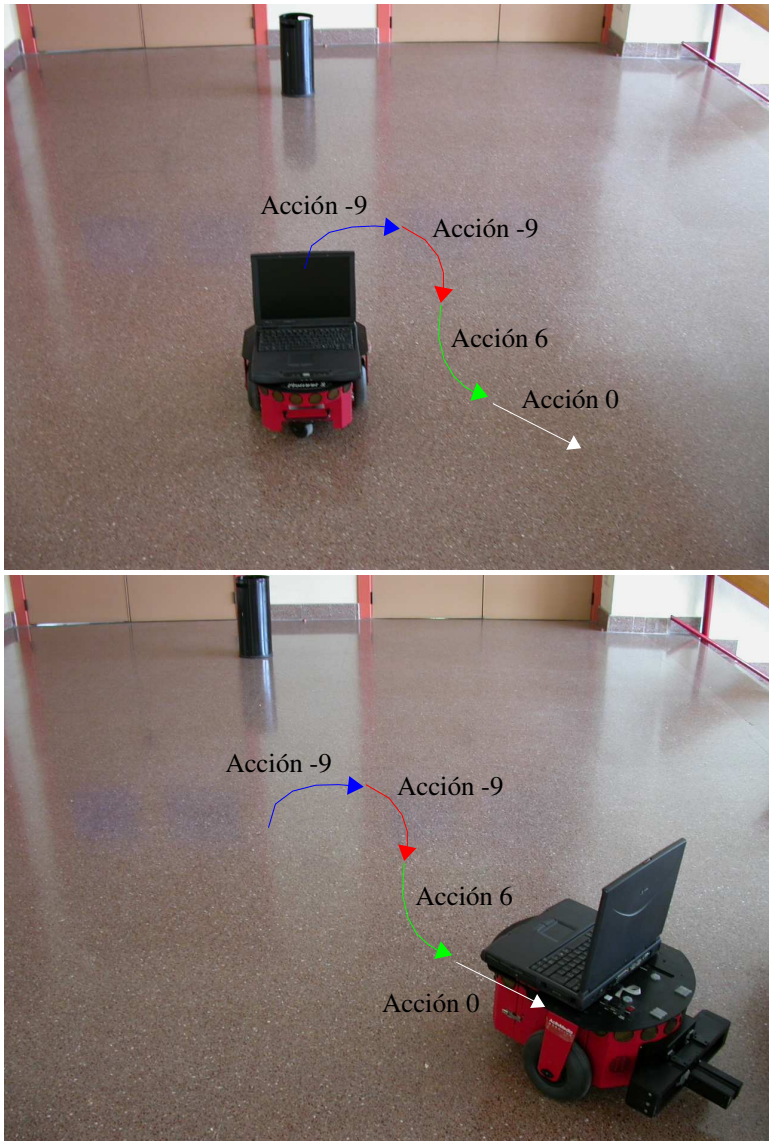


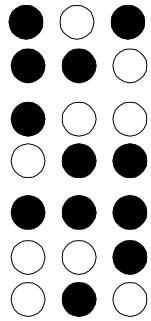
Figura 79. Etapa de ejecución de órdenes en el robot real utilizando el modelo interno inducido tras haber aprendido con un profesor nuevo que cambia el estado interno.

Como conclusión a esta prueba, se ha demostrado la adaptabilidad del MDB a cambios simultáneos en el modelo de comunicación y en el modelo interno, consiguiendo resultados muy satisfactorios con rapidez. Debemos destacar la capacidad que posee el profesor para guiar las acciones del agente con este planteamiento, cambiando las acciones que toma simplemente a base de cambiar los premios.

Cambio de lenguaje

Continuando con la filosofía de aprendizaje de más alto nivel que es la base de este ejemplo, haremos una prueba de cambio de profesor similar a la del apartado anterior pero, en este caso, cambiaremos el lenguaje utilizado. Es decir, la correspondencia entre las órdenes y las acciones ya no será la de los apartados anteriores, sino que el agente debe aprender una nueva relación entre ellas. En concreto, la nueva codificación para las órdenes, es decir, el nuevo lenguaje, es el siguiente:

- SIN GIRO :
- GIRO PEQUEÑO A LA DERECHA:
- GIRO MEDIO A LA DERECHA:
- GIRO GRANDE A LA DERECHA:
- GIRO PEQUEÑO A LA IZQUIERDA:
- GIRO MEDIO A LA IZQUIERDA:
- GIRO GRANDE A LA IZQUIERDA:



mientras que la codificación para las acciones es la misma que establecimos en el apartado 7.2.2.

Este experimento se puede dividir, al igual que en el cambio de estado interno del apartado anterior, en cuatro etapas básicas:

1. Durante la primera, el profesor enseña al robot a alcanzar el objeto con el lenguaje original (apartado 7.2.2).
2. A partir de un cierto instante, el profesor deja de dar órdenes y el MDB utiliza el modelo interno inducido para continuar con el mismo comportamiento.
3. Tras estas dos etapas vuelve el profesor, que cambia el lenguaje aunque no la relación entre obediencia y premios. Es decir, en este caso cambian los modelos de comunicación, pero no los modelos internos al no haber cambio en el estado interno.
4. Del mismo modo que con la primera fase de enseñanza, tras una serie de iteraciones de aprendizaje con el nuevo lenguaje, el profesor desaparece y se pasa a utilizar el modelo interno inducido.

En este caso, ninguna de la cuatro etapas implica cambios en el estado interno, por lo que el ritmo de capturas del objeto por parte del robot debería mantenerse constante dentro del margen de error establecido. Esta hipótesis se confirma con el resultado de capturas de la figura 80, donde el robot alcanza el objeto de forma prácticamente continuada tras cambios en el lenguaje cada 1000 iteraciones. Los parámetros del PBGA en esta ejecución son los mismos que se utilizaron en el apartado 7.2.2 a excepción del número de iteraciones que ha aumentado hasta 4000.

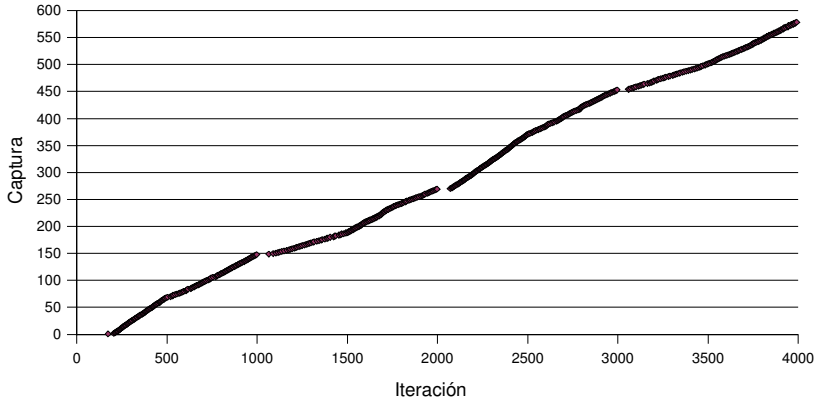


Figura 80. Capturas del objeto por parte del Pioneer 2 frente a la iteración en la que tienen lugar con cambios de lenguaje cada 1000 iteraciones.

Las zonas donde se rompe la continuidad corresponden a los cambios de lenguaje, es decir, a las transiciones donde se pasa de utilizar el modelo interno inducido al nuevo modelo de comunicación. De acuerdo con la gráfica de la figura 81 donde hemos representado la evolución del error para el modelo de comunicación actual, en estas zonas se detecta inestabilidad en los modelos de comunicación y la estrategia de reemplazo de la MCP pasa a ser de tipo temporal puro (FIFO). Los modelos que se inyectan desde la MLP tardan un tiempo en ser adecuados a las nuevas zonas y de ahí la falta de capturas que vemos en la figura 80 cada 1000 iteraciones.

Si analizamos con detalle la figura 81, vemos como el mecanismo de gestión de la MLP funciona de forma adecuada, tanto en la detección de las inestabilidades cada 1000 iteraciones, como en la inyección de los modelos en la población, logrando con rapidez el mismo nivel de error, por ejemplo, en la iteración 500 y en la 2100. De nuevo, las zonas sin datos corresponden a las etapas donde no hay profesor y, por tanto, tampoco se evolucionan los modelos de comunicación.

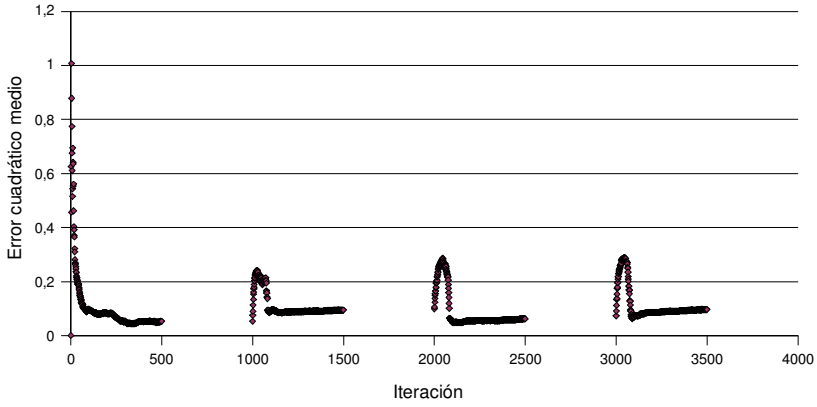


Figura 81. Evolución del error cuadrático medio para la predicción de los modelos de comunicación con cambio de lenguaje cada 1000 iteraciones.

