FACULTAD DE INFORMÁTICA

Departamento de Tecnologías de la Información y las Comunicaciones

TESIS DOCTORAL

Desarrollo y Simplificación de Redes de Neuronas Artificiales mediante el uso de Técnicas de Computación Evolutiva

Doctorando: Daniel Rivero Cebrián

Directores: Julián Dorado de la Calle
Juan Ramón Rabuñal Dopico

A Coruña, 2007

DEPARTAMENTO DE TECNOLOXÍAS DA INFORMACIÓN E AS COMUNICACIÓNS

Facultade de Informática
Campus de Elviña, s/n.
15071 A Coruña
Telf. 981 16 70 00
Fax: 981 16 71 60
E-mail: tic@udc.es

D. Julián Dorado de la Calle, profesor Titular de Universidad del área de Ciencias de la Computación e Inteligencia Artificial del Dpto. De Tecnoloxías da Información e as Comunicación en la Universidade da Coruña y **D. Juan Ramón Rabuñal Dopico**, profesor a tiempo completo del mismo área de conocimiento, en cumplimiento del punto 1 del art. 18 del reglamento de estudios de doctorado, aprobado en Junta de Gobierno del 9 de junio de 1999, y modificado en el Consejo de Gobierno del 22 de junio de 2005,

AUTORIZAN la presentación de la tesis doctoral titulada *"Desarrollo y Simplificación de Redes de Neuronas Artificiales mediante el uso de técnicas de Computación Evolutiva"*, que ha sido realizada bajo nuestra dirección por **D. Daniel Rivero Cebrián**, en el Departamento de Tecnologías da Información e as Comunicacións de la Universidade da Coruña, y que presenta para optar al grado de Doctor en Informática.

A Coruña, a 6 de junio de 2007

Dr. Julián Dorado de la Calle Dr. Juan Ramón Rabuñal Dopico

AGRADECIMIENTOS

En primer lugar, a mis directores de Tesis, Julián Dorado de la Calle y Juan Ramón Rabuñal Dopico, por sus sabios consejos y su guía, que ha resultado imprescindible para llevar a buen puerto este trabajo.

A todos los miembros del grupo de investigación RNASA-IMEDIR, por hacerme sentir en el laboratorio como en casa.

A mi familia, por su constante apoyo en todo momento y la enorme paciencia que han tenido conmigo.

A mis amigos y mi novia, porque con ellos he vivido las mayores experiencias de mi vida, y me han enseñado aquello que no se aprende en la escuela.

A todos, muchas gracias.

A mi familia

“The answer to any question you could wish to know can be found in nature.
All you need to know is how to ask the question and how to recognise the answer”
‘The Power of One’, Bryce Courtenay

ÍNDICE

CAPÍTULO
1 INTRODUCCIÓN

1.1 Motivación

En 1950 Alan Turing formuló la cuestión "¿Pueden pensar las máquinas?". Aventuró que, en el plazo de un siglo, un ordenador debería ser capaz de pasar un test de inteligencia estándar (conocido como "El test de Turing") en no menos del 30% de pruebas que se realizasen. A principios del siglo XXI, pocos investigadores comparten la confianza que tenía Turing en que una máquina probaría su inteligencia en un corto período de tiempo. Sin embargo, muchos investigadores sí se dedican al desarrollo de tal máquina. Este es el objetivo del trabajo en el campo de los sistemas adaptativos.

El término sistemas adaptativos abarca numerosas técnicas que, aplicadas a un problema, ajustan la relativa importancia de los parámetros de entrada de manera autónoma para conseguir la resolución de ese problema. Se quiere conseguir, por tanto, una autonomía en el aprendizaje de las máquinas, lo cual llevaría a un antiguo sueño en el reciente campo de estudio de la inteligencia artificial: la programación automática.

Ya en la década de los 50 Arthur Samuel planteó esta cuestión: "¿Cómo pueden los ordenadores aprender a resolver problemas sin ser explícitamente programados?". Con esta pregunta se iniciaba el mundo de la programación automática, un campo de estudio que abarca distintas técnicas, y de las cuales algunas de las más exitosas se basan en copiar estructuras presentes en la naturaleza, con el objetivo de conseguir sistemas que se adapten a su entorno de una forma similar a lo que ocurre en la naturaleza. Estas técnicas se agrupan en el campo de estudio llamado Computación Evolutiva.

Hoy en día, a pesar de que todavía se ve lejos aquella afirmación realizada por Turing, los investigadores sí son más optimistas respecto a la programación automática,

dado el gran éxito que han tenido las técnicas de computación evolutiva, en concreto los Algoritmos Genéticos (en adelante, AA.GG.) y la Programación Genética (en adelante, PG), y los buenos resultados que han conseguido en la resolución de problemas para los cuales las técnicas tradicionales no eran válidas.

Sin embargo, tal vez la técnica adaptativa que más éxito ha tenido en el mundo de la Inteligencia Artificial (en adelante, IA) sean las Redes de Neuronas Artificiales (en adelante, RR.NN.AA.), y han sido aplicadas con éxito a una gran cantidad de entornos distintos, desde empresariales al mundo de la medicina, dada su gran versatilidad y las enormes ventajas que ofrecen, como pueden ser abstracción, la capacidad de tolerancia a fallos, etc.

Todo este conjunto de técnicas ofrece al experto un panorama en el cual existe una gran variedad de técnicas disponibles para resolver problemas de diversa naturaleza. Sin embargo, la aplicación de estas técnicas no es trivial. A menudo es necesario tener un profundo conocimiento de cada una de las técnicas para ser capaz de aplicarlas de una forma satisfactoria.

Este es el caso, por ejemplo, de las RR.NN.AA., en las cuales tienen un proceso de desarrollo que está fuertemente marcado por la intervención humana, y que está basado en la experiencia previa del experto que las utilice. Esto convierte este desarrollo en un proceso lento y repetitivo.

Desde el punto de vista del experto, posiblemente no muy familiarizado con los conceptos relacionados con el mundo de la IA, lo deseable es tener un sistema que funcione de forma totalmente automatizada, en el cual se puedan obtener resultados con una mínima participación, o incluso sin ella, lo cual sería el mejor caso.

1.2 Objetivos

El principal objetivo de esta Tesis es el desarrollo de un sistema que permita desarrollar RR.NN.AA. de una forma automatizada, con el objetivo de resolver problemas, sin participación del experto humano en el desarrollo de las mismas.

Las RR.NN.AA. son sistemas que, tomando como base el funcionamiento del cerebro, son capaces de aprender a partir de un conjunto de patrones. Las RR.NN.AA. constan de una serie de nodos (neuronas) unidos por una serie de conexiones, cada una de ellas con un valor de peso asociado. En cada nodo se computa el valor de su salida a partir de sus entradas, y este valor de salida se va propagando a través de las conexiones a los nodos siguientes. Generalmente el número de conexiones es muy alto, con lo que

la red resultante es muy compleja. El sistema que se genera de esta forma tiene ciertas características que lo convierten en una herramienta muy útil y potente, con capacidad de generalización y tolerancia a ruido. Es decir, que el sistema creado puede utilizarse con datos que no ha visto con anterioridad y que pueden tener ruido.

Para desarrollar las redes, existen varios algoritmos. En general, la mayoría de estos algoritmos se basan en ajustar los parámetros de una red (es decir, los valores de los pesos de las conexiones) que ha sido previamente diseñada, con lo cual este proceso de entrenamiento está claramente influido por las limitaciones impuestas en el diseño de la red. Habitualmente, este proceso de diseño previo al entrenamiento propiamente dicho, se realiza de forma manual; es decir, no existe ningún método que determine qué arquitectura de red va a ser necesaria para resolver un problema determinado y es necesario probar varias arquitecturas distintas, entrenando cada arquitectura por separado, para finalmente escoger una de ellas, la que mejores resultados haya ofrecido.

El proceso de entrenamiento en sí no tiene menos problemas. Los algoritmos tradicionales tenían el problema de que se estancaban frecuentemente y el aprendizaje se paraba prematuramente. Recientemente, los AA.GG. también han sido aplicados en el entrenamiento de las RR.NN.AA. de una forma eficaz y eficiente. Sin embargo, sigue presente el problema del diseño de la red, lo cual obliga a utilizar los AA.GG. con varias arquitecturas distintas.

Como se describe más adelante, el proceso de desarrollo de RR.NN.AA. es un proceso que está marcado por la excesiva participación del experto humano, así como por su experiencia, que es, a la postre, quien le guía en el diseño de la topología de la red. Todo esto convierte a la creación de RR.NN.AA. en un proceso muy dependiente del experto.

Para solucionar este problema, en esta Tesis se estudia el uso de una técnica de CE para el desarrollo automatizado de RR.NN.AA. En concreto, la técnica a utilizar es la PG. Esta técnica puede considerarse como una técnica de búsqueda que permite la resolución de problemas por medio de la inducción automática de programas que describen una relación entrada/salida, y supone una evolución de los AA.GG. tradicionales, que precisamente tienen la teoría de la evolución de Darwin como modelo de funcionamiento. Esta técnica ofrece diversas ventajas, como pueden ser:

- Implementa un proceso de búsqueda estocástico, realizando búsquedas simultáneas en varias regiones del espacio de estados, con continuas

exploraciones en nuevos sectores. Se revela así como un eficiente algoritmo de búsqueda, apto para aquellos problemas en los que su espacio de estados tenga muchos máximos locales, valles, etc. en los que los otros algoritmos, como puedan ser de minimización de gradiente, no llegarían a la solución.

- Al contrario que en los algoritmos genéticos tradicionales, la codificación de cada solución particular (que suele tomar el nombre de cromosoma) se realiza en términos del propio problema. En los algoritmos genéticos tradicionales era necesario realizar una tarea de codificación/decodificación generalmente en cadenas de bits, pero en la programación genética esto ya no es necesario, puesto que la solución buscada va a ser encontrada en forma de algoritmo, de una forma similar a la que entienden los compiladores: en forma de árbol.
- Al estar trabajando con este tipo de estructuras y codificación, se minimiza y facilita el análisis necesario para la codificación, puesto que ahora se va a realizar de una forma próxima al problema a tratar, lo que también proporciona una gran versatilidad y variedad de problemas que va a poder resolver.
- Son algoritmos inherentemente paralelos, puesto que se basan en realizar cálculos de forma masiva e independiente (evaluar muchas soluciones distintas e independientes).
- Permite la obtención de soluciones de forma jerárquica.

Por estas razones, la PG parece ser la técnica ideal para este determinado propósito. Sin embargo, su aplicación tiene un grave inconveniente. Este nace del hecho de que la forma nativa de codificación de problemas que tiene la PG es en forma de árboles, y las RR.NN.AA. tienen forma de grafos, usualmente con un nivel de conectividad muy alto, con lo que la representación de las RR.NN.AA. como individuos dentro de la PG no es un proceso trivial. En general, la compleja estructura de las RR.NN.AA. ha impedido que exista un método para lograr una representación de su conocimiento de forma automatizada.

En esta Tesis se plantean dos alternativas a este problema: en primer lugar, el uso de operadores especiales que permitan simular una estructura de grafo dentro de un árbol, y en segundo lugar, la modificación de los algoritmos propios de la PG para que la forma de representación de soluciones sea con grafos, no con árboles, con lo que se conseguiría representar RR.NN.AA. de una forma directa.

En esta Tesis, además, se plantea el uso de una arquitectura híbrida, en la cual, si bien se utiliza PG para hacer evolucionar las redes, estas son sometidas a un proceso de optimización de los valores de los pesos de las conexiones por medio de un AG. Se realiza un estudio comparativo de los resultados obtenidos con y sin este proceso de optimización, que a su vez es aplicado al uso de árboles y grafos como métodos de representación de RR.NN.AA.

1.3 Estructuración de la Tesis

La presente Tesis está estructurada en varios capítulos de forma que partiendo de una serie de descripciones sobre los métodos y técnicas existentes actualmente, se analizan las diversas formas de conseguir uno por uno los diferentes objetivos que se acaban de presentar.

En el capítulo 2 se realiza una descripción de las técnicas de IA que se utilizan en este trabajo. Respecto a las RR.NN.AA., se verá qué son, tanto desde la perspectiva biológica como desde la perspectiva artificial de sus componentes y la interrelación entre ellos. Se verá cual es uno de los procesos más comunes de entrenamiento de estas RR.NN.AA y los tipos, así como las características específicas de las redes recurrentes. Con ello, se pretende identificar los datos que son necesarios sobre la complejidad de las diferentes estructuras de las RR.NN.AA. Se aportarán nociones sobre AA.GG., entrando en el campo de la CE, su funcionamiento y cómo, consecuencia de una evolución de esta técnica, surgió la PG, así como el funcionamiento de la misma.

El capítulo 3 analiza los trabajos existentes en los campos que atañen a esta Tesis: desarrollo de RR.NN.AA. mediante técnicas de CE y algoritmos de PG cuya codificación se basa en el uso de grafos. Una vez realizada esta descripción, el capítulo 4 expone la hipótesis de trabajo que se ha tomado para la realización de esta Tesis.

Posteriormente, el capítulo 5 ya realiza una completa descripción del modelo que se propone en esta Tesis. El capítulo 6 muestra un completo conjunto de experimentos realizado sobre diversos problemas conocidos, cuyo objetivo es, además de comprobar el funcionamiento del sistema, obtener un conjunto de parámetros que ofrezca buenos resultados sobre problemas de distinta complejidad.

El capítulo 7, por su parte, realiza una comparativa del método propuesto en este trabajo con algunos de los métodos más importantes de generación y entrenamiento de RR.NN.AA. mediante técnicas de CE.

Finalmente, los capítulos 8, 9 y 10 exponen las conclusiones, trabajos futuros y referencias respectivamente, con lo que se concluye este trabajo.

CAPÍTULO

2 FUNDAMENTOS

Para poder diseñar y realizar un sistema que genere RR.NN.AA es necesario conocer la técnica con un cierto grado de profundidad. Para ello, se van a exponer los conceptos fundamentales de las neuronas artificiales y su interconexión formando la RNA. Con este fin se explicarán las arquitecturas de RR.NN.AA. más frecuentes, los diferentes tipos de neuronas o elementos de proceso más habituales, el proceso de entrenamiento y la puesta en funcionamiento de las mismas, que son conceptos básicos en el mundo de las RR.NN.AA.

Igualmente, y dado que se usará la PG como técnica principal para el desarrollo de RR.NN.AA., se realizará una extensa descripción de la misma, así como de los AA.GG., que también serán utilizados en este trabajo. Para poder usar estas técnicas, es preciso conocer cómo funcionan los métodos evolutivos, los operadores genéticos y cómo se pueden codificar los problemas en forma de individuos y material genético, y cómo actúa el proceso de evolución natural en la búsqueda de soluciones.

2.1 Redes de Neuronas Artificiales

E1 primer modelo de neurona artificial fue propuesto por McCulloch y Pitts [McCulloch 1943] donde modelizaban una estructura y un funcionamiento simplificado de las neuronas del cerebro, considerándolas como dispositivos con *"m"* entradas, una única salida y solo dos estados posibles: activa o inactiva.

Una RNA era, en ese planteamiento inicial, una colección de neuronas de McCulloch y Pitts, todas sincronizadas, donde las salidas de unas neuronas estaban conectadas a las entradas de otras. Algunos de los planteamientos de McCulloch y Pitts

se han mantenido desde 1943 sin modificaciones, otros por el contrario han ido evolucionando, pero todas las formalizaciones matemáticas que se han realizado desde entonces, sobre las RR.NN.AA, aún sin pretender ser una modelización exacta de las redes de neuronas biológicas, sí han resultado un punto de partida útil para el estudio de las mismas.

Una de las definiciones que se estima más certera de RNA es la siguiente: "Las redes neuronales son conjuntos de elementos de cálculo simples, usualmente adaptativos, interconectados masivamente en paralelo y con una organización jerárquica que les permite interactuar con algún sistema del mismo modo que lo hace el sistema nervioso biológico" [Kohonen 1988]. Su aprendizaje adaptativo, auto-organización, tolerancia a fallos, operación en tiempo real y fácil inserción dentro de la tecnología existente, han hecho que su utilización se haya extendido en áreas como la biológica, financiera, industrial, medio ambiental, militar, salud, etc. [Hilera 1995]. Están funcionando en aplicaciones que incluyen identificación de procesos [González 98], detección de fallos en sistemas de control [Aldrich 1995], modelación de dinámicas no lineales [Meert 1998] [Wang 1998], control de sistemas no lineales [Bloch 1997][Levin 1993] y optimización de procesos [Oller 1998][Altissimi 1998][Aguiar 1998].

En general, se puede encontrar que una RNA se suele caracterizar por tres partes fundamentales: la topología de la red, la regla de aprendizaje y el tipo de entrenamiento.

En este afán de emular el cerebro, esto es, simular tanto su estructura como su funcionamiento, se han desarrollado numerosos modelos de RNA [Freeman 1993], entre los que se pueden mencionar: Perceptron (1957), Adaline y Madaline (1960), Avalancha (1967), Retropropagación (1974), Hopfield y SOM (1980), ART (1986), etc. De los modelos anteriores se puede apreciar que esta idea tiene ya 50 años, sin embargo, salvo casos puntuales como en Madaline, sólo en las últimas décadas se ha desarrollado la tecnología que permita su aplicación de manera eficiente. Aunque se han propuesto una gran cantidad de modelos, todos usan una estructura en red en la cual los nodos o neuronas son procesos numéricos que involucran estados de otros nodos según sus uniones.

Las RR.NN.AA se han hecho muy populares, en gran medida, debido a la facilidad en su uso (ver figura 2.1) e implementación y la habilidad para aproximar cualquier función matemática [Haykin 1999].

Las RR.NN.AA., con su marcada habilidad para obtener resultados de datos complicados e imprecisos, pueden utilizarse para extraer patrones y detectar tramas que son muy difíciles de apreciar por humanos u otras técnicas computacionales. Por ejemplo, una red neuronal entrenada puede usarse como un "experto" para categorizar la información que se ha dado para su análisis. Este experto puede usarse para proveer proyecciones de nuevas situaciones.

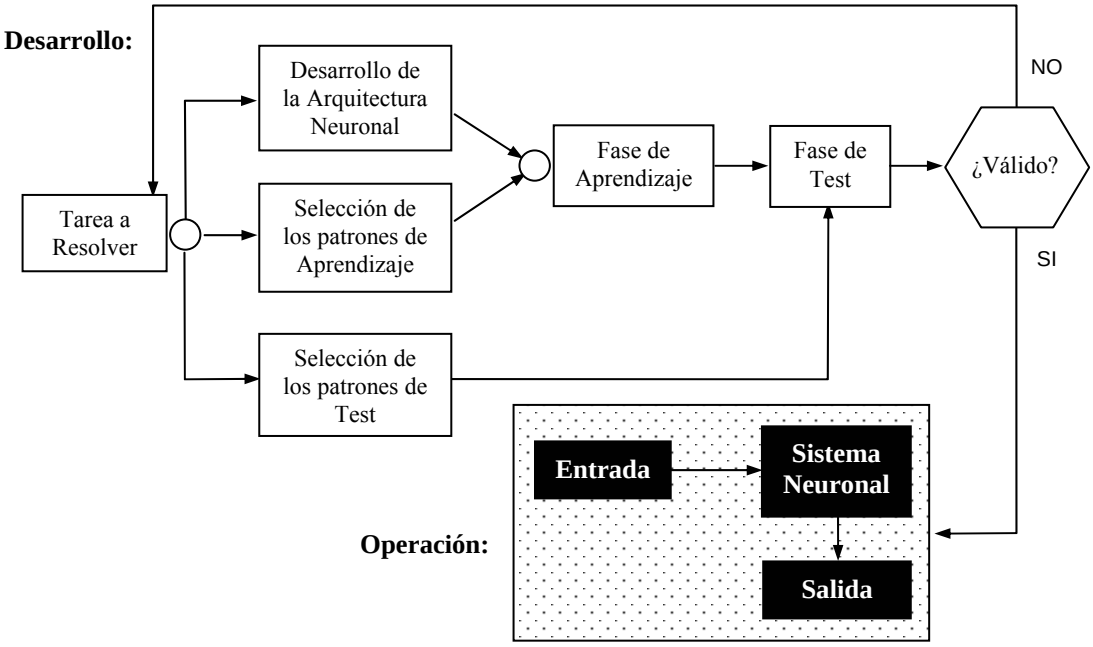


Figura 2.1. *Modo de trabajo con Redes de Neuronas Artificiales*

Las ventajas más reseñables de las RNA son las siguientes:

1. **Aprendizaje adaptativo**. Capacidad de aprender a realizar tareas basadas en un entrenamiento o una experiencia inicial, e ir adelantando su respuesta a las circunstancias cambiantes de su entorno.

2. **Autoorganización**. Una red neuronal puede crear su propia organización o representación de la información que recibe durante la etapa de aprendizaje.

3. **Tolerancia a fallos gracias a poseer la información distribuida o vía información redundante**. La destrucción parcial de una red puede conducir a una degradación de su estructura; sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo daños considerables.

4. **Capacidad de generalización y tolerancia a ruido**. Ante la entrada de datos nuevos, o de datos con ruido, es capaz de producir resultados

coherentes de acuerdo con la naturaleza del problema para el cual han sido entrenadas.

5. **Operación en tiempo real**. El cómputo neuronal puede realizarse en paralelo, bien vía software o mediante máquinas especiales para obtener esta ventaja (hardware conexionista o masivamente paralelo), lo que permite obtener la respuesta de forma inmediata.

Estas propiedades hacen atractivas a las RNA para determinadas aplicaciones comerciales, militares e industriales [Hernández 1993] y, por supuesto, para múltiples facetas dentro del mundo de la investigación [Rabuñal 2003].

2.1.1 Fundamentos y Conceptos Básicos de las RR.NN.AA.

La neurona artificial o elemento formal está conceptualmente inspirada en la neurona biológica. Esto es, los investigadores están en su inmensa mayoría pensando en la organización cerebral cuando consideran configuraciones y algoritmos de RNA.

Se considera una neurona como un elemento formal o módulo o unidad básica de la red que recibe información de otros módulos o del entorno; la integra, la computa y emite una única salida que se va a transmitir idéntica a múltiples neuronas posteriores [Wasserman 89].

En las RR.NN.AA existe un peso o fuerza sináptica que va a ser un valor numérico que pondera las señales que se reciben por sus entradas. El peso de las conexiones se denota de la siguiente manera:

$\mathbf{W_{ij}}$ *= Peso de la conexión entre la neurona j (que emite) y la neurona i (que recibe).*

A través de estas conexiones, cada una con un valor de peso asociado, llegan las entradas a la neurona. A partir de dichas entradas la neurona emite una salida, que viene dada por 3 funciones, que se aplican de forma consecutiva:

- Función de propagación (también conocida como función de excitación). Consiste, por regla general, en el sumatorio de cada entrada multiplicada por el peso de su conexión. Si el peso es positivo, la conexión se denomina excitatoria; si es negativo, se denomina inhibitoria. A partir de este cálculo se obtiene un valor que se denomina "NET", y que, como se ha descrito, responde a la siguiente ecuación, para la neurona i, siendo O_j la salida de la neurona j:

$$NET_i = \sum_{j=0}^{N-1} \left[W_{ij} * O_j \right]$$

- Función de activación. Recibe el valor NET_i y calcula el estado de activación de la neurona a partir de ese valor y del estado de activación previo. Esta función puede no existir, lo cual es muy común, siendo por tanto la salida de esta función en este caso el valor NET_i hallado anteriormente. Sin embargo, en el caso general el cómputo de la misma es el siguiente, siendo $A_i(t)$ el estado de activación de la neurona i en el instante t:

$$A_i(t) = FA\left(A_i(t\text{-}1), NET_i(t)\right)$$

  Existen dos modelos de función de activación:

  - ✓ Modelos acotados: El valor de la activación de la neurona puede ser cualquiera dentro de un rango continuo de valores.
  - ✓ Modelos No acotados: No existe ningún límite para los valores de activación.

- Función de transferencia. Toma como entrada el valor devuelto por la función de activación. Se utiliza para acotar la salida de la neurona.

En la figura 2.2 puede verse un resumen de la estructura general de una neurona artificial.