Como ejemplo del tipo de cambio que produce el lenguaje, en la figura 82 podemos ver una serie de órdenes con la nueva codificación, correspondientes a las iteraciones 1310 a 1316 en el simulador. Como se deduce de la figura 80, las nuevas órdenes son rápidamente aprendidas y el robot alcanza el objeto sin fallo.

Lo realmente interesante de esta prueba reside en su implementación en el robot real, ya que permite que cualquier profesor se pueda comunicar con el robot sin necesidad de que exista un lenguaje preestablecido. Así, con independencia de la secuencia de señales que proporcione el profesor, siempre y cuando sea coherente, el robot la aprenderá y se adaptará a ella. Esta es la conclusión más relevante que se puede extraer de este experimento, ya que implica una gran facilidad en la interacción hombre-

	ORDEN	ACCIÓN	PREMIO
A	● ● ●	GIRO PEQ IZDA	8,3
B	● ● ●	GIRO PEQ IZDA	7,6
C	○ ○ ●	GIRO MEDIO IZDA	8,8
D	● ● ●	GIRO PEQ IZDA	8,2
E	● ● ●	GIRO PEQ IZDA	8,2
F	● ● ●	GIRO PEQ IZDA	8,2

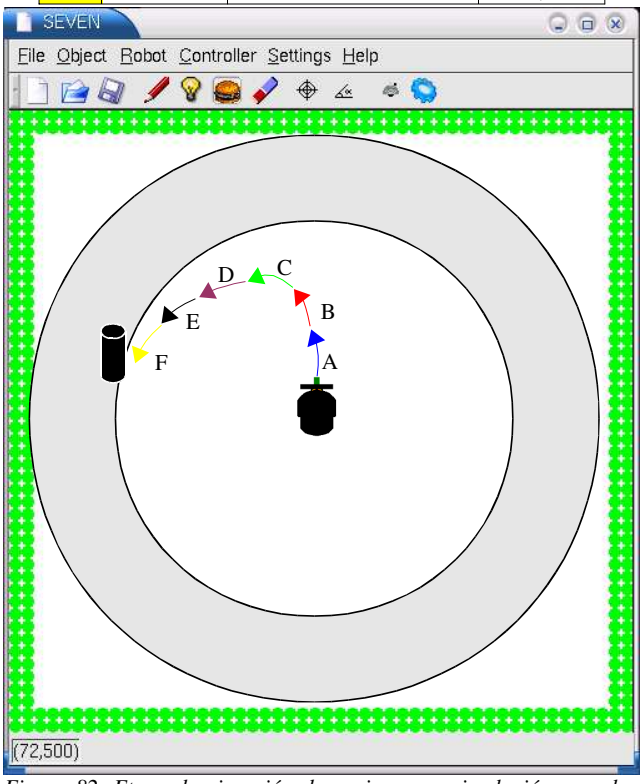


Figura 82. Etapa de ejecución de acciones en simulación para las iteraciones 1310-1316 donde la selección de acciones se basa en las órdenes del profesor en el nuevo lenguaje.

máquina al evitar que el profesor tenga que aprender el lenguaje predefinido por la máquina.

El resultado obtenido utilizando el mismo cambio de lenguaje que en el simulador se observa en la figura 83, donde representamos de forma esquemática la secuencia de acciones llevadas a cabo por el robot con el nuevo lenguaje y utilizando el modelo interno inducido.

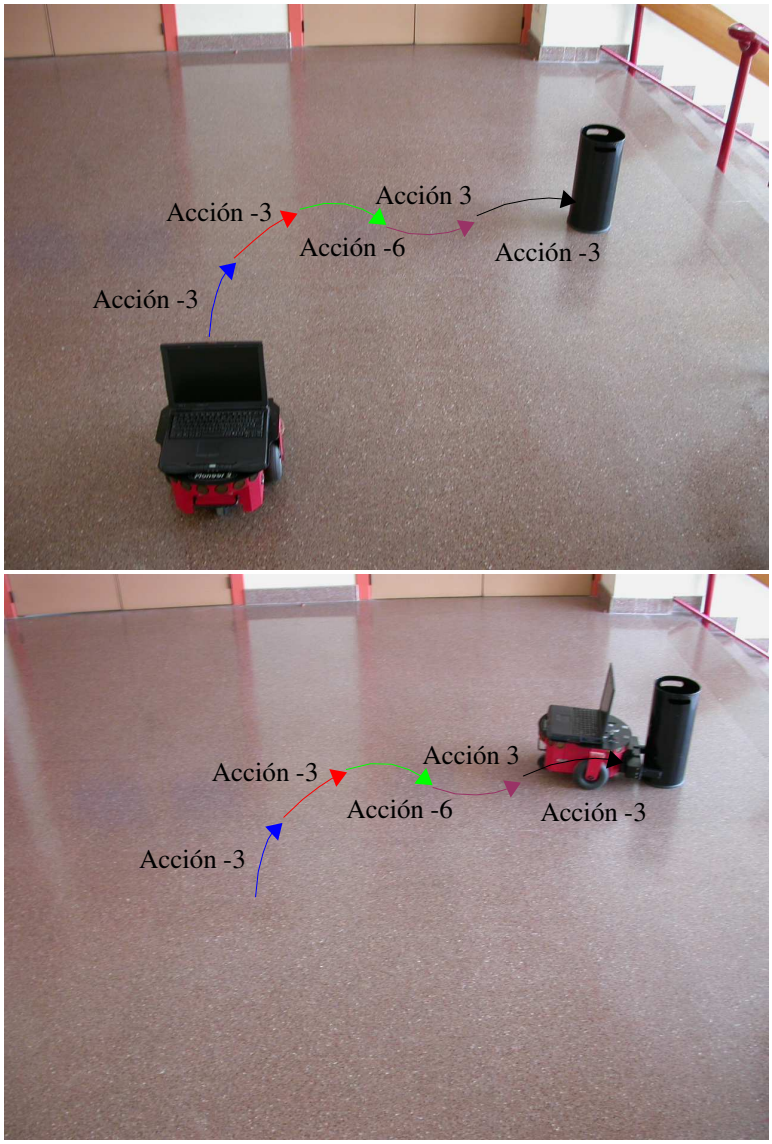


Figura 83. Etapa de ejecución de órdenes en el robot real utilizando el modelo interno inducido tras haber aprendido con un lenguaje nuevo.

7.2.4 Conclusiones a la aplicación del MDB en comportamientos de alto nivel

Son varias las conclusiones que podemos extraer de este ejemplo. La primera y más general desde el punto de vista del presente trabajo, reside en que la aplicación del MDB completo a un problema real ha resultado satisfactoria. Con el montaje experimental realizado, se ha comprobado el correcto funcionamiento del PBGA con un modelo de mundo complejo. También ha resultado correcta la estrategia de reemplazo para la Memoria a Corto Plazo, permitiendo que el modelo de mundo se pudiese aproximar con precisión mediante 60 muestras. Por último, el mecanismo de gestión de la Memoria a Largo Plazo nos ha permitido una adaptabilidad muy eficiente ante los cambios en los modelos reutilizando la información aprendida.

Además de comprobar la operatividad del MDB en un caso real, el planteamiento realizado incluyendo un modelo de mundo separado para la comunicación hombre-máquina, ha conseguido que el aprendizaje por inducción del modelo interno sea muy eficiente. Y no sólo eso, sino que proporciona capacidades de alto nivel de enseñanza al profesor, como puede ser la de guiar el comportamiento futuro del agente en su ausencia o incluso hacer que otro robot sea el profesor y supervisor.

También hemos simplificado la tarea de enseñar al agente al evitar que el profesor deba aprender un lenguaje de comunicación particular, ya que es el propio agente el que se adapta con rapidez a los cambios en dicho lenguaje.

Para finalizar, se ha ejecutado el MDB en el robot real y en tiempo real, comprobando así su correcto funcionamiento en las condiciones para las que ha sido diseñado, esto es, el aprendizaje en “tiempo de vida” del agente.

Conclusiones

8 Conclusiones

De acuerdo con lo establecido en el apartado 3, el objetivo principal de esta Tesis Doctoral ha sido el desarrollo de un Mecanismo Cognitivo que pueda ser aplicado a cualquier agente y que le proporcione capacidades adaptativas y de razonamiento inteligente de forma autónoma. El cumplimiento de este objetivo principal ha conllevado una serie de estudios y resultados que se detallan a continuación.

Hemos desarrollado un Mecanismo Cognitivo evolutivo basado en teorías darwinistas y que ha sido bautizado como MDB (Multilevel Darwinist Brain), diseñado para trabajar en tiempo real con entornos dinámicos, de modo que cualquier agente posea la capacidad de aprender por sí mismo a base de interactuar con el mundo real sin que su comportamiento esté predeterminado de antemano por el diseñador. Este planteamiento permite a los agentes predecir las consecuencias de sus acciones antes de llevarlas a cabo y proporciona un método para el aprendizaje.

De cara a enmarcar el mecanismo, se han revisado los principales trabajos llevados a cabo en el campo de las Ciencias Cognitivas aplicados a agentes autónomos y se ha establecido una definición de agente y una taxonomía propia. Además, hemos seleccionado como plataforma de aplicación inicial para el mecanismo los agentes hardware o robots autónomos. Esta elección se ha hecho, fundamentalmente, porque son plataformas de aplicación práctica complejas tanto por sus deficiencias sensoriales como actuadoras. Esto nos ha permitido desarrollar el MDB en las condiciones más adversas posibles favoreciendo así la robustez del mecanismo.

La principal novedad que aporta el MDB reside en su planteamiento de base, es decir, en las teorías biopsicológicas que lo fundamentan. Así, hemos optado por un enfoque darwinista donde se hace necesario el desarrollo interno de modelos que evolucionan, tal y como ocurre en los procesos selectivos naturales. Esto separa al MDB del enfoque reactivo que tienden a utilizar muchos autores y también del deliberativo, en el sentido de que los modelos que se utilizan internamente son simples y surgen automáticamente de la interacción con el entorno. Remarcamos así el deseo de obtener un mecanismo donde el diseñador intervenga lo menos posible a la hora de su aplicación real.

Uno de los aspectos iniciales de esta tesis ha sido el establecimiento de los elementos básicos que forman parte de un proceso de razonamiento de alto nivel donde existe una motivación que guía el comportamiento. Por una parte, hemos formalizado matemáticamente el problema global de optimización y aprendizaje que debe resolver continuamente un Mecanismo Cognitivo. Este estudio nos ha llevado a la conclusión de que es necesario un proceso de maximización del estado interno del agente, para lo cual se hace necesaria la obtención de un *modelo interno* y un *modelo de mundo* que nos proporcionan el estado interno predicho a partir de una acción y una serie de datos de sensorización del entorno. El mecanismo debe llevar a cabo de forma continuada un proceso de aprendizaje de los modelos y de optimización de la acción aplicada. Estos dos procesos se han incluido en un sistema realimentado que utiliza la información obtenida del entorno real en la interacción del agente para ir mejorando los modelos. En este sistema cerrado, se distingue una operación interna y una operación externa de manera natural (al igual que ocurre en los seres vivos). Cada una de ellas puede verse como una "caja negra" que se comunica con la otra, de forma que su funcionamiento por separado es totalmente independiente.

Además de la obtención de los modelos de mundo e interno y de la optimización de la estrategia, hemos establecido la existencia de dos elementos básicos en el MDB que son la Memoria a Corto Plazo y la Memoria a Largo Plazo. La primera almacena los pares acción-percepción que representan los datos reales (obtenidos tras la aplicación de una acción en el entorno) y que son utilizados en el aprendizaje de los modelos. La segunda almacena soluciones que resultaron adecuadas en el pasado maximizando la reutilización de la información aprendida.

Con el objeto de dotar al mecanismo de capacidades de aprendizaje autónomo para los modelos, nos basamos en un aprendizaje clásico que utiliza pares acción-consecuencia. De las distintas técnicas posibles para implementar computacionalmente los procesos de aprendizaje de los modelos, hemos justificado la elección de los algoritmos evolutivos aplicados sobre redes neuronales artificiales, que poseen grandes capacidades de generalización a partir de muestras y una probada adaptabilidad. En definitiva, el MDB tiene como base un aprendizaje por refuerzo de forma implícita, entendiendo el cálculo de la calidad de los individuos como una función de refuerzo.

En cuanto a la implementación práctica del mecanismo en problemas reales, hemos desarrollado un algoritmo genético basado en genes promotores (PBGA) para ser aplicado en entornos reales y que posee la capacidad de obtener de forma automática el tamaño de las redes que son requeridas al aproximar un modelo e incluso el número de dichas redes. Se fundamenta por un lado en una transformación genotipo-fenotipo no directa para las redes y por otro en que la evolución tiene lugar a nivel de neurona como unidad funcional. Así, en el genotipo utilizamos el concepto de gen promotor como parámetro regulador que indica si la unidad funcional a la que afecta estará o no presente en el fenotipo, permitiendo la obtención de redes de longitud variable de forma automática. El uso de genes promotores implica el almacenamiento de mucha información en forma de genes desactivados, que entran en el proceso de evolución pero no se muestran en la ejecución de la red y, por lo tanto, están apantallados en la selección.

La otra característica básica del PBGA consiste en que proporciona, de forma automática, el número de redes necesarias para modelar con la mayor precisión posible una determinada función. En la versión simple del algoritmo, las redes que se forman deben afectar a distintas salidas de la función a obtener, pero hemos extendido esta característica al desarrollar el PBGA generalizado: GPBGA. En esta versión, más avanzada y general del algoritmo, la evolución se basa en la formación de grupos de individuos por afinidad. El concepto de afinidad se concreta en un vector que define los deseos y las carencias de cada individuo de la población y representa sus preferencias a la hora de unirse a otro u otros individuos en grupos y así colaborar para lograr la meta común. Hemos desarrollado un método para la formación automática de grupos de individuos en función de sus vectores de afinidad, de modo que el concepto de individuo como solución aportada por el algoritmo evolutivo se generaliza y ahora tenemos varios individuos que, de forma conjunta, proporcionan dicha solución. Lo más destacable del GPBGA es que los vectores de afinidad que guían la formación de grupos son autoorganizativos y sus valores se establecen de manera totalmente autónoma llevando a un proceso de formación de grupos de redes que colaboran para la consecución de un objetivo, ya sea éste modelar una función o cualquier otro de una manera automática y sin que el diseñador haya de definir las características de los grupos o el número de individuos que los han de formar. Esto, a su vez, se traduce en la posibilidad de obtener una función compleja como agregación de funciones más simples

o elementales que a su vez podrán ser utilizadas como partes de otras funciones, maximizando así la reutilización de experiencia adquirida.

De esta forma, hemos sentado las bases para un algoritmo evolutivo que obtenga soluciones complejas a base de extraer automáticamente los modelos más simples y básicos posibles. La aplicación práctica del GPBGA ha resultado satisfactoria y muestra la potencia del algoritmo desarrollado.

Como hemos dicho, el modelo de mecanismo que se plantea requiere la inclusión de una memoria que almacene la información real tras la aplicación de las estrategias en el entorno, para que los modelos puedan ser aprendidos de forma gradual, denominada Memoria a Corto Plazo (MCP). Se ha estudiado el comportamiento y los requisitos de dicha memoria bajo diferentes circunstancias y se ha presentado una estrategia de reemplazo para la misma de modo que almacene las muestras más relevantes en función de las necesidades del agente en cada instante.

La estrategia de reemplazo desarrollada para la MCP se basa en cuatro términos: distancia, relevancia inicial, funcionalidad y antigüedad. Cada uno de ellos posee características muy importantes para catalogar las muestras que son susceptibles de ser almacenadas. Así, el término de distancia distribuye espacialmente las muestras, aunque no tiene en cuenta sus características funcionales. De eso se encargan tanto el término de relevancia inicial como el de funcionalidad. La inclusión de la antigüedad de las muestras a la hora de catalogarlas, nos permite almacenar las más recientes y, por tanto, eliminar las más antiguas. Del estudio en profundidad de todas ellas y de sus combinaciones que se ha llevado a cabo, podemos concluir que la estrategia de reemplazo debe ser dinámica y que la combinación de estos cuatro términos debe estar regulada por parámetros que permitan una contribución gradual de cada uno.

De cara a reutilizar los modelos optimizando así los procesos de aprendizaje, hemos incluido en el mecanismo la Memoria a Largo Plazo (MLP) y se ha presentado un mecanismo de gestión para dicha memoria. Este mecanismo consta de tres elementos fundamentales: un criterio de estabilidad para decidir qué modelos se deben almacenar en la MLP, una estrategia de reemplazo que debe comparar los modelos existentes para decidir si alguno es sustituido y un criterio de inestabilidad que debe indicar si se ha producido un cambio en el entorno o en la motivación. La detección de inestabilidades se hace imprescindible en entornos reales donde la Memoria a Corto Plazo puede guardar muestras correspondientes a distintas funciones, que haría que los modelos que se almacenan en la MLP no fuesen adecuados.

En cuanto a la reutilización del contenido de la Memoria a Largo Plazo, los modelos que almacena son inyectados en las poblaciones de los procesos evolutivos del MDB, de modo que pueden servir de semillas en las búsquedas reduciendo así el tiempo necesario de aprendizaje y favoreciendo la combinación de soluciones previas de cara a afrontar nuevas situaciones. Esta característica se ve favorecida por la utilización del algoritmo PBGA que evoluciona unidades neuronales completas, de forma que los modelos que se introducen en las poblaciones de los algoritmos evolutivos pueden aportar unidades funcionales independientes que combinadas aceleren el aprendizaje. En este sentido, el algoritmo GPBGA presentado busca la formación de grupos de individuos lo más básicos y fundamentales posibles (funcionalmente hablando), de forma que la combinación de Memoria a Largo Plazo y GPBGA favorece todavía más el aprendizaje en escalas de tiempo reducidas al inyectar modelos que son unidades funcionales independientes.