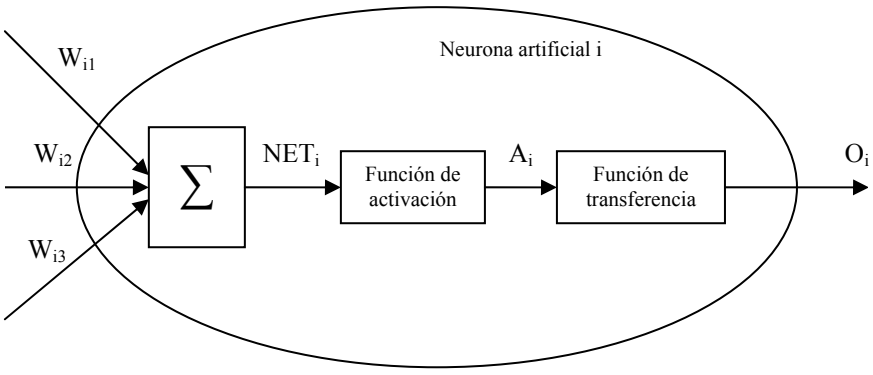


Figura 2.2. *Estructura de una neurona artificial*

La linealidad de las funciones que definen a los elementos de la red es quizás lo que va a proporcionar la característica más definitoria. De esta forma, se pueden clasificar las neuronas en lineales y no lineales.

2.1.1.1 Neuronas lineales

Una neurona es lineal cuando tanto su función de activación como de transferencia son lineales, por lo que la composición de ambas funciones da lugar a otra función lineal. En estas neuronas la salida es linealmente dependiente (es decir, proporcional) de sus entradas ponderadas cada una de ellas por el peso asociado a las sinapsis por las que le llega las entradas señal. La respuesta de las neuronas lineales no está acotada y puede tomar cualquier valor.

Está demostrado que los cómputos que realizan redes con más de dos capas con unidades lineales se pueden realizar también con redes de dos capas, con lo que se hace superfluo construir redes de más de dos capas si las funciones utilizadas son siempre lineales.

Esto conlleva a ciertos problemas como la falta de persistencia en las respuestas, de modo que cambios muy pequeños en las entradas pueden producir fluctuaciones bastante grandes en las respuestas, o la falta de adecuación simultánea a señales grandes y pequeñas, pues es imposible que con neuronas lineales la respuesta de una neurona se adapte tanto a señales grandes como a pequeñas. Este último efecto se produce porque si las funciones utilizadas amplifican mucho las señales de entrada, entonces señales de entrada de pequeña intensidad no se perderán sino que provocarán una respuesta de la neurona (señales de entrada moderada provocarán respuestas elevadísimas). En el caso de que dichas funciones amplifiquen poco las señales de entrada (si produce salidas moderadas ante valores medios), entonces las señales de entrada débiles producirán señales poco significativas.

2.1.1.2 Neuronas no lineales

En estas neuronas o bien la función de activación o bien la función de transferencia (o ambas) es una función no lineal, dando lugar a que la respuesta de la neurona no sea función lineal de sus entradas.

Este tipo de neuronas van a producir respuestas acotadas, desapareciendo los problemas de fluctuación y la falta de adecuación a señales pequeñas y grandes.

Como funciones no lineales se destacan las siguientes:

- Umbral:

$$OUT = \begin{cases} 1 \text{ si } A_i > 0 \\ -1 \text{ si } A_i < 0 \end{cases}$$

- Sigmoide:

$$OUT = \frac{1}{1+e^{-A_i}}$$

- Hiperbólica tangente:

$$OUT = \frac{1-e^{-A_i}}{1+e^{-A_i}}$$

Este es el caso más general de neurona artificial. La salida, por lo tanto, está acotada, y suele tomar valores entre 0 y 1, o entre -1 y 1, lo cual condiciona fuertemente el rango de los datos que se utilicen en las entradas, así como la interpretación de los mismos, y de las salidas.

2.1.2 Flujos de datos en las RR.NN.AA.

2.1.2.1 Redes alimentadas hacia adelante

Son aquellas en las que, como su nombre indica, la información se mueve en un único sentido, desde la entrada hacia la salida. Estas redes están clásicamente organizadas en "capas". Cada capa agrupa a un conjunto de neuronas que reciben informaciones de las neuronas de la capa anterior y emiten salidas hacia las neuronas de la capa siguiente. Entre las neuronas de una misma capa no hay conexiones.

En este tipo de redes existe al menos una capa de entrada, formada por las neuronas que reciben las señales de entrada a la red y una capa de salida, formada por una o más neuronas que emiten la respuesta de la red al exterior. Entre la capa de entrada y la de salida existen una o más capas intermedias.

En redes así construidas es evidente que la información sólo puede moverse en un sentido: desde la capa de entrada hasta la capa de salida, atravesando todas y cada una de las capas intermedias una sola vez.

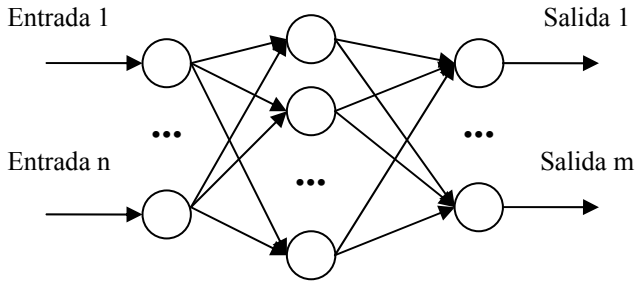


Figura 2.3. *RNA alimentada hacia delante.*

El procesamiento de la información en estas redes se realiza en un tiempo predeterminado e igual para todas las posibles configuraciones de señales de entrada.

El hecho de que no haya conexión entre las neuronas de una misma capa hace que no haya tiempos de espera en los que las neuronas estén interactuando unas sobre otras hasta que toda la capa adquiera un estado estable. Se trata por tanto de redes rápidas en sus cálculos.

2.1.2.2 Redes con retropropagación total o parcial

En este tipo de redes los elementos pueden enviar estímulos a neuronas de capas anteriores, de su propia capa o a ellos mismos, por lo que desaparece el concepto de agrupamiento de las neuronas en capas. Cada neurona puede estar conectada a todas las demás; de este modo, cuando se recibe información de entrada a la red, cada neurona tendrá que calcular y recalcular su estado varias veces, hasta que todas las neuronas de la red alcancen un estado estable. Un estado estable es aquel en el que no ocurren cambios en la salida de ninguna neurona. No habiendo cambios en las salidas, las entradas de todas las neuronas serán también constantes, por lo que no tendrán que modificar su estado de activación ni su respuesta, manteniéndose así un estado global estable [Pearlmutter 1990].

En este tipo de redes no hay forma de saber cuánto se tardará en alcanzar un estado estable. Además, unos estímulos de entrada provocarán que se alcance el estado estable en menos tiempo que otros.

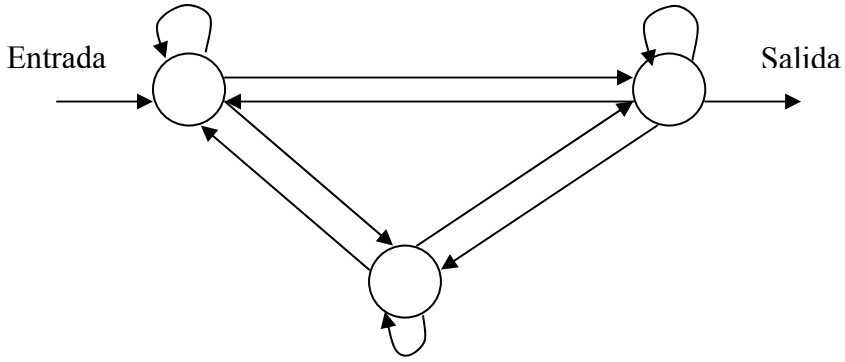


Figura 2.4. *RNA con retroalimentación*

Las redes retroalimentadas emulan más fielmente la estructura del cerebro humano, en donde los fenómenos de retroalimentación son fundamentales.

2.1.3 Manejo de información en las RR.NN.AA.

En esta sección se presentan los tipos de información que maneja una RNA, así como los métodos para alimentarla con dicha información; en otras palabras, se verá cómo se le presenta a la RNA el problema que debe resolver. También se verán los problemas típicos con los que las redes se enfrentan.

La información de entrada en números tiene que ser adecuada para las funciones de activación y transferencia de la RNA y se deben distribuir esas informaciones entre las neuronas de la capa de entrada de forma adecuada.

Por tanto, a la hora de diseñar una RNA es necesario realizar un análisis tanto de la información de entrada que hay que suministrarle para presentarle el problema, como de la información de salida que la RNA proporcionará como solución a dicho problema.

Si los datos de entrada, es decir, la información del problema que se desea que la RNA resuelva, no se pueden representar mediante valores dicotómicos, se tendrán que modificar las funciones de la red para hacerla así capaz de trabajar con valores con los que se puedan representar los datos.

Si, por ejemplo, se trabaja con información incierta, en donde sólo se puede dar un valor de probabilidad de que un determinado elemento pertenezca a un conjunto, se necesitará que la función sea continua para poder expresar valores de probabilidad y el rango de la salida de la función de transferencia, y por tanto la salida de cada neurona, irá de 0 a 1.

Si los valores de salidas son continuos y los datos también, se tendrá que volver a normalizar los datos de entrada para hacerlos corresponder con el rango de valores que se pueden tomar. Es decir, es necesario reajustar la escala de los datos para que se adapte a la escala de las neuronas (generalmente entre 0 y 1 o entre -1 y 1). Por ejemplo, un dato de entrada es una longitud en metros (desde 0 a 1.000 m.) y se usan valores de -1 a 1, se tendrá que establecer una correspondencia en la que a 0 m. le corresponda -1, a 1.000 m. le corresponda +1, a 500 m. le corresponda 0, etc.

2.1.4 Funcionamiento de las RR.NN.AA.

El funcionamiento básico de las RR.NN.AA. abarca dos grandes etapas. La primera comprende todas las fases de creación y desarrollo de la misma, y posteriormente viene la fase de funcionamiento real o fase de ejecución, durante la cual la RNA ya es operativa y, tanto su estructura interna como los valores de los pesos de las conexiones no vuelven a ser modificados, a pesar de la existencia de varios tipos de RR.NN.AA. en los que esto ocurre. Durante esta fase se usa la RNA como si se tratara de cualquier otro programa informático convencional, y es cuando se utiliza de forma efectiva para resolver los problemas para los que ha sido diseñada.

2.1.4.1 Creación y desarrollo de RR.NN.AA.

Esta etapa comprende todas las fases necesarias para el desarrollo de una RNA, y comprende las siguientes etapas:

- Diseño de la arquitectura.
- Entrenamiento de la red.
- Validación de la red.
- Test de la red.

2.1.4.1.1 Diseño de la topología

Esta etapa comprende determinar el número de neuronas que tendrá la red, así como su disposición en capas y la conectividad entre las mismas.

A partir de un análisis de los datos del problema, se determinan cuántas entradas y salidas tiene la red, así como el número de neuronas y cómo estas están distribuidas en capas e interconectadas entre sí. Esta etapa es crítica, puesto que la topología de la red

determina la capacidad de representatividad de la misma, y, por lo tanto, la cantidad de conocimiento que puede albergar. La topología de la red debe adecuarse al problema a resolver, y la no existencia de técnicas que realicen esta función hace que haya que recurrir a la experiencia y a la técnica de "ensayo y error", probando varias topologías distintas, para finalmente conseguir una que se adapte de forma satisfactoria al problema.

Esta etapa también alberga el determinar las funciones de activación y transferencia que se usarán (habitualmente, las mismas para todas las neuronas).

2.1.4.1.2 Entrenamiento y validación

Una vez diseñada la arquitectura de la red (capas y número de neuronas por capa) y las funciones que la regirán, se tiene que proceder a entrenar a la red para que "aprenda" el comportamiento que debe tener; es decir, para que aprenda a dar la respuesta adecuada a la configuración de estímulos o patrón de entrada que se le presente [Pazos 1996]. Esto se realiza determinando los valores de los pesos de las conexiones de la red, y para ello existen diferentes algoritmos, el más común de ellos es el de *backpropagation* (BP) o retropropagación del error [Rumelhart 1986], que se aplica a redes alimentadas hacia adelante, dispuestas en varias capas, y con una conectividad total entre las neuronas de una capa y las de la capa siguiente.

El proceso de validación de una red suele realizarse de forma conjunta al entrenamiento, y controla el mismo para impedir que la red caiga en alguno de los problemas habituales del entrenamiento. La validación, por tanto, asegura que la red ha sido entrenada de forma satisfactoria.

Existen dos grandes tipos de aprendizaje: supervisado y no supervisado. El aprendizaje supervisado se caracteriza por la existencia de un "maestro" o "supervisor" que le indica a la red, durante el proceso de entrenamiento, cuál es la salida que debe de ofrecer, y, por lo tanto, también le indica qué error está cometiendo. Este "supervisor" no es más que un juego de ensayo, formado por parejas "patrón de estímulos - respuesta correcta". Cada pareja se denomina hecho, y en el juego de ensayo debe estar representada equilibradamente toda la información que la RNA necesite aprender.

En este tipo de aprendizaje, el entrenamiento consiste en presentarle a la red repetitivamente patrones de estímulos de entrada pertenecientes al juego de ensayo, y a partir de las salidas que ofrece la red calcular un valor de error con respecto a las

respuestas que se desean, es decir, se comparan las respuestas de la red con las salidas deseada del juego de ensayo. En virtud de esa comparación, se reajustan los pesos de las conexiones, para lo cual existen varios algoritmos, siendo el más común el mencionado anteriormente, BP. El reajuste de los pesos de las conexiones está orientado a que, ante el patrón de entrada, la red se acerque cada vez más a la respuesta correcta.

Cuando ante un patrón de entrada la red ya responde bien, se pasa al siguiente patrón del juego de ensayo y se procede de la misma manera. Cuando se termina con el último patrón del juego de ensayo, se tiene que volver a empezar con el primero, ya que los pesos se han seguido modificando.

En casos sencillos, al cabo de unos pocos pasos de entrenamiento completos, con todos los elementos del juego de ensayo, los pesos de conexiones de todas las neuronas se estabilizan en torno a unos valores óptimos. Se dice entonces que el algoritmo de aprendizaje converge. Es decir, después de sucesivas presentaciones de todos los patrones estimulares del juego de ensayo, la RNA responderá correctamente a todos ellos y se puede considerar entrenada y dar por terminada la fase de aprendizaje.

El aprendizaje no supervisado se caracteriza por la no existencia de ese elemento "supervisor", es decir, para un conjunto de entradas no existen unas salidas deseadas. En estos casos es cuando se desea que la red encuentre relaciones y dependencias entre los datos de entrada. Ejemplos de estas redes son los mapas autoorganizativos (*self-organizing maps*, SOM) [Kohonen 1988], cuyo objetivo es, a partir de un conjunto de datos, realizar una tarea de *clusterización* de los mismos.

Por último, existen otro tipo de redes que no son entrenadas, sino construidas; es decir, los valores de los pesos de las conexiones no se determina ante la presencia de un conjunto de entrenamiento, sino que estos valores son fijados de antemano y no son modificados. Este es el caso de las redes Hopfield, que son utilizadas como memorias autoasociativas, con aplicaciones en campos como la percepción o el reconocimiento de imágenes [Hopfield 1982].

2.1.4.1.3 Test

Finalmente, cuando se tiene una red entrenada y validada, se realiza un test de la misma, en unas condiciones que no han sido presentadas durante el entrenamiento, para evaluar su capacidad de generalización y su comportamiento en casos que la red no ha

visto, con lo que se tiene una medida real de cómo se va a comportar la red con el problema para el cual ha sido creada.

2.1.4.2 Selección de los elementos del juego de ensayo

En vista de la gran cantidad de información que se le puede proporcionar a una RNA, se ha de buscar un criterio de selección para crear el juego de ensayo.

En el juego de ensayo debe de haber suficientes hechos: parejas "patrones de estímulos - respuesta correcta". Además, los hechos del juego de ensayo deberán cubrir ampliamente la totalidad de las características a los que la RNA debe de enfrentarse.

No se debe desechar alegremente información por pensar que es irrelevante. A menudo, las redes de neuronas pueden encontrar asociaciones que nunca se hubiese supuesto que existiesen. Se debe, por el contrario, entrenar a la red con cualquier pieza de información disponible y, simultáneamente, entrenar otra RNA con los hechos mínimos que se consideran imprescindibles, para luego cotejar cual de las dos redes se comporta mejor. Puede resultar una curiosa experiencia el descubrir cuales son los hechos realmente importantes.

Por otra parte, a una red evaluadora es importante mostrarle tanto los patrones de entrada que llevan a evaluaciones positivas, como los patrones de entrada que llevan a evaluaciones negativas. Es decir, el entrenamiento de la RNA debe incluir situaciones que se evalúen negativamente, pues de lo contrario la red simplemente aprenderá que todo está correcto siempre. También se debe incluir en el juego de ensayo cada uno de los casos en los cuales el valor de una entrada es causa de un "mal" resultado. Por ejemplo, una RNA que evalúe la capacidad de un avión para volar debe saber que un indicador de combustible que marque vacío es definitivamente malo y que el avión no estará en disposición de volar.

Por otra parte, no se puede incluir en el juego de ensayo una colección exageradamente grande de hechos. Es necesario seleccionar aquellos hechos que reflejen claramente cada uno de los patrones a reconocer y las situaciones extremas de evaluación.

Lo ideal es preparar una colección amplia de hechos de entrenamiento que cubran todos los problemas a los que se pueda tener que enfrentar la red. Después de esa amplia

colección de hechos, se seleccionarán algunos de ellos para el juego de ensayo, teniendo cuidado de que todos los problemas queden bien representados.

2.1.4.3 Problemas que se pueden dar durante el entrenamiento

Uno de los problemas más comunes al entrenar una RNA es que la red no se entrene con precisión suficiente; es decir, que tenga un alto porcentaje de "fallos" que no se reduce por más veces que se le pase el juego de ensayo o que la red tarde mucho tiempo en entrenarse.

Cuando esto sucede, se deberá analizar el diseño de la red y del juego de ensayo. El hecho de cambiar el número de capas ocultas aumentará o disminuirá el tiempo de aprendizaje, pero probablemente no afectará en gran medida a la precisión o proporción de respuestas acertadas de la red.

Si los resultados son pobres al presentarle una colección de patrones de entrada, es preciso comprobar el juego de ensayo. Tal vez se haya olvidado representar en el juego de ensayo alguno de los problemas tipo con los que la red debe enfrentarse, o la información sea engañosa o contradictoria.

Se tendrá en cuenta que, cuando se modifica el juego de ensayo añadiendo nuevos hechos o corrigiendo alguno de los existentes, la red debe de ser entrenada de nuevo con la totalidad de los hechos y no tan sólo con los nuevos o con los corregidos.

Una forma de determinar qué hechos están produciendo problemas consiste en entrenar la red hasta que responda con un nivel de aciertos aproximadamente del 90%. Luego, al probar la red, se observan los hechos a los cuales la red responde mal. A continuación, se le deberá proporcionar a la red una mayor información similar a la de los hechos no aprendidos y se entrenará de nuevo.

Si se llega a la conclusión de que se le han dado suficientes hechos a la red; es decir, que el juego de ensayo es completo y cubre todos los problemas a resolver y ésta sigue comportándose poco satisfactoriamente, habrá que considerar la posibilidad de cambiar el modo de presentar la información de entrada a la red. Si se están utilizando entradas no distribuidas, se deberá cambiar a distribuidas lo cual requerirá una estructuración completamente nueva de la red.

Existen, por supuesto, problemas tan complejos que una red de neuronas no puede resolver en un tiempo y un espacio razonables. Si este es el caso, se podrá montar una

cadena de redes de neuronas de modo que las salidas de una red alimenten las entradas de otras. Las primeras redes deberán simplemente identificar los objetos y sus salidas se utilizarán para alimentar a otra red que realice la evaluación. El añadir más capas a una única red no produce el mismo efecto.

Otro de los problemas que pueden ocurrir, y de hecho es bastante común, es el del sobreentrenamiento. Este problema puede observarse al hacer un test de una red que acaba de ser entrenada con un alto porcentaje de acierto en dicho entrenamiento. Si el comportamiento que ofrece la red en el test no es todo lo bueno que puede hacer suponer por los resultados obtenidos en el entrenamiento, la red obtenida está sobreentrenada. Esto quiere decir que la red ha aprendido los patrones existentes en el juego de ensayo, pero realmente no ha sido capaz de abstraer y generalizar las relaciones entre los mismos, con lo que esta red no es aplicable en el mundo real, en el que se le presentarán entradas que no están presentes en el juego de ensayo. Este problema puede estar provocado por dos motivos fundamentales:

- ✓ La topología de la red tiene una complejidad muy alta. Es decir, que tiene un número de capas y/o neuronas demasiado alto, con lo que tiene un nivel de representatividad demasiado elevado para el problema que se está intentando resolver. No existe ningún método que determine la arquitectura de una red adecuada al problema a resolver, sino que esta tarea se basa en la experiencia del experto que diseñe la red.
- ✓ La red se ha entrenado durante demasiados ciclos.

Una situación muy común es entrenar una RNA con un juego de ensayo que contiene ruido, puesto que en el mundo real en muchas ocasiones no se puede eliminar totalmente el ruido de las mediciones que se realizan. Cualquier algoritmo de aprendizaje máquina (y las RR.NN.AA. no son una excepción) que sea capaz de aprender y ofrecer una precisión del 100% (o, en general, un porcentaje superior a la precisión de los patrones sin el ruido) ha aprendido los patrones de entrenamiento y el ruido de los mismos, sin llegar a discriminarlo, es decir, ha perdido la capacidad de generalización.

El proceso de entrenamiento tiende a minimizar el error que se consigue en el conjunto de entrenamiento durante el mismo, (figura 2.5 a)). Sin embargo, si durante el entrenamiento se realizan tests a la red con patrones que no se presentan durante el

entrenamiento, el error observado en estos tests no tiende siempre a minimizarse, como puede verse en la figura 2.5 b), sino que llega a un punto en el que este empieza a crecer. Este es el punto en el que la red empieza a sobreentrenarse y perder la capacidad de generalización, por eso se comporta mal en los test.

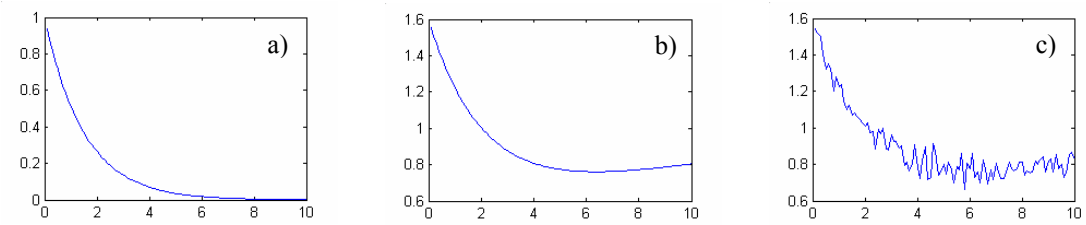


Figura 2.5. *Errores cometidos durante el entrenamiento*

Una solución a este problema podría ser el realizar test durante el entrenamiento, y detectar el punto en el que el error en el test comienza a crecer. Esto no se puede realizar con el conjunto de test, puesto que, de hacerlo así, se está condicionando la red de alguna forma a este conjunto de test, lo cual no es deseable, porque este conjunto de test está pensado para ser presentado a una red final, no a una que todavía está en el proceso de entrenamiento, y con patrones que nunca haya visto.

Para solucionar esto, se cuenta con otro conjunto de patrones, denominado conjunto de validación, que tendrá patrones distintos a los del entrenamiento y test [Bishop 1995] [Ripley 1996]. Al ser evaluada la red durante el entrenamiento con este nuevo conjunto de patrones, se ofrece una estimación de cómo se comportará la red cuando se realice el test, es decir, en su ejecución real con patrones que no ha visto. Este conjunto de validación también sirve para tomar decisiones acerca de qué red es mejor tomar, entre varias redes distintas (posiblemente con topologías distintas) ya entrenadas.