Los ejemplos de aplicación de este mecanismo de gestión para la MLP han mostrado que su funcionamiento es adecuado tanto con funciones simples como con funciones complejas obtenidas de entornos reales, permitiendo que el agente se adapte a condiciones cambiantes del entorno de forma rápida.

En cuanto a la aplicación práctica del MDB, con el objetivo de probar si el concepto de mecanismo planteado era correcto en su dominio de aplicación, hemos utilizado un robot hexápodo que debía aprender a caminar y a girar por sí mismo. La motivación de su comportamiento consistía simplemente en la minimización de la distancia a un cierto objetivo, y fue suficiente para lograr un resultado muy satisfactorio.

Cuando aprendía a caminar, en las primeras iteraciones del mecanismo, los gaits (combinaciones de amplitudes y fases en los motores de las patas) que aplicaba el robot eran pobres y acababa cayendo o bien giraba de forma inadecuada. A medida que los modelos mejoraban, los gaits seleccionados resultaban óptimos. Con el mejor gait obtenido en este primer ejemplo, se ha aplicado el MDB tratando de conseguir que el robot hexápodo gradúe la amplitud de cada movimiento de pata con el objetivo de girar para alcanzar un objetivo. De nuevo tuvimos que esperar un tiempo de aprendizaje en el que los giros eran inadecuados antes de conseguir el comportamiento buscado.

En ambos casos, lo más interesante fue comprobar que el robot aprendía de modo análogo a como lo haría cualquier ser vivo, con un periodo inicial de dudas y errores y un periodo final acertado. Desde un punto de vista didáctico, el aprendizaje en el MDB se produce tanto con los aciertos en la acciones aplicadas como con los errores. Tanto unos como los otros constituyen simplemente nueva información para mejorar los modelos.

El ejemplo final de la Tesis utiliza el MDB completo, es decir, incluye todos los elementos originales desarrollados interactuando unos con los otros. En él, el robot Pioneer 2 debe aprender a seguir las órdenes de un profesor para alcanzar una meta. En este caso hemos utilizado dos modelos de mundo, uno que relaciona sensorización y acción de forma usual, y otro que le permite entender las órdenes que le da el profesor, esto es, un modelo de comunicación. Aquí reside una de la principales novedades de este ejemplo, y es que utilizamos un lenguaje basado en símbolos que el agente aprende, pero de tal modo que cuando cambiamos dicho lenguaje, la adaptación es rápida. Esto muestra la importancia de la Memoria a Largo Plazo en el mecanismo y permite que el profesor no tenga que aprender el lenguaje del robot, sino que es el propio robot el que se adapta al profesor.

La otra característica fundamental de este ejemplo es que el agente aprende el modelo interno por inducción, es decir, aprende a satisfacer las motivaciones que le llevarán a actuar cuando ya no haya órdenes por parte del profesor. Logramos así un comportamiento de tipo *Pauloviano*, donde el agente continúa con la tarea de alcanzar el objeto aunque no se le indique de manera explícita.

Como hemos dicho, los resultados de este ejemplo han sido obtenidos utilizando todos los elementos desarrollados para el MDB. Por una parte, el PBGA permite que la obtención automática de varias redes que modelan un entorno real, complejo y lleno de fuentes de ruido. La estrategia de reemplazo de la MCP consigue almacenar muestras relevantes en la evolución de los modelos internos, algo que no es sencillo teniendo en cuenta la metodología utilizada para el aprendizaje. Por último, la MLP almacena los distintos lenguajes y estados internos, cambiando de uno a otro de forma inmediata.

Cada uno de estos tres elementos básicos que hemos implementado a lo largo del presente trabajo son independientes de su aplicación en el MDB y, de hecho, el estudio de dichos elementos se ha llevado a cabo sin aplicación explícita al mecanismo. Así, el algoritmo PBGA puede ser utilizado como cualquier otro algoritmo evolutivo que trabaje sobre redes neuronales y, por extensión, el GPBGA también. De igual modo, los mecanismos de gestión de la Memoria a Corto Plazo y de la Memoria a Largo Plazo son extrapolables fuera del MDB.

En resumen, se ha establecido un mecanismo que permite al agente, de forma paralela y en tiempo real, el establecimiento de las correspondencias sensomotoras necesarias para permitir un modelado del mundo e interno útil para la toma de decisiones en cuanto a estrategias a seguir sin que el diseñador predetermine qué se aprende, proporcionando únicamente la estructura base y dejando que el agente interactúe con el entorno.

El diseño no ha limitado el tipo de agente al que puede ser aplicado el MDB, de modo que es un mecanismo de posible aplicación allí donde se requiera un aprendizaje a partir de muestras en tiempo real y algo más que un comportamiento reactivo.

Para finalizar debemos resaltar que, tal y como se ha concebido el MDB, se fundamenta en una serie de procesos que pueden ser ejecutados en paralelo y, en general, de forma asíncrona: aprendizaje de modelos de mundo, aprendizaje de modelos internos, gestión de la Memoria a Corto Plazo y gestión de la Memoria a Largo Plazo. A la vez, el proceso fundamental de selección de acciones se está ejecutando de forma continua, actualizando los modelos sobre los que se lleva a cabo dicha selección únicamente cuando han cambiado. Este funcionamiento distribuido del MDB permite construir estructuras computacionales escalables adecuadas al coste computacional y, por lo tanto, alcanzar una ejecución verdaderamente en tiempo real independientemente del aumento de complejidad de los entornos o tareas a realizar.

Trabajo Futuro

9 Trabajo Futuro

La naturaleza del campo en el que se enmarca este trabajo y del propio mecanismo que presentamos, deja una gran cantidad de líneas de investigación para el futuro. Las Ciencias Cognitivas y la Inteligencia Artificial se encuentran en plena ebullición y, tal y como argumenta Rodney Brooks en [Brooks, 02], nos encontramos en otra de las grandes revoluciones científicas, la tecnológica y computacional, que se dirige hacia la obtención de seres artificiales lo más inteligentes posibles. Por este motivo, se nos abre un mundo de posibilidades de cara a seguir mejorando y profundizando en el MDB.

Concretando la líneas futuras más inmediatas, existe una serie de aspectos fundamentales que debemos tratar:

- Las estrategias utilizadas deben estar formadas por más de una acción, de modo que exista planificación y razonamiento a largo plazo.
- La creación de grupos por afinidad debe ser generalizada para trabajar con grupos y vectores de afinidad de cualquier tamaño. Los grupos deberán estar formados por modelos simples obtenidos de forma automática en el PBGA, de tal modo que sean estos modelos los almacenados en la Memoria a Largo Plazo.
- De acuerdo con el punto anterior, de la Memoria a Largo Plazo se podrán extraer los grupos como conjunto o bien los modelos que contienen, que podrán ser combinados para obtener nuevos grupos. Es decir, reincidimos en la idea de que la información aprendida sea reutilizada y combinada de forma automática.
- En los ejemplos de aplicación presentados, los modelos internos utilizados no incluyen como entrada la percepción interna i(t), es decir, no hemos utilizado sensores internos. En el futuro, trabajaremos con modelos internos más generales cuyas entradas sean s(t) e i(t) obtenidas cada una de ellas a partir de una función que depende de la estrategia aplicada.
- El algoritmo PBGA junto con la mecanismo de gestión de la Memoria a Corto Plazo, pueden ser aplicados a problemas de modelado de funciones en tiempo real a partir de muestras. En este sentido, debemos continuar la línea iniciada en el trabajo [López Peña, 03], donde trabajamos con datos de velocimetría láser Doppler.
- De cara a aplicar el MDB a cualquier tipo de agente, crearemos una versión del mecanismo lo más exportable posible, de modo que pueda ser visto como un módulo que requiere una mínima especificación de los sensores y actuadores del agente, y sea aplicable directamente. Entre estas futuras aplicaciones destacamos la utilización del MDB en el campo de la domótica como mecanismo que aprende del usuario, interacciona con él y es capaz de reaccionar ante los cambios en tiempo real.
- Para que la ejecución práctica del MDB sea eficiente, debemos implementar una versión distribuida del mismo, donde el coste computacional se reduzca logrando una aplicación en tiempo real.
- Por último, como estrategia de futuro a nivel del Grupo de Sistemas Autónomos, pretendemos enlazar el MDB con la herramienta SEVEN desarrollada por J.A. Becerra en [Becerra, 03], y que permite obtener de forma estructurada e incremental sistemas de control para robots autónomos que han de realizar tareas en entornos dinámicos y no estructurados. El nexo de unión entre ambos sistemas se sitúa en la formación de grupos por afinidad que estudiamos en el apartado 6.3, donde uno de

los problemas reside en establecer de manera automática el modo en que se combinan los distintos miembros del grupo. Es en este punto donde el sistema desarrollado por J.A. Becerra proporciona una metodología eficiente y totalmente autónoma para se combinen mediante modulaciones, de modo que se establezcan jerarquías de individuos dentro de la población, unos controlando la actuación de los otros.

Apéndices

Apéndice A

Sensor virtual de posición

Los sensores infrarrojos se han utilizado con frecuencia en el campo de la robótica por ser baratos, simples y bastante robustos, pero presentan problemas de imprecisión a la hora de detectar la posición exacta de un objeto. El robot Hermes II que utilizamos en el ejemplo del apartado 7.1, posee 6 sensores infrarrojos situados encima de cada una de las patas y que barren una cierta área al moverse el robot. La salida que proporciona cada uno de los sensores está codificada entre 0 y 20 (muy cerca y muy lejos respectivamente) aunque la precisión es baja y, por ejemplo, dos medidas repetidas en la mismas condiciones de sensorización dan resultados ligeramente distintos.