Sin embargo, este proceso no es tan sencillo. El error de validación mostrado en la figura 2.5 b) es un error teórico. En la práctica se tiene uno similar al mostrado en la figura 2.6 c), con lo que no resulta sencillo determinar cuándo hay que parar el entrenamiento. Existen diversas técnicas que, utilizando el conjunto de validación, dictan cuándo es razonablemente aconsejable parar el entrenamiento [Prechelt 1996] [Prechelt 1998]. Sin embargo, ninguna de ellas consigue determinar con exactitud cuándo es el menor momento.

2.1.4.4 Ejecución

Tras la fase de entrenamiento viene la fase de ejecución, durante la cual se le pedirá a la RNA que responda a estímulos diferentes a los presentados durante la fase de entrenamiento. Gracias a los ejemplos aprendidos del juego de ensayo, la RNA deberá ser capaz de generalizar y dar respuestas correctas ante patrones de estímulos nuevos.

En otras palabras, una vez terminado el aprendizaje, una red puede generalizar; es decir, ante entradas similares a las de su juego de ensayo, producirá salidas correctas. Hay que tener en cuenta que es muy difícil conseguir la capacidad de generalización de una RNA sin utilizar grandes cantidades de datos y que estos sean muy variados.

Para operar con una RNA entrenada, el proceso es el mismo que cuando se realizaba el entrenamiento: se le sigue suministrando información de entrada a la red, sólo que ahora no se realizará ningún ajuste en los pesos de las conexiones. La red reconocerá, evaluará y dará una respuesta.

Para conseguir el mejor rendimiento de generalización, los datos usados para el entrenamiento deben cubrir un rango de hechos suficientemente amplio. En general, cuando aumenta el tamaño y variedad del juego de ensayo disminuye la necesidad de que los datos de entrada durante la fase de trabajo normal se parezcan mucho a los patrones del juego de ensayo; es decir, la red generalizará mejor. Pero si los datos de un problema se diferencian demasiado de todos los patrones del juego de ensayo, la red tendrá dificultades para encontrar la respuesta correcta.

2.1.5 Redes recurrentes

Cuando se trabaja con patrones dinámicos; es decir, con patrones de secuencias en las que aparece el concepto tiempo, las RNA alimentadas sólo hacia adelante se encuentran bastante limitadas ya que no permiten conexiones que unan neuronas creando bucles.

En las redes recurrentes no se impone ninguna restricción en su conectividad, con lo que se gana un número mayor de pesos por neurona y por lo tanto una mayor representatividad, dado que las RNA representan la información de forma distribuida en sus pesos.

De esta forma, la principal característica de este tipo de redes es la de realimentar su salida a su entrada, evolucionando hasta un estado de equilibrio donde proporciona la

salida final de la red [Demuth 1994]. Esta característica las hace útiles cuando se quiere simular sistemas dinámicos; sin embargo, su entrenamiento es más lento que el de una red alimentada sólo hacia delante, y a la vez mucho más complejo.

El primer algoritmo de entrenamiento de este tipo de redes aparece en 1987, cuando se adapta el algoritmo de retropropagación del error de las RNA alimentadas sólo hacia delante a las redes recurrentes aplicadas a patrones estáticos ("*Recurrent Backpropagation*") y se pudieron aplicar estas redes a los mismos problemas a los que se aplicaban las multicapa alimentadas hacia delante [Pineda 1987].

Además, otros investigadores se centran en desarrollar aproximaciones del algoritmo de aprendizaje que lo hagan más práctico surgiendo el algoritmo llamado "Real-Time Recurrent Learning" o RTRL indicado para tareas de tiempo real [William 1994].

A partir de entonces, las redes recurrentes se han venido aplicando en un buen número de tareas, desde reconocimiento del habla hasta la simulación de autómatas finitos. Sin embargo, la aplicación de redes recurrentes presenta un mayor número de problemas. En el caso de patrones estáticos, una red recurrente funciona presentándole un patrón, haciendo después evolucionar la red hasta que sus salidas se estabilizan. Sin embargo, esto no está asegurado, pudiéndose dar comportamientos oscilatorios o caóticos y aunque existen estudios para establecer las condiciones para que esto no ocurra, se limitan a ciertas arquitecturas muy concretas como las Hopfield.

E1 caso de los patrones dinámicos es todavía más complicado, ya que, si se sabe poco del comportamiento de una red recurrente (por ejemplo la dificultad de estabilizarse), se sabe aún menos de su comportamiento dinámico. El poco conocimiento es empírico y no existen estudios formales ni de la red recurrente más simple: una neurona aislada con una conexión a sí misma. Tampoco existen estudios teóricos que avalen utilizar un algoritmo basado en el descenso del gradiente para tareas de tratamiento de patrones dinámicos. Un problema sencillo, como es enseñar a oscilar a una neurona aislada con realimentación, da muchos problemas del tipo de mínimos locales y hasta ahora no se conoce su justificación teórica.

Además, en redes multicapa se conoce más o menos bien qué arquitectura hay que utilizar en la mayoría de los problemas, gracias a conocimientos basados fundamentalmente en la experiencia. Sin embargo, por una parte, la variedad

arquitectónica en redes recurrentes es infinitamente superior, por lo que su diseño es más complicado y, por otra, la gran variedad de este tipo de patrones hace difícil su categorización.

2.1.5.1 Aprendizaje por épocas

El aprendizaje en este tipo de redes se realiza por épocas. La red se hace evolucionar durante un periodo de tiempo (época) en el que se le ha introducido una secuencia de ejemplo y ha debido dar la respuesta adecuada. Una vez llegado al final de una época se reinicializa la red para que el estado inicial no dependa del estado final de la época anterior y se entra en una nueva época. La variación de los pesos sólo se realiza al acabar cada época y, de la misma forma que en las multicapa, se podrá llevar a cabo "por lotes" (calcular todos los incrementos de los patrones-épocas) o de forma incremental (después de cada patrón-época), siendo ésta última, la más utilizada en redes recurrentes.

2.1.5.2 Aprendizaje en modo continuo

Aparte de "por épocas", existe el modo de operación continuo, donde en ningún momento se inicializa la red. Este tipo es el más apropiado para aplicaciones en tiempo real, donde no se sabe "a priori" la salida deseada. El aprendizaje suele ser distinto ya que, en el caso de operación por épocas, normalmente se pueden aplicar las secuencias un cierto número de veces, mientras en el caso de la operación continua, esto normalmente no se puede hacer. Además, no está nada claro el momento en que se deberían actualizar los pesos de la red.

La elección de un modo u otro dependerá de los problemas a resolver, e incluso existen técnicas que aprovechan las características de ambos modos. La principal diferencia entre ellos radica en que, en el modo "por épocas", la transición del estado final de una época al estado inicial de la siguiente se produce externamente mientras que, en el continuo, esta transición la debe aprender a realizar la propia red, por lo que se le está exigiendo más.

Otra cuestión problemática es la asignación de la salida deseada. En ciertos problemas, como la predicción de una serie temporal o la simulación de un autómata finito, la salida deseada se exige en todo momento. Sin embargo, en otros problemas como los de reconocimiento, no se puede establecer cual es la salida deseada en cada

instante. Esto plantea preguntas como: ¿Qué salida se exige a la red cuando se le introduce el patrón?, ¿en qué momento se exige la salida deseada?, ¿al acabar de introducir el patrón o cuanto tiempo después?, ¿la salida será puntual o continua?, etc.

Las respuestas dependen siempre del problema a tratar; por ejemplo, podría emplearse una salida neutra 0 durante la entrada del patrón si se emplea un intervalo [-1,1]. En el caso de las otras preguntas, se relacionan de forma que, si se ha escogido no forzar un comportamiento de la salida mientras se está realizando la entrada, no es conveniente utilizar una salida puntual ya que será difícil determinar si la salida en un determinado momento corresponde a un estado transitorio o al reconocimiento de una secuencia.

Otro problema que puede surgir está relacionado con la forma de la salida. Si la salida es de tipo escalón donde debe pasar de un punto extremo a otro (por ejemplo, de -1 a 1, o de 0 a 1) en un paso de tiempo, es bastante normal que la red no sea capaz de realizar una transición tan brusca. En este caso es necesario proponer una salida suavizada o bien dejar un intervalo de transición entre los dos estados sin salida deseada.

2.1.6 El proceso de activación atenuada en el tiempo

Dentro del proceso de evaluación de una RNA; es decir, dadas unas entradas a la red calcular en función de su arquitectura y de los valores de los pesos de las conexiones los valores de las salidas que produce, existe un concepto más incorporado a dicha tarea, éste es la activación atenuada en el tiempo [Pazos 1999].

En los elementos de proceso tradicionales, sobre la función de propagación, que pondera los pesos con las activaciones de las neuronas de la capa anterior, se le aplica una función de activación y otra de transferencia, siendo las más utilizadas la exponencial y la hiperbólica tangente (además se pueden utilizar la lineal y la umbral). La salida de la neurona se propaga hacia otras neuronas, y aquellas a las que llega esa salida tendrán como entrada el producto del peso de la conexión por el valor de la activación.

Esta forma de propagar la salida de las neuronas funciona de una forma muy puntual en el tiempo, sin dar la idea de continuidad que se aprecia en el modelo natural donde la señal fluye desde un punto máximo (potencial de acción) hasta llegar progresivamente a “cero” (potencial de reposo). Una opción para emular este proceso es

añadir a la arquitectura clásica de elemento de proceso la función del potencial de acción, que se puede aproximar como una recta que va desde el nivel de salida de la función de activación hasta el nivel cero, como se puede ver en la figura 2.6.

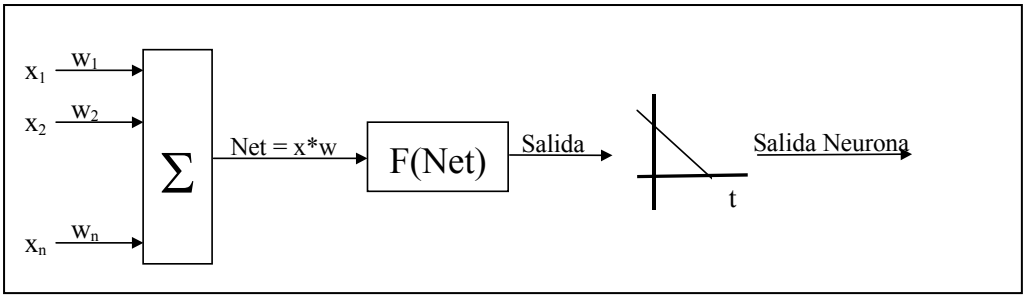


Figura 2.6. *Representación de la Conexión Atenuada en el Tiempo*

Este modelo de RNA sólo cambia en la concepción de las conexiones entre neuronas que pasan de ser fijas (un valor numérico) a ser representadas por una función (una recta con pendiente negativa) caracterizada por el valor de la pendiente.

El funcionamiento de la neurona hay que expresarlo ahora en función del tiempo. Es decir, ahora, la activación que produce en una neurona la conjunción de una serie de entradas en un cierto momento del tiempo, no afecta solo a las neuronas conectadas a su salida en ese instante sino también en *n* instantes posteriores (figura 2.7). El valor de *n* depende de la pendiente de la recta que tenga esa conexión.

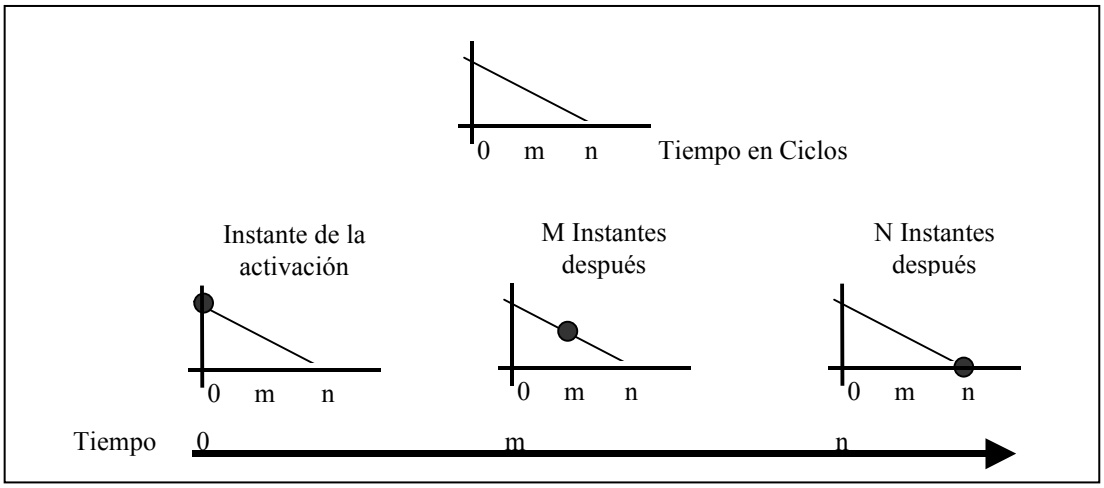


Figura 2.7. *Funcionamiento de la Activación de la Neurona*

La recta que emula la activación atenuada de las neuronas naturales sigue la fórmula clásica y=m*x+b. En esta fórmula ***y*** representa el valor de salida de la neurona que reciben las neuronas siguientes. La letra ***b*** representa la salida de la neurona. La letra ***x*** representa el tiempo que pasa desde la activación y la letra ***m*** es la pendiente de la recta que es siempre negativa. En el momento en que se produce la activación, el

tiempo es 0 por tanto la salida ***y* = *b***, a partir de ahí, y dependiendo de la pendiente, la salida **y** decrece hasta 0 en un tiempo ***x* = *t***.