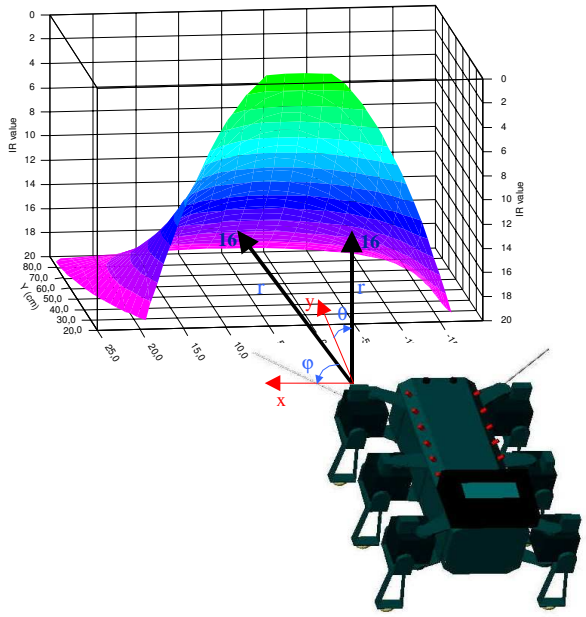


Figura 84. Curva de respuesta del sensor infrarrojo delantero izquierdo para el robot Hermes II.

En la figura 84 podemos ver la curva de respuesta del sensor delantero izquierdo donde se refleja la ambigüedad propia de estos sensores infrarrojos dado que, si situamos un objeto en los dos puntos marcados por las flechas, la respuesta sensorial será 16 en ambos casos. Como es lógico, esta ambigüedad se da a ambos lados del eje central de la curva de respuesta de cada sensor.

Para solventar el problema, hemos optado por introducir información temporal a la hora de sensar transformando así cada sensor simple en un sensor virtual que procesa la información proporcionada por el infrarrojo y devuelve resultados más complejos. La estructura utilizada para llevar a cabo el procesado de la información del sensor es una red neuronal con capacidad para manejar tiempo. Es decir, hemos intercalado una red

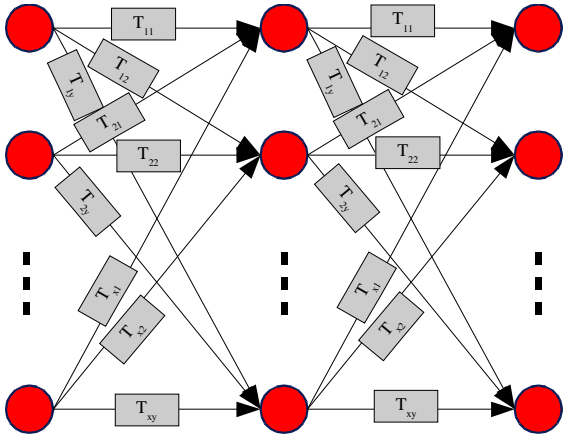


Figura 85. Esquema de una red neuronal perceptrón multicapa con retardos sinápticos.

neuronal entre la captura directa de datos por parte del sensor infrarrojo y su llegada al interfaz de salida. Un esquema de la red utilizada se muestra en la figura 85 y podemos ver cómo, además de los pesos, en las conexiones sinápticas se utilizan otros coeficientes denominados retardos, que también deben ser ajustados (entrenados en este caso). Los detalles sobre la red y el algoritmo de entrenamiento (una variación del algoritmo de retropropagación clásico) fueron presentados en el artículo de R. Duro y J. Santos [Duro, 99].

En la figura 84 podemos ver, además, el montaje experimental utilizado con los ángulos y distancias que se manejan. En la etapa de toma de datos, para distintas posiciones de un objeto enfrente del robot, se guardan los pares entrada-salida que son utilizados en el entrenamiento de la red. En cada barrido de la pata delantera izquierda se obtienen una serie de valores del sensor infrarrojo, además de conocer el ángulo que ha girado la pata desde su posición de reposo paralela al cuerpo. Utilizamos, por tanto, dos entradas a la red: el valor de infrarrojo y el ángulo de giro en cada instante discreto de tiempo (discretizamos el giro en 12 muestras). Las salidas que proporciona la red son la distancia al objeto y el ángulo en coordenadas polares.

El tamaño de red que obtuvo mejores resultados tenía dos capas ocultas de 15 neuronas cada una. Tras 20000 pasos de entrenamiento se obtuvo una red que proporciona los resultados que se muestran en la figura 86. Al situar un objeto en las posiciones señaladas por las elipses, el sensor virtual nos devuelve las posiciones predichas que están marcadas con puntos en la figura (cinco valores para cada posición del objeto). Para permitir una visualización más clara, las salidas de distancia y ángulo proporcionadas por la red se han transformado a coordenadas rectangulares (x,y), de forma que el robot estaría situado en el punto (0,0). La conclusión que se extrae de esta figura es que ahora un único sensor es capaz de distinguir con bastante precisión si un objeto está situado a su izquierda o a su derecha. Se podría aumentar la precisión de los resultados con un barrido más fino usando más de doce instantes de medida o bien presentando más casos a la hora de entrenar.

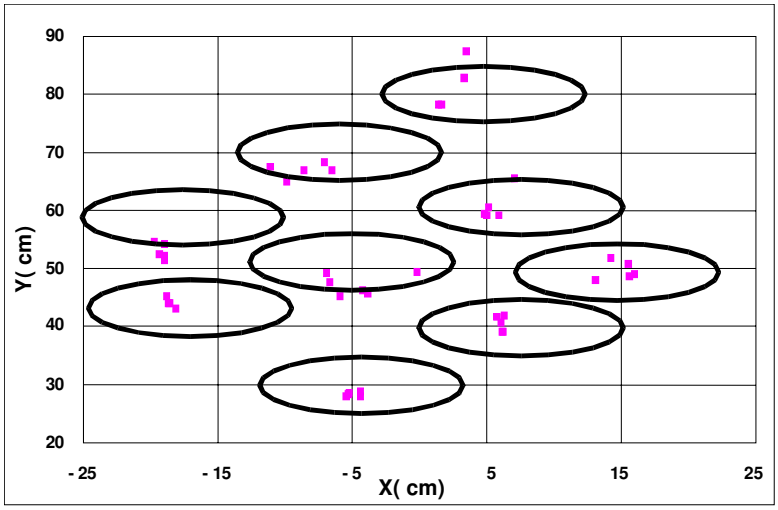


Figura 86. Posición del objeto (elipse) y posición predicha por el sensor virtual (puntos rojos).

Para analizar con más detalle esta implementación del sensor virtual y, en general, el uso de tiempo para mejorar las capacidades de los robots móviles, recomendamos consultar el artículo [Bellas, 00].

Apéndice B

En las siguientes tablas mostramos los parámetros del algoritmo PBGA (y GPBGA) que han dado lugar a los resultados que se muestran en las gráficas señaladas en la primera fila de cada tabla. Hemos incluido únicamente los parámetros del estudio realizado en el apartado 6, ya que en los ejemplos del apartado 7 ya se han presentado dichos paramétros. Las casillas con una marca * reflejan que ese parámetro es variable en dicho experimento (toma distintos valores).

PARÁMETROS DEL PBGA FIGURA 4	
Iteraciones	1000
Generaciones por iteración	20
Tamaño de la población	600
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	2-6-6-1

PARÁMETROS DEL PBGA FIGURAS 5 y 6	
Iteraciones	1000
Generaciones por iteración	100
Tamaño de la población	600
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	2-6-6-1

PARÁMETROS DEL PBGA FIGURA 7	
Iteraciones	2000
Generaciones por iteración	4
Tamaño de la población	600
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	3-8-8-2

PARÁMETROS DEL PBGA FIGURA 8	
Iteraciones	2000
Generaciones por iteración	4
Tamaño de la población	600
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	6-6-6-4

PARÁMETROS DEL PBGA FIGURAS 9, 10 y 11 (funciones seno y logística)	
Iteraciones	1000
Generaciones por iteración	*
Tamaño de la población	600
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	1-4-4-1

PARÁMETROS DEL PBGA FIGURAS 9 y 10 (función 3D)	
Iteraciones	1000
Generaciones por iteración	*
Tamaño de la población	800
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	2-6-6-1

PARÁMETROS DEL PBGA FIGURAS 13 a 18	
Iteraciones	5000
Generaciones por iteración	4
Tamaño de la población	800
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	2-5-5-1

PARÁMETROS DEL PBGA FIGURAS 19 y 20	
Iteraciones	5000
Generaciones por iteración	4
Tamaño de la población	500
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	1-4-4-1

PARÁMETROS DEL PBGA FIGURA 21	
Iteraciones	1000
Generaciones por iteración	4
Tamaño de la población	1000
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	4-6-6-2

PARÁMETROS DEL PBGA FIGURAS 22 a 28	
Iteraciones	5000
Generaciones por iteración	4
Tamaño de la población	800
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	2-5-5-1

PARÁMETROS DEL PBGA FIGURA 32	
Iteraciones	5000
Generaciones por iteración	4
Tamaño de la población	800
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	2-5-5-1

PARÁMETROS DEL PBGA FIGURAS 34 y 35	
Iteraciones	5000
Generaciones por iteración	4
Tamaño de la población	2200
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	5-12-12-3

PARÁMETROS DEL GPBGA FIGURAS 38 a 41	
Iteraciones	5000
Generaciones por iteración	4
Tamaño de la población	800
Probabilidad de cruce paramétrico	70%
Probabilidad de cruce por capa	50%
Probabilidad de mutación estructural	2%
Probabilidad de mutación paramétrica	10%
Tamaño máximo de la red	2-5-5-1

Bibliografía

Bibliografía

Adenso, B., "Optimización Heurística y Redes Neuronales *", Paraninfo*, 1996.