2.2 Computación Evolutiva

La computación evolutiva se basa en realizar modelos de ciertas características de la naturaleza, fundamentalmente de la capacidad que tienen los seres vivos para adaptarse a su ambiente, lo que ya había sido tomado como base por Darwin para realizar su teoría de la evolución según el principio de selección natural en 1859 [Darwin 1859]. Las técnicas de CE se basan en este principio para desarrollar algoritmos que son capaces de adaptarse al problema que se pretende solucionar.

2.2.1 Algoritmos Genéticos

Un investigador de la Universidad de Michigan llamado John Holland era consciente de la importancia de la selección natural, y a finales de los años 60 desarrolló una técnica que permitió incorporarla en un programa de ordenador. Su objetivo era lograr que las máquinas aprendieran por sí mismas. A la técnica que inventó Holland se le llamó originalmente "planes reproductivos", pero se hizo popular bajo el nombre "algoritmo genético" (AG) tras la publicación de su libro "Adaptation in Natural and Artificial Systems" en 1975 [Holland 1975].

Un AG es un algoritmo de búsqueda basado en la observación de que la reproducción sexual y el principio de supervivencia del más apto permiten a las especies biológicas adaptarse a su ambiente y competir por los recursos.

Una definición bastante completa de un AG es la propuesta por John Koza: "*Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud*" [Koza 1992].

Más formalmente, y siguiendo la definición dada por Goldberg, *"Los Algoritmos Genéticos son algoritmos de búsqueda basados en la mecánica de selección natural y*

de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con un intercambio de información estructurado, aunque aleatorizado, para constituir así un algoritmo de búsqueda que tenga algo de las genialidades de las búsquedas humanas" [Goldberg 1989].

2.2.1.1 Codificación de las soluciones

Cualquier solución potencial a un problema puede ser presentada dando valores a una serie de parámetros. El conjunto de todos los parámetros (*genes* en la terminología de AA.GG.) se codifica en una cadena de valores denominada *cromosoma*.

El conjunto de los parámetros representado por un cromosoma particular recibe el nombre de *genotipo*. El genotipo contiene la información necesaria para la construcción del organismo, es decir, la solución real al problema, denominada *fenotipo*. Por ejemplo, en términos biológicos, la información genética contenida en el ADN de un individuo sería el genotipo, mientras que la expresión de ese ADN (el propio individuo) sería el fenotipo.

Desde los primeros trabajos de John Holland la codificación suele hacerse mediante valores binarios. Se asigna un determinado número de bits a cada parámetro y se realiza una discretización de la variable representada por cada gen. El número de bits asignados dependerá del grado de ajuste que se desee alcanzar. Evidentemente no todos los parámetros tienen por qué estar codificados con el mismo número de bits. Cada uno de los bits pertenecientes a un gen suele recibir el nombre de *alelo.*

En la siguiente figura se muestra un ejemplo de un individuo binario que codifica 3 parámetros:

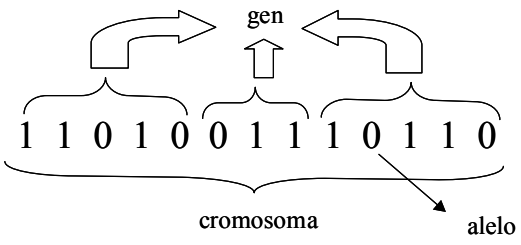


Figura 2.8. *Individuo genético binario*

Sin embargo, también existen representaciones que codifican directamente cada parámetro con un valor entero, real o en punto flotante. A pesar de que se acusa a estas representaciones de degradar el paralelismo implícito de las representaciones

binarias, permiten el desarrollo de operadores genéticos más específicos al campo de aplicación del AG.

2.2.1.2 Funcionamiento

Para alcanzar la solución a un problema se parte de un conjunto inicial de individuos, llamado población, generado de manera aleatoria. Cada uno de estos individuos representa una posible solución al problema. Estos individuos evolucionarán tomando como base los esquemas propuestos por Darwin sobre la selección natural, y se adaptarán en mayor medida tras el paso de cada generación a la solución requerida. La evolución de estos individuos se realizará de forma análoga a como se realiza en el mundo natural: se seleccionarán un conjunto de individuos que se combinarán entre sí (cruce), y su descendencia se insertará en la población. Con una probabilidad muy baja, cuando se inserte un individuo nuevo en la población, este sufrirá una mutación. El funcionamiento general del algoritmo puede verse ilustrado en el diagrama expuesto en la figura 2.9.

Los AA.GG. trabajan sobre una población de individuos. Cada uno de ellos representa una posible solución al problema que se desea resolver. Todo individuo tiene asociado un ajuste de acuerdo a la bondad con respecto al problema de la solución que representa (en la naturaleza el equivalente sería una medida de la eficiencia del individuo en la lucha por los recursos).

En la elaboración de una nueva generación, se denomina reproducción a la creación de nuevos individuos a partir de los ya existentes en la población que forma la generación anterior. Existen dos tipos fundamentales de reproducción:

- ✓ Asexual: Ocurre cuando un individuo de la generación anterior da lugar a uno de la nueva, generalmente por copia de aquél.
- ✓ Sexual: Ocurre cuando se generan varios individuos nuevos a partir del mismo número de individuos de la generación anterior. A esta operación se denomina cruce, y se basa en crear nuevas cadenas de bits a partir de subcadenas de los individuos seleccionados. Típicamente se toman dos individuos (progenitores) para generar otros dos nuevos. Se establece una posición antes de la cual los bits corresponderán a un progenitor y después de ella los bits serán los del otro. A este cruce se denomina cruce de un solo punto, pero podrían utilizarse varios puntos, correspondiendo las

subcadenas de bits que separan los puntos alternativamente a progenitores distintos.

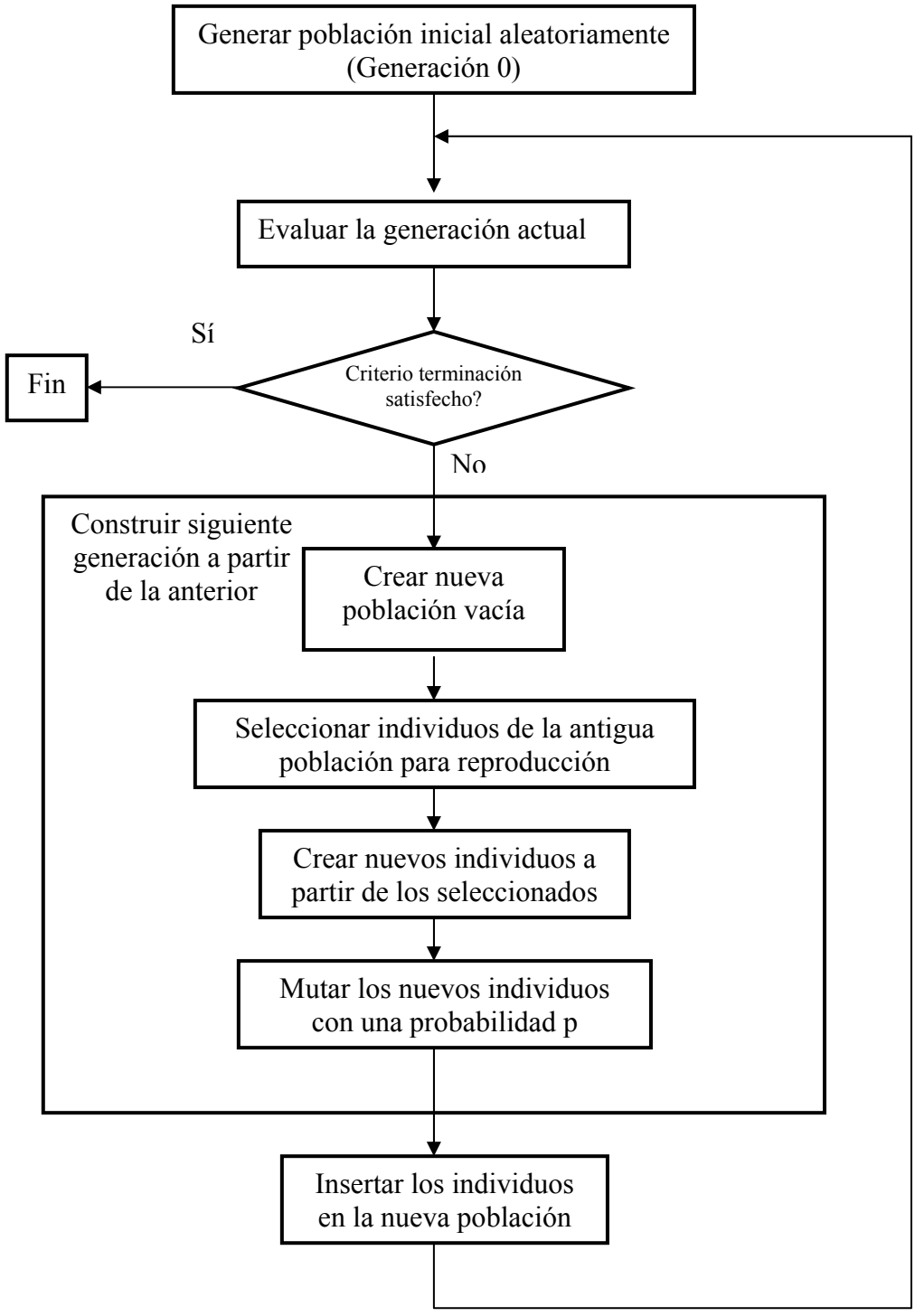


Figura 2.9. *Diagrama de funcionamiento general del AG*

Posteriormente a la creación de nuevos individuos, y con una probabilidad que suele ser muy baja (1% ~ 2%), cada individuo nuevo es sometido a un proceso de mutación: se varía la cadena de bits haciendo un cambio al azar.

Finalmente, los individuos nuevos son insertados en la nueva generación creando así una nueva población que conforma la siguiente generación.

Este proceso se repite hasta que se satisface algún criterio de parada establecido, como podrían ser:

- ✓ En la población hay algún individuo que ha alcanzado un ajuste lo suficientemente bueno.
- ✓ Ha transcurrido el número de generaciones máximo prefijado.
- ✓ La población ha convergido. Un gen ha convergido cuando el 95% de la población tiene el mismo valor para él, en el caso de trabajar con codificaciones binarias, o valores dentro de un rango especificado, en el caso de trabajar con otro tipo de codificaciones. Una vez que todos los genes alcanzan la convergencia se dice que la población ha convergido. Cuando esto ocurre la media de bondad de la población se aproxima a la bondad del mejor individuo.

Dado que cada individuo representa una posible solución al problema, la existencia de un gran número de individuos en la población implica que el algoritmo realiza una búsqueda en muchas regiones distintas del espacio de estados simultáneamente.

El resultado de la evolución (ya sea natural o simulada) no ha sido descubierta por un método de búsqueda ciego a través del espacio de estados del problema, sino por una búsqueda directa desde posiciones aleatorias en ese espacio. De hecho, de acuerdo con Goldberg [Goldberg 89], la evolución simulada de una solución a través de los AA.GG. en muchos casos es más eficiente y robusta que otras técnicas de búsqueda, como la ciega o la basada en cálculos.

Una ventaja adicional de los AA.GG. es que la estrategia de resolución de problemas utiliza una medida de ajuste para dirigir la búsqueda, y no se requiere ningún conocimiento específico del espacio de búsqueda, y pueden operar bien en espacios que tengan saltos, ruido, valles, etc. Como cada individuo dentro de una población dirige la búsqueda, el AG realiza una búsqueda en paralelo en numerosos puntos del espacio de estados con numerosas direcciones de búsqueda.

Sobre este algoritmo inicialmente propuesto por Holland se han definido numerosas variantes.

Una de las más extendidas consiste en prescindir de la población temporal de manera que los operadores genéticos de cruce y mutación se aplican directamente sobre la población genética. Con esta variante el proceso de cruces varía ligeramente. Ahora no basta, en el caso de que el cruce se produzca, con insertar directamente la descendencia en la población. Puesto que el número de individuos de la población se ha de mantener constante, antes de insertar la descendencia en la población se le ha de hacer sitio. Existen para ello diversas opciones:

- Reemplazo de padres: para hacer hueco a la descendencia en la población se eliminan de ella a los padres.
- Reemplazo de individuos similares: cada uno de los individuos de la descendencia reemplazará a un individuo de la población con un ajuste similar al suyo. Para escoger este individuo se obtiene la posición en la que se debería insertar el nuevo individuo para mantener ordenada la población y se escoge para insertarlo una posición al azar de su vecindad (p.e. uno de entre los cinco individuos superiores o inferiores).
- Reemplazo de los peores individuos: los individuos que se eliminarán de la población para dejar paso a la descendencia se seleccionarán aleatoriamente de entre los peores individuos de la población. Por lo general se consideran individuos pertenecientes al último 10%.
- Reemplazo aleatorio: los individuos eliminados se seleccionan al azar.

Evidentemente trabajando con una única población no se puede decir que se pase a la siguiente generación cuando se llene la población, pues siempre está llena. En este caso el paso a la siguiente generación se producirá una vez que se hayan alcanzado cierto número de cruces y mutaciones. Este número dependerá de la tasa de cruces y mutaciones especificadas por el usuario y del tamaño de la población. Así, con una tasa de cruces del 90%, una tasa de mutaciones del 0.02% y trabajando con 100 individuos se pasaría a la siguiente generación cuando se alcanzasen 45 cruces (cada cruce genera 2 individuos con lo que se habrían insertado en la población 90 individuos, esto es el 90%) ó 2 mutaciones.

2.2.1.3 Operadores genéticos

Para el paso de una generación a la siguiente se aplican una serie de operadores genéticos. Los más empleados son los operadores de selección, cruce, copia y mutación. En el caso de no trabajar con una población intermedia temporal también cobran relevancia los algoritmos de reemplazo.

2.2.1.3.1 Selección

Los algoritmos de selección serán los encargados de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no. Puesto que se trata de imitar lo que ocurre en la naturaleza, se ha de otorgar un mayor número de oportunidades de reproducción a los individuos más aptos. Por lo tanto, la selección de un individuo estará relacionada con su valor de ajuste. No se debe sin embargo eliminar por completo las opciones de reproducción de los individuos menos aptos, pues en pocas generaciones la población se volvería homogénea.

Para evitar una predominancia de un individuo en la reproducción, ya sea por cruce o copia, es necesario regular bien la selección de individuos para aplicarles estos operadores. Existen muchos algoritmos de selección, pertenecientes a los AA.GG. tradicionales, que realizan la tarea de escoger qué individuos se van a reproducir y cuáles no.

En general todos los algoritmos de selección se basan en la misma idea: que los individuos más aptos tengan más posibilidades de ser escogidos para reproducirse, pero sin eliminar por completo las posibilidades de los menos aptos, puesto que de ser así la población convergería en pocas generaciones. Para medir la bondad de un individuo, éste está valorado con un nivel de ajuste o aptitud, y en base a ese nivel se puede decidir la elección de individuos. Algunos de los algoritmos más utilizados son el de la selección por torneo o el de la ruleta.

En la selección por torneo [Wetzel 1983], se escoge un número de individuos al azar de la población (típicamente dos individuos) y es seleccionado el mejor de ellos. Este método de selección es muy usado, y permite regular la presión de selección que se ejerce sobre la población variando el número de individuos que participan en el torneo. De esta forma, si participa un número bajo, se ejerce poca presión y se dan más oportunidades de ser seleccionados a los menos aptos. Conforme crece el número de individuos, serán seleccionados los mejores más frecuentemente. Un caso particular es

el *elitismo global.* Se trata de un torneo en el que participan todos los individuos de la población con lo cual la selección se vuelve totalmente determinística. Además, es posible seleccionar un individuo varias veces.

Existen dos versiones de selección mediante torneo:

- Determinística
- Probabilística

En la versión determinística se selecciona al azar un número p de individuos (generalmente se escoge p=2). De entre los individuos seleccionados se selecciona el más apto para pasarlo a la siguiente generación.

La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio en el intervalo [0, 1], si es menor que un parámetro *p* (fijado para todo el proceso evolutivo) se escoge el individuo más apto y en caso contrario el menos apto. Generalmente p toma valores en el rango $0.5 < p \leq 1$

En la selección por ruleta [DeJong 1975], todos los individuos de la población son dispuestos en una ruleta ocupando cada uno una parte proporcional al nivel de ajuste del individuo comparado con el nivel de ajuste de toda la población (suma de ajustes); es decir, ocupan más espacio en la ruleta los mejores individuos. Para realizar la selección, simplemente se hace girar la ruleta y se devuelve el individuo seleccionado por ella. Este método es muy utilizado por su simplicidad y sus buenos resultados. Sin embargo, presenta el problema de que al poder seleccionar el mismo individuo varias veces, el mejor individuo sea escogido muchas veces y acabe predominando en la población. Es un método muy sencillo pero ineficiente a medida que aumenta el tamaño de la población (su complejidad es $O(n^2)$). En mucha bibliografía se suele referenciar a este método con el nombre de Selección de Montecarlo.

Existen muchos otros algoritmos, en los que el número de veces que un individuo es seleccionado se determina de forma determinística. Esto evita problemas de predominancia de un individuo. Cada uno de estos algoritmos presenta variaciones respecto al número de veces que se tomarán los mejores y peores. De esta forma, se impondrá una presión en la búsqueda en el espacio de estados en la zona donde se encuentra el mejor individuo (en el caso de que se seleccionen más veces los mejores), o bien que se tienda a repartir la búsqueda por el espacio de estados, pero sin dejar de

tender a buscar en la mejor zona (caso de repartir más la selección). Los más destacados algoritmos son: "sobrante estocástico" [Brindle 1981] [Booker 1982], "universal estocástica" [Baker 1987] o "muestreo determinístico" [DeJong 1975]. El auténtico motor de la búsqueda es la selección de individuos para la generación de la nueva población. La elección de un algoritmo u otro determinará la forma de búsqueda que se utilizará, estableciendo más presión de búsqueda en la mejor solución hallada hasta el momento o permitiendo la exploración de nuevas zonas en el espacio de estados.

2.2.1.3.2 Cruce

Una vez seleccionados los individuos, éstos son recombinados para producir la descendencia que se insertará en la siguiente generación. Tal y como se ha indicado anteriormente el cruce es una estrategia de reproducción sexual. Su importancia para la transición entre generaciones es elevada puesto que las tasas de cruce con las que se suele trabajar rondan el 90%.

Los diferentes métodos de cruce podrán operar de dos formas diferentes. Si se opta por una estrategia destructiva los descendientes se insertarán en la población temporal aunque sus padres tengan mejor ajuste (trabajando con una única población esta comparación se realizará con los individuos a reemplazar). Por el contrario, utilizando una estrategia no destructiva la descendencia pasará a la siguiente generación únicamente si supera la bondad del ajuste de los progenitores (o de los individuos a reemplazar).

La idea principal del cruce se basa en que, si se toman dos individuos correctamente adaptados al medio y se obtiene una descendencia que comparta genes de ambos, existe la posibilidad de que los genes heredados sean precisamente los causantes de la bondad de los progenitores. Al compartir las características buenas de dos individuos, la descendencia, o al menos parte de ella, debería tener una bondad mayor que cada uno de los padres por separado. Si el cruce no agrupa las mejores características en uno de los hijos y la descendencia tiene un peor ajuste que los padres no significa que se esté dando un paso atrás. Optando por una estrategia de cruce no destructiva garantizamos que pasen a la siguiente generación los mejores individuos. Si, aún con un ajuste peor, se opta por insertar a la descendencia, y puesto que los genes de los padres continuarán en la población –aunque dispersos y posiblemente levemente

modificados por la mutación–, en posteriores cruces se podrán volver a obtener estos padres, recuperando así la bondad previamente perdida.

Existen multitud de algoritmos de cruce. Sin embargo los más empleados son los que se detallarán a continuación:

- Cruce de 1 punto
- Cruce de 2 puntos
- Cruce uniforme

El cruce de un punto es la más sencilla de las técnicas de cruce. Una vez seleccionados dos individuos se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los dos individuos para generar los nuevos descendientes. De esta manera, ambos descendientes heredan información genética de los padres.

En la siguiente figura se puede ver con claridad el proceso:

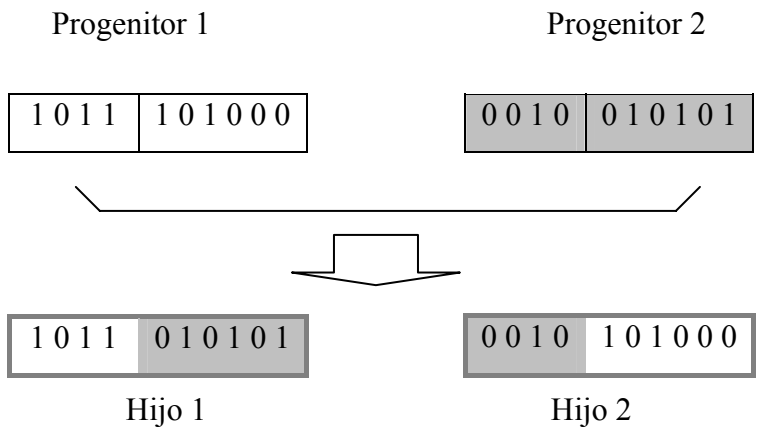


Figura 2.10. *Ejemplo de cruce de un solo punto*

En la bibliografía suele referirse a este tipo de cruce con el nombre de SPX (*Single Point Crossover*).

El cruce de dos puntos es una generalización del cruce de 1 punto. En vez de cortar por un único punto los cromosomas de los padres, como en el caso anterior, se realizan dos cortes. Deberá tenerse en cuenta que ninguno de estos puntos de corte coincida con el extremo de los cromosomas para garantizar que se originen tres segmentos. Para generar la descendencia se escoge el segmento central de uno de los

padres y los segmentos laterales del otro padre. Generalmente se suele referir a este tipo de cruce con las siglas DPX (Double Point Crossover). En la siguiente figura se muestra un ejemplo de cruce en dos puntos.

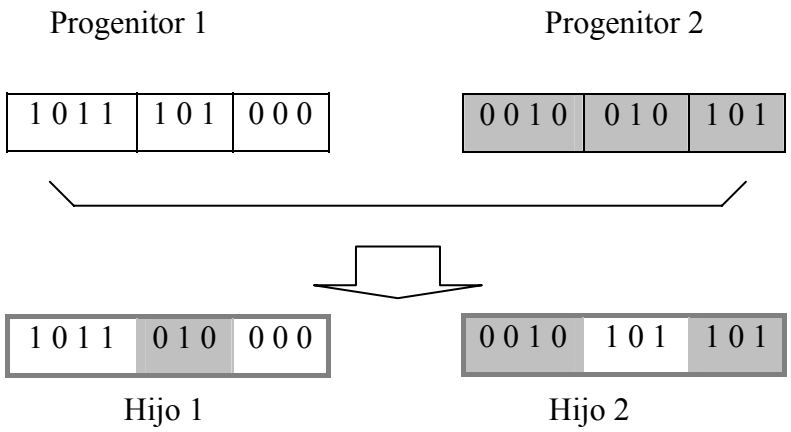


Figura 2.11. *Ejemplo de cruce de dos puntos*

Generalizando, se pueden añadir más puntos de cruce dando lugar a algoritmos de cruce multipunto. Sin embargo existen estudios que desaprueban esta técnica [DeJong 75]. Aunque se admite que el cruce de 2 puntos aporta una sustancial mejora con respecto al cruce de un solo punto, el hecho de añadir un mayor número de puntos de cruce reduce el rendimiento del AG. El problema principal de añadir nuevos puntos de cruce radica en que es más fácil que los segmentos originados sean corrompibles; es decir, que por separado quizás pierdan las características de bondad que poseían conjuntamente. Sin embargo, no todo son desventajas y añadiendo más puntos de cruce se consigue que el espacio de búsqueda del problema sea explorado más a fondo.