Agre, P., Chapman, D., "PENGI: An Implementation of a Theory of Activity*", Proceedings of the Sixth National Conference on Artificial Intelligence*, pp 268-272, 1987.

Alfeld, P., "Scattered Data Interpolation in Three or More Variables*", Mathematical Methods in Computer Aided Geometric Design*, pp 1-33, 1989.

Allen, J., Kautz, H., Pelavin, R., Tenenberg, J., "Reasoning About Plans*", Morgan Kaufmann Publishers*, 1991.

Andersson, B., Svensson, P., Nordahl, M., Nordin, P., "On-Line Evolution of Control for a Four-Legged Robot Using Genetic Programming*", EvoWorkshops 2000*, pp 319-326, 2000.

Aragon, C., Johnson, McGeoch, D., Schevon C., "Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning*", Operations Research, 39:3*, pp 378-406, 1991.

Arbib, M., Lee, H., "Visuomotor Coordination for Detour Behavior: From Retina to Motor Schemas*", Proceedings of the second International Conference on Simulation of Adaptive Behavior*, pp 42-52, 1992.

Arkin R., "Integrating Behavioral, Perceptual and World Knowledge in Reactive Navigation*", Robotics and Autonomous Systems 6*, pp 105-122, 1990.

Arkin, R., Ali, K., "Integration of Reactive and Telerobotic Control in Multi-agent Robotic Systems*", Proceedings of the third International Conference on Simulation of Adaptive Behavior*, pp 473-479, 1994.

Arkin, R., "Behavior-Based Robotics*", MIT Press*, 1998.

Ayache, N., "Artificial Vision for Mobile Robots*", MIT Press*, 1989.

Ball, G., Ling, D., Kurlander, D., Miller, J., Pugh, D., Skelly, T., Stankosky,A., Thiel, D. van Dantzich, M., Wax, T., "Lifelike Computer Characters: the Persona Project at Microsoft*", Software Agents*, pp 191-222, 1997.

Becerra, J., "Aproximación Evolutiva Para la Obtención Incremental de Arquitecturas Modulares de Comportamientos en Robots Autónomos *", Tesis Doctoral, Universidade da Coruña*, 2003.

Beer, R., Chiel H., Quinn R., Larsson P., "A Distributed Neural Network Architecture for Hexapod Robot Locomotion*", Neural Computation Nº4*, pp 356-365, 1992.

Beer R., Quinn R., Chiel H., Ritzmann R., "Biologically Inspired Approaches to Robotics*", Communications of the ACM, V.40 N. 3*, pp 30-38, 1997.

Bellas F., Becerra J., Santos J., Duro R., "Applying Synaptic Delays for Virtual Sensing and Actuation in Mobile Robots*", Proceedings IJCNN 2000*, pp 6144-6153, 2000.

Bellas, F., "Bases de un Mecanismo Cognitivo Darwinista para Agentes Autónomos*", Memoria de Licenciatura, Universidade de Santiago*, 2001.

Bellas, F., Duro, R., Becerra, J., "Funciones de Calidad Muestreadas en Algoritmos Evolucionistas*", Actas del primer congreso español de algoritmos evolutivos y bioinspirados*, pp 375-382, 2002.

Bentham, J., "Introduction to the Principles of Morals and Legislation*", Oxford: Clarendon Press,* 1789

Bradshaw, J., "Software Agents*", The MIT Press*, 1997.

Bratman, M., Israel, D., Pollack, M., "Plans and Resource-bounded Practical Reasoning*", Computational Intelligence, 4*, pp 349-355, 1988.

Bratman, M., "What is intention?*", Intentions in Communication*, pp 15-32, 1990

Brenner W., Zarnekow R., Witting H., "Intelligent Software Agents*", Springer-Verlag*, 1998.

Brooker L.B., "Instinct as an Inductive Bias for Learning Behavioral Sequences*", Proceedings of the first International Conference on Simulation of Adaptive Behavior*, pp 230-238, 1990.

Brooks R., "Aspects of Mobile Robot Visual Map Marking*", Robotics Research 2*, pp 369-375, 1984.

Brooks, R., "A Robust Layered Control System for a Mobile Robot", *IEEE J. Robotics and Automation RA-2 (1)*, pp 14-23, 1986.

Brooks, R., "Elephants Don't Play Chess", *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pp 3-15, 1991.

Brooks, R., "Cuerpos y Máquinas", *Ediciones B*, 2002.

Bäck, T., "Handbook of Evolutionary Computation", *Oxford University Press*, 1997.

Calvin, W., "The Brain as a Darwin Machine", Nature Vol. 330, pp 33-34, 1987.

Cao, Y., Fukunaga, A., Kahng, A., "Cooperative Mobile Robotics: Antecedents and Directions", *Autonomous Robots 4(1)*, pp 7-27, 1997.

Caudell, T., Dolan, C., "Parametric Connectivity: Training of Constrained Networks using Genetic Algorithms", *Proceedings ICGA 1989*, pp 370-374, 1989.

Changeux, J., Courrege, P., Danchin, A., "A Theory of the Epigenesis of Neural Networks by Selective Stabilization of Sinapsis", *Proceedings Natl. Acad. Sci. USA 70*, pp 2974-2978, 1973.

Changeux, J., Danchin, A., "Selective Stabilization of Developing Synapsis as a Mechanism for the Specification of Neural Networks", Nature 264, pp 705-712, 1976.

Changeux, J., Heidmann, T., Patte, P., "Learning by Selection", *Springer-Verlag*, 1984.

Chapman, D., "Planning for conjunctive goals", *Artificial Intelligence, 32*, pp 333-378, 1987.

Chavez, A., Maes, P., "Kasbah: An Agent Marketplace for Buying and Selling Goods", *Proceedings the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pp 75-90, 1996.

Cherian S., Troxell W., "A Neural Network Based Behavior Hierarchy for Locomotion Control", *Proceedings of the second International Conference on Simulation of Adaptive Behavior*, pp 61-71, 1992.

Chiel, H., Beer, R., "The Brain has a Body: Adaptive Behavior Emerges from Interactions of Nervous System, Body and Environment", *Trends in Neurosciences 20*, pp 553-557, 1997.

Coello, C., Van Veldhuizen, D., Lamont, G., "Evolutionary Algorithms for Solving Multi-Objective Problems", *Kluwer Academic Publishers*, 2002.

Cohen, P., Levesque, H., "Intention is choice with Commitment", *Artificial Intelligence, 42*, pp 213-261, 1990.

Conrad, M., "Evolutionary Learning Circuits", *Theor. Biol. 46*, pp 167-188, 1974

Conrad, M., "Complementary Molecular Models of Learning and Memory", *BioSystems 8*, pp 119-138, 1976.

Crespo, J., Santos, J., Duro, R., "Robust Visual Recognition with High-Order Gaussian Synapses Networks", *Proceedings of the International Joint Conference on Artificial Neural Networks*, pp 6135-6141, 2000.

Darwen, P., Yao, X., "Speciation as Automatic Categorical Modularization", *IEEE Transactions on Evolutionary Computation, 1(2)*, pp 101-108, 1997.

Dasgupta, D., McGregor, D., "sGA : A Structured Genetic Algorithm", Technical Report no. IKBS-11-93, 1993.

Davison, A., Murray, D., "Simultaneous Localization and Map-Building Using Active Vision", *Pattern Analysis and Machine Intelligence*, pp 865-880, 2002.

Delcomyn, F., Nelson, M., Cocatre-Zilgien, J., "Sense Organs of Insect Legs and the Selection of Sensors for Agile Walking Robots", *International Journal of Robotics Research V. 15, N. 2*, pp 113-127, 1996.

Dellaert, F., "Toward a Biologically Defensible Model of Development", *Master Thesis, Case Western Reserve University*, 1995.

Digney B., Gupta M., "A Distributed Adaptive Control System for a Quadruped Mobile Robot", *Proceedings of the third International Conference on Simulation of Adaptive Behavior*, pp 344-354, 1994.

Donnart, J., Meyer, J., "A Hierarchical Classifier System Implementing a Motivationally Autonomous Animat", *Proceedings of the third International Conference on Simulation of Adaptive Behavior*, pp 144-154, 1994.

Duro, R., Santos, J., "Discrete Time Backpropagation for Training Synaptic Delay Based Artificial Neural Networks", *IEEE Transactions on Artificial Neural Networks Vol.10, No. 4*, pp 779-790, 1999.

Edelman, G., "Neural Darwinism. The Theory of Neuronal Group Selection", *Basic Books,* 1987.

Etzioni, O., "A Softbot-Based Interface to the Internet", *Communications of the ACM*, pp 72-76, 1994.

Etzioni, O., Lesh, N., Richard S., "Building softbots for Unix ", *Working Notes of the AAAI Spring Syposium on Software Agents*, pp 9-16, 1994.

Etzioni, O., Weld, D., "Intelligent Agents on the Internet: Fact, Fiction and Forecast", *IEEE Expert 10(4)*, pp 44-49, 1995.

Everett, H., "Sensors for Mobile Robots", *A K Peters Ltd*, 1995.

Fensel, D.,Hendler, J., Lieberman H., Wahlster, W., "Spinning the Semantic Web", *MIT Press*, 2003.

Feo, T., Resende, M., "A Probabilistic Heuristic for a Computationally Difficult set Covering Problem", *Operations Research Letters, 8*, pp 67-71, 1989.

Ferguson, I., "TuringMachines: An Architecture for Dynamic, Rational, Mobile Agents", *Cambridge University Press*, 1992.

Fiesler, E., Beale, R., "Handbook of Evolutionary Computation", *Oxford University Press*, 1997.

Firby R., "Adaptive Execution in Complex Dynamic Worlds", *Ph. D. Thesis, Yale University*, 1989.

Fischer, K., Kuhn, N., "A DAI-Approach to Modelling the Transportation Domain ", *Technical Report RR-93-25, Deutsches Forschungszentrum für Künstliche Intelligenz*, 1993.

Floreano, D., Mondada, F., "Evolution of Homing Navigation in a Real Mobile Robot", *IEEE Transactions on Systems Man and Cybernetics Part B: Cybernetics, 26(3)*, pp 396-407, 1996.

Fogel, D., Fogel, L., Porto, V., "Evolving Neural Networks", *Biological Cybernetics, 63*, pp 487-493, 1990.

Fogel, D., "Evolutionary Computation", *IEEE Press*, 1998.

Foley, T., "Scattered Data Interpolation and Approximation with Error Bounds", *Computer Aided Geometric Design, Num 3*, pp 163-177, 1986.

Foner, L., Maes, P., "Paying Attention to What's Important: Using Focus of Attention to Improve Unsupervised Learning", *Proceedings of the third International Conference on Simulation of Adaptive Behavior*, pp 256-266, 1994.

Foner, L., "Entertaining Agents: a Sociological Case Study", *The Proceedings of the First International Conference on Autonomous Agents (AA '97)*, pp 122-129, 1997

Franklin, S., "Artificial Minds", *MIT Press*, 1995.

Franklin S., Graesser A., "Is it an Agent, or just a Program? A taxonomy for Autonomous Agents", *Procs. 3rd. International Workshop on Agent Theories, Architectures and Languages*, pp 21-35, 1996.

Gat E., "Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real World Mobile Robots", *Proceedings AAAI92*, pp 809-815, 1992.

Georgeff, M., Lansky, A., "Reactive Reasoning and Planning", *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp 677-682, 1987.

Ghosh, B., Xi, N., Tarn, T., "Control in Robotics and Automation", *Academic Press*, 1999.

Giarrantano, J., Riley, G., "Expert Systems: Principles and Programming", *PWS*, 1994.

Glover, F., Laguna, M., "Tabu Search", *Modern Heuristic Techniques for Combinatorial Problems*, pp 70-150, 1993.

Goldberg, D., Richardson, J., "Genetic Algorithms with Sharing for Multimodal Function Optimization", *Proceedings of the 2nd International Conference on Genetic Algorithms*, pp 41-49, 1987.

Goldberg, D., "Genetic Algorithms", *Addison-Wesley*, 1989.

Gould J., Marler, P., "Ethology and the Natural History of Learning", *The Biology of Learning*, pp 47-74, 1984.

Grefenstette, J., "Genetic Algorithms for Changing Environments", *Parallel Problem Solving from Nature 2*, pp 137-144, 1992.

Hartono, P., Hashimoto, S., "Migrational GA that Preserves Solutions in Non-Static Optimization Problems", *Proceedings of IEEE-SMC 2001*, pp 255-260, 2001.

Harvey, I., Husbands, P., Cliff, D., "Seeing the Light: Artificial Evolution; Real Vision", *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pp 392-401, 1994.

Hayes-Roth, B., "An Architecture for Adaptive Intelligent Systems", *Artificial Intelligence: Special Issue on Agents and Interactivity, 72*, pp 329-365, 1995.

Hebb, D., "The Organization of Behavior", *Wiley*, 1949.

Heiligenberg, W., "Processes Governing Behavioral States of Readiness", *Adv. Study Behavior 5*, pp 173-200, 1974.

Holland, J., "Robust Algorithms for Adaptation set in a General Formal Framework", *Proceedings of the IEEE Symposium on Adaptive Processes*, pp 127-146, 1970.

Johnson, D., Aragon, C., McGeoch L., Shevon, C., "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning", *Operations Research, 37:6*, pp 865-893, 1989.

Kaelbling L., "An Architecture for Intelligent Reactive Systems" Reasoning about Actions and Plans", *Proceedings for the 1986 Workshop on Reasoning about Actions and Plans*, pp 395-410, 1987.

Kaelbling, L., "A Situated Automata Approach to the Design of Embedded Agents", *SIGART Bulletin, 2(4)*, pp 85-88, 1991.

Kargupta, H., Sarkar, K., "Function Induction, Gene Expression, and Evolutionary Representation Construction", *Proceedings of the Genetic and Evolutionary Computation Conference. vol 1*, pp 313-320, 1999.

Kaufmann, A., Gupta, M., "Introduction to Fuzzy Arithmetic Theory and Application", *Van Nostrand Reinhold*, 1991.

Keil, F. C., "Concepts, Kinds, and Cognitive Development", *MIT Press*, 1989.

Kincaid, D., Cheney, W., "Análisis Numérico", *Addison-Wesley*, 1994.

Kirpatrick S., Gelatt, C., Vecchi, M., "Optimization by Simulated Annealing", *Science 220*, pp 671-680, 1983.

Kitano, H., "Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms", *Proceedings of the National Conference on Artificial Intelligence*, pp 789-795, 1990.

Kodjabachian, J., Meyer J., "Evolution and Development of Neural Networks Controlling Locomotion, Gradient-Following, and Obstacle-Avoidance in Artificial Insects", *IEEE Transactions on Neural Networks 9 (5)*, pp 796-812, 1998.

Kortenkamp, D., Bonasso, C. Murphy, R., "Artificial Intelligence and Mobile Robots", *MIT Press*, 1998.

Kosko, B., "Heaven in a Chip: Fuzzy Visions of Society and Science in the Digital Age", *Three Rivers Press/Random House*, 2000.

Koza, J., "Genetic Programming", *MIT Press*, 1992.

Koza, J., "Genetic Programming II: Automatic Discovery of Reusable Programs", *MIT Press*, 1994.

Koza, J., Bennett, F., Andre, D., Keane, M., "Genetic Programming III: Darwinian Invention and Problem Solving", *Morgan Kaufmann*, 1999.

Koza, J., Keane, M., Streeter, M., Mydlowec, W., Yu, J., Lanza, G., "Genetic Programming IV: Routine Human-Competitive Machine Intelligence", *Kluwer Academic*, 2003.

Kubalik, J., Rothkrantz, L., "Genetic Algorithm with Limited Convergence", *Proceedings 6th Joint Conference on Information Sciences 2002*, pp 610-613, 2002.

Laguna, M., Martí, R., "Scatter Search", *Kluwer Academic*, 2003.

Lamas, A., "Seguimiento de Trayectorias por Control Borroso de un Robot Móvil en un entorno DADS/Matlab", *Proyecto fin de carrera*, *Universidade de Vigo*, 1999.

Lieberman, H., "Letizia: An Agent that Assists Web Browsing", *Proceedings of IJCAI 95*, pp 924-929, 1995.

López Peña, F., Bellas, F., Duro, R., Sánchez, M., "Reconstructing Irregularly Sampled Laser Doppler Velocimetry Signals by Using Artificial Neural Networks", *Proceedings IDAACS 2003*, pp 99-105, 2003.

Maes, P., "Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back", *MIT Press*, 1990.

Maes, P., "The Agent Network Architecture (ANA)", *SIGART Bulletin, 2(4)*, pp 115-120, 1991.

Mann, S., "Cubic precision Clough-Tocher interpolation", *Computer Aided Geometric Design, Vol 16*, pp 85-55, 1999.

Martí, R., "Arc Crossing Minimization in Graphs with GRASP", *IEEE Transactions, 33*, pp 913-919, 2001.

Marín J., Solé R, "Macroevolutionary Algorithms: A New Optimization Method on Fitness Landscapes", *IEEE Transactions on Evolutionary Computation Vol 3 Nº4*, pp 272-28, 1999.

Mascaro, S., Korb, K., Nicholson, A., "Suicide as an Evolutionary Stable Strategy", *Advances in Artificial Life. 6th European Conference ECAL 2001*, pp 120-132, 2001

Mataric M., "Designing Emergent Behaviors: From Local Interactions to Collective Intelligence", *Proceedings of the second International Conference on Simulation of Adaptive Behavior*, pp 432-442, 1992.

Mataric, M., Cliff, D., "Challenges in Evolving Controllers for Physical Robots", *Evolutional Robotics, special issue of Robotics and Autonomous Systems, 19(1)*, pp 67-83, 1996.

Mataric, M., "Learning in Behavior-Based Multi-Robot Systems: Policies, Models, and Other Agents", *Cognitive Systems Research*, pp 81-93, 2001.

McGovern, A., Barto, A., "Accelerating Reinforcement Learning through the Discovery of Useful Subgoals", *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS 2001*, pp 570-576, 2001.

McKerrow, P., "Introduction to Robotics", *Addison-Wesley*, 1991

Mitchell, T., Caruana, R., Freitag, D., McDermott, J., Zabowski, D., "Experience with a Learning Personal Assistant", *Communications of the ACM 37 (7)*, pp 80-91, 1994.

Montana, D., Davis L., "Training Feedforward Neural Networks using Genetic Algorithms", *Proc. II th lnt. Joint Conf. on Artificial Intelligence*, pp 762-767, 1989.

Moore, R., "A Formal Theory of Knowledge and Action", *Formal Theories of the Commonsense World*, pp 319-358, 1985.

Moriarty, D., Miikkulainen, R., "Efficient Learning from Delayed Rewards through Symbiotic Evolution", *Machine Learning: Proceedings of the Twelfth International Conference*, pp 396-404, 1995.

Mura, F., Franceschini, N., "Visual Control of Altitude and Speed in a Flying Agent", *Proceedings of the third International Conference on Simulation of Adaptive Behavior*, pp 91-102, 1994.

Müller, J., "A Conceptual Model of Agent Interaction", *Draft proceedings of the Second International Working Conference on Cooperating Knowledge Based Systems*, pp 389-404, 1994.

Nayar, S., Poggio, T., "Early Visual Learning", *Oxford University Press*, 1996.

Newell, A., Simon, H., "Computer Science as Empirical Inquiry: Symbols and search ", *Communications of the Association for Computing Machinery, 19(3)*, pp 113-126, 1976.

Nilsson N., "Action Networks", *Proceedings of the Rochester Planning Workshop: From Formal Systems to Practical Systems*, pp 20-51, 1989.

Nordin, P., Banzhaf, W., Brameier, M., "Evolution of a World Model for a Miniature Robot Using Genetic Programming", *Robotics and Autonomous Systems Vol. 25*, pp 105-116, 1998.

Nwana, H., "Software Agents: An Overview", *Knowledge Engineering Review, vol 11*, pp 1-40, 1996.

O'Hare G., Jennings N., "Foundations of Distributed Artificial Intelligence ", *Wiley-Interscience*, 1996.

Parker L., "Adaptive Action Selection for Cooperative Agent Teams", *Proceedings of the second International Conference on Simulation of Adaptive Behavior*, pp 442-451, 1992.

Peow, K., Wong, K., "A New Diploid Scheme and Dominance Change Mechanism for Nonstationary Function Optimisation", *Proceedings of the Sixth International Conference on Genetic Algorithms 1995*, pp 159-166, 1995.

Pujol, J., Poli, R., "Efficient Evolution of Asymetric Recurrent Neural Networks Using a PDGP-inspired Two-dimensional Representation", *Proceedings of the First European Workshop on Genetic Programming*, pp 130-141, 1998.

Rao, A., Georgeff, M., "A model-theoretic Approach to the Verification of Situated Reasoning Systems", *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp 318-324, 1993.

Rennie, J., McCallum, A., "Using Reinforcement Learning to Spider the Web Efficiently", *Proceedings of ICML'99*, pp 335-343, 1999.

Resende, M., Teo, T., "Greedy Randomized Adaptive Search Procedures", *Journal of Global Optimization*, pp 109-133, 1995.

Rich, E., Knight, K., "Inteligencia Artificial", *McGraw-Hill*, 1994.

Roitblat, H., Moore, P., Helweg, D.,Nachtigall, P., "Representation and Processing of Acoustic Information in a Biomimetic Neural Network", *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pp 90-100, 1992.

Rosenblatt, F., "Two Theorems of Statistical Separability in the Perceptron", *Mechanization of Thought Processes*, pp 421-456, 1957.

Rosenschein S., Kaelbling L., "The Synthesis of Digital Machines with Provable Epistemic Properties", *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge*, pp 83-98, 1987.

Russell, S., Norvig, P., "Inteligencia Artificial", *Prentice Hall*, 1996.

Ryan, C., Collins, J., "Polygenic Inheritance - A Haploid Scheme that Can Outperform Diploidy", *Proceedings PPSN 1998*, pp 178-187, 1998.

Sarkar, S., Boyer, K., "Perceptual Organization in Computer Vision: A Review and a Proposal for a Classificatory Structure", *IEEE Trans on Systems, Man and Cybernetics. V. 23, N. 2*, pp 382-399, 1993.

Schaffer, J., "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms", *Proceedings on the First International Conference on Genetic Algorithms*, pp 93-100, 1985.

Singh, M., "Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications", *Springer-Verlag*, 1994.

Singh, M., Cannata, P., Jacobs,N., Ksiezyk, T., Ong, K., Sheth, A., Tomlinson, C., Woelk, D., "The Carnot Heterogeneous Database Project: Implemented Applications", *Distributed and Parallel Databases: An International Journal, 5(2)*, pp 207-225, 1997.

Smart, W., Kaelbling, L., "Practical Reinforcement Learning in Continuous Spaces", *Proceedings of the Seventeenth International Conference on Machine Learning*, pp 903-910, 2000.

Solé, R., Manrubia, S., "Criticality and Unpredictability in Macroevolution", *Phyical Review E 55*, pp 4500-4508, 1997.

Sony, 03, http://www.sonymagiclink.com/

Steels, L., "A Selectionist Mechanism for Autonomous Behavior Acquisition", *Robotics and Autonomous Systems, vol. 20, nr. 2-4*, pp 117-131, 1997.

Steep, R., Cammarata, S., Hayes-Roth, F., Thorndyke, P., Wesson, R., "Distributed Intelligence for Air Fleet Control", pp 125-129, 1988.

Sutton, R., "Two Problems with Backpropagation and other Steepest-Descent Learning Procedures for Networks", *Proceedings of the 8th anual conference of the Cognitive Science Society*, pp 823-831, 1986.

Sutton, R., Barto, A., "Reinforcement Learning", *The MIT Press*, 1998.

Sycara, K., "Distributed Intelligent Agents", *IEEE Expert 1996*, pp 36-46, 1996.

Tenenberg, J., Karlsson, J., Whitehead, S., "Learning Via Task Decomposition", *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pp 337-346, 1992.

Van der Smagt, Dev, A., Groen, F., "A Visually Guided Robot and a Neural Network join to Grasp Slanted objects", *Proceedings of the 3rd SNN Symposium on Neural Networks*, pp 144-150, 1995.

Van Dyke Parunak, H., "Agents in Overalls: Experiences and Issues in the Development and Deployment of Industrial Agent-Based Systems", *IJCIS 9(3)*, pp 209-228, 2000.

Varga, L., Jennings, N., Cockburn D., "Integrating Intelligent Systems into a Cooperating Community for Electricity Distribution Management", *Intenational Journal of Expert Systems with applications, 7(4)*, pp 563-579, 1994.

Vassilev, V., Miller, J., "The Advantages of Landscape Neutrality in Digital Circuit Evolution", *Proceedings ICES 2000*, pp 252-263, 2000.

Vere, S., Bickmore, T., "A basic agent", *Computational Intelligence, 6*, pp 41-60, 1990.

Watson, J., "Behavior-Based Control for Autonomous Robotics", *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pp 185-190, 1994.

Weihmayer, R., Velthuijsen, H., "Intelligent Agents in Telecommunications", *Agent Technology, Foundations, Applications, and Markets*, pp 203-219, 1998.

Weiss, G., "Combining Neural and Evolutionary Learning: aspects and approaches", *Institut fur Informatik Technical Report*, 1990.

Weiss, G., "Multiagent Systems: a Modern Approach to DAI", *MIT Press*, 1999.

Wilson, S., "Perceptron Redux: Emergence of Structure", *Physica D, 42(1-3)*, pp 249-256, 1990.

Wooldridge, M., "The Logical Modelling of Computational Multi-Agent Systems", 1992.

Wooldridge M., Jennings N., "Intelligent Agents: Theory and Practice", *Knowledge Engineering Review 10(2)*, pp 115-152, 1995.

Wooldridge M., Jennings N., "Agent Theories, Architectures, and Languages: a Survey", *Intelligent Agents*, pp 1-22, 1995.

Yao, X., "Optimization by Genetic Annealing", *Proc. of the 2nd Australian Conf. on Neural Networks*, pp 94-97, 1991.

Yao, X., "A Review of Evolutionary Artificial Neural Networks", *International Journal of Intelligent Systems, 8(4)*, pp 539-567, 1993.

Yao, X., "Automatic Acquisition of Strategies by co-evolutionary Learning", *Proc. of the International Conference on Computational Intelligence and Multimedia Applications*, pp 23-29, 1997.

Zadeh, L., "Fuzzy Sets", *Information & Control, 8*, pp 338-353, 1965.