El cruce uniforme, en cambio, es una técnica completamente diferente de las vistas hasta el momento. Cada gen de la descendencia tiene las mismas probabilidades de pertenecer a uno u otro progenitor. Aunque se puede implementar de muy diversas formas, la técnica implica la generación de una máscara de cruce con valores binarios. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer progenitor. Si por el contrario hay un 0 el gen se copia del segundo progenitor. Para producir el segundo descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y los ceros de la máscara de cruce.

Tal y como se puede apreciar en la siguiente figura, la descendencia contiene una mezcla de genes de cada uno de los progenitores. El número efectivo de puntos de cruce no es fijo, pero será por término medio L/2, siendo L la longitud del cromosoma (número de alelos en representaciones binarias o de genes en otro tipo de representaciones).

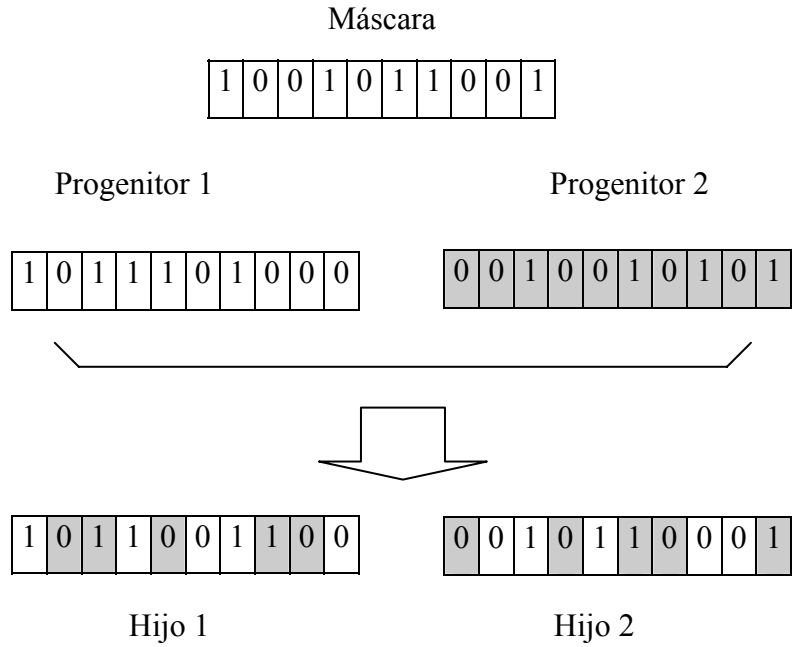


Figura 2.12. *Ejemplo de cruce uniforme*

La máscara de cruce no permanece fija durante todo el proceso evolutivo. Se genera de manera aleatoria para cada cruce.

Se suele referir a este tipo de cruce con las siglas UPX (*Uniform Point Crossover*).

Los tres tipos de cruce vistos hasta el momento son válidos para cualquier tipo de representación del genotipo. Si se emplean genotipos compuestos por valores enteros o reales pueden definirse otros tipos de operadores de cruce:

- Media: el gen de la descendencia toma el valor medio de los genes de los padres. Tiene la desventaja de que únicamente se genera un descendiente en el cruce de dos progenitores.
- Media geométrica: cada gen de la descendencia toma como valor la raíz cuadrada del producto de los genes de los padres. Presenta el problema añadido de qué signo dar al resultado si los padres tienen signos diferentes.

- Extensión: se toma la diferencia existente entre los genes situados en las mismas posiciones de los padres y se suma al valor más alto o se resta del valor más bajo. Solventa el problema de generar un único descendiente.

2.2.1.3.3 Copia

La copia es otra estrategia reproductiva para la obtención de una nueva generación a partir de la anterior. A diferencia del cruce, se trata de una estrategia de reproducción asexual. Consiste simplemente en la copia de un individuo en la nueva generación.

El porcentaje de copias de una generación a la siguiente es relativamente reducido, pues en caso contrario se corre el riesgo de una convergencia prematura de la población hacia ese individuo. De esta manera, el tamaño efectivo de la población se reduciría notablemente y la búsqueda en el espacio del problema se focalizaría en el entorno de ese individuo.

Lo que generalmente se suele hacer es seleccionar dos individuos para el cruce, y si éste finalmente no tiene lugar, se insertan en la siguiente generación los individuos seleccionados.

2.2.1.3.4 Mutación

La mutación de un individuo provoca que alguno de sus genes, generalmente uno sólo, varíe su valor de forma aleatoria.

Aunque se pueden seleccionar los individuos directamente de la población actual y mutarlos antes de introducirlos en la nueva población, la mutación se suele utilizar de manera conjunta con el operador de cruce. Primero se seleccionan dos individuos de la población para realizar el cruce. Si el cruce tiene éxito entonces uno de los descendientes, o ambos, se muta con cierta probabilidad P_m. Se imita de esta manera el comportamiento que se da en la naturaleza, pues cuando se genera la descendencia siempre se produce algún tipo de error, por lo general sin mayor trascendencia, en el paso de la carga genética de padres a hijos.

La probabilidad de mutación es muy baja, generalmente menor al 1%. Esto se debe sobre todo a que los individuos suelen tener un ajuste menor después de mutados. Sin embargo, se realizan mutaciones para garantizar que ningún punto del espacio de búsqueda tenga una probabilidad nula de ser examinado.

Tal y como se ha comentado, la mutación más usual es el reemplazo aleatorio. Este consiste en variar aleatoriamente un gen de un cromosoma. Si se trabaja con codificaciones binarias consistirá simplemente en negar un bit. También es posible realizar la mutación intercambiando los valores de dos alelos del cromosoma. Con otro tipo de codificaciones no binarias existen otras opciones:

- Incrementar o decrementar a un gen una pequeña cantidad generada aleatoriamente.
- Multiplicar un gen por un valor aleatorio próximo a 1.

Aunque no es lo más común, existen implementaciones de AA.GG. en las que no todos los individuos tienen los cromosomas de la misma longitud. Esto implica que no todos ellos codifican el mismo conjunto de variables. En este caso existen mutaciones adicionales como puede ser añadir un nuevo gen o eliminar uno ya existente.

2.2.1.3.5 Algoritmos de reemplazo

Cuando en vez de trabajar con una población temporal se hace con una única población, sobre la que se realizan las selecciones e inserciones, deberá tenerse en cuenta que para insertar un nuevo individuo deberá de eliminarse previamente otro de la población. Existen diferentes métodos de reemplazo:

- Aleatorio: el nuevo individuo se inserta en un lugar cualquiera de la población.
- Reemplazo de padres: se obtiene espacio para la nueva descendencia liberando el espacio ocupado por los padres.
- Reemplazo de similares: una vez obtenido el ajuste de la descendencia se selecciona un grupo de individuos (entre seis y diez) de la población con un ajuste similar. Se reemplazan aleatoriamente los que sean necesarios.
- Reemplazo de los peores: de entre un porcentaje de los peores individuos de la población se seleccionan aleatoriamente los necesarios para dejar sitio a la descendencia.

2.2.1.4 Evaluación

Para el correcto funcionamiento de un AG se debe de poseer un método que indique si los individuos de la población representan o no buenas soluciones al problema planteado. Por lo tanto, para cada tipo de problema que se desee resolver deberá derivarse un nuevo método, al igual que ocurrirá con la propia codificación de los individuos.

De esto se encarga la función de evaluación, que establece una medida numérica de la bondad de una solución. Esta medida recibe el nombre de ajuste. En la naturaleza el ajuste (o adecuación) de un individuo puede considerarse como la probabilidad de que ese individuo sobreviva hasta la edad de reproducción y se reproduzca. Esta probabilidad deberá estar ponderada con el número de descendientes. Evidentemente no es lo mismo una probabilidad de reproducción del 25% en una población de un par de cientos de individuos que esa misma probabilidad en una población de varios millones.

En el mundo de los AA.GG. se empleará esta medición para controlar la aplicación de los operadores genéticos. Es decir, permitirá controlar el número de selecciones, cruces, copias y mutaciones llevadas a cabo.

La cuantificación de la bondad de un determinado individuo se realiza por medio del ajuste de ese individuo. Este valor representa lo bien que el fenotipo del individuo soluciona el problema actual.

El ajuste es probablemente el principal concepto en la evolución darwiniana, se refiere a la habilidad que tiene un individuo de competir en un entorno por los recursos disponibles. Goldberg describió la función de ajuste como "*una medida de beneficio, utilidad o bondad que queremos maximizar*" [Goldberg 89].

En el AG, esta competición se basa en la actuación del cromosoma dentro del dominio del problema. Se determina una escala adecuada a la tarea como "tiempo antes de fallo" [Randall 94], o "tiempo antes de estabilizarse" [Koza 90]. Después de haber aplicado un cromosoma al problema, se le asigna un valor de ajuste que refleje su actuación. De esta manera, cuando la población entera haya sido probada, la habilidad relativa de cada cromosoma puede ser identificada.

Para valorar esta medida de ajuste, existen tres tipos fundamentales [Koza 92]:

- Estandarizado. Este tipo de ajuste s(i,t) mide la bondad de un individuo i en la generación t, de tal forma que valores próximos a cero indican un buen valor de ajuste y valores lejanos un mal individuo. Por tanto, en una generación t un individuo i será peor que otro j si s(i,t) > s(j,t). Esta medida es muy útil en problemas en los que la cuantificación del nivel de ajuste de los individuos se basa en penalizaciones, como puede ser el error en inducción de fórmulas, error cuadrático medio, número de ejecuciones necesarias para encontrar la solución, etc.

- Ajustado. Este valor se obtiene a partir del ajuste estandarizado de la siguiente forma:

$$a(i,t) = \frac{1}{1 + s(i,t)}$$

  Con esta medición la bondad se cuantifica entre 0 y 1, siendo el valor 1 correspondiente al mejor individuo.

- Normalizado. Es un valor de ajuste comparativo del individuo con toda la población. Se obtiene de la siguiente expresión, dado el tamaño de población M:

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)}$$

  Este valor está comprendido entre 0 y 1. Este tipo de ajuste indica el nivel de bondad dentro de la población: ha desaparecido el componente de objetividad de evaluación de los tipos anteriores y un valor cercano a 1 ya no indica que ese individuo represente una solución buena al problema, sino que ese individuo representa una solución destacada y notablemente mejor que la del resto de la población. Este valor es utilizado para las selecciones proporcionales al ajuste, como la ruleta. En este caso, la proporción de ruleta ocupada por un individuo será este valor, ya que la suma de todos será la unidad.

La aproximación más común consiste en crear explícitamente una medida de ajuste para cada individuo de la población. A cada uno de los individuos se les asigna un valor de ajuste escalar por medio de un procedimiento de evaluación bien definido. Tal y como se ha comentado, este procedimiento de evaluación será específico del dominio del problema en el que se aplica el AG. También puede calcularse el ajuste mediante una manera "co-evolutiva". Por ejemplo, el ajuste de una estrategia de juego se determina aplicando esa estrategia contra la población entera (o en su defecto una muestra) de estrategias de oposición.

2.2.1.5 Parámetros

Los principales parámetros que se pueden configurar para variar la ejecución son:

- Tamaño de la población. Este parámetro indica el número de individuos que va a tener la población. En general este parámetro se ajusta de forma proporcional a la complejidad del problema, tomando valores altos cuanto mayor sea ésta. De esta forma, cuanto más complicado es un problema, habrá más opciones de conseguir mejores resultados en un menor número de generaciones, puesto que se generan más individuos nuevos. Sin embargo, un tamaño alto puede no ser siempre la mejor solución: es posible tomar un tamaño menor y confiar más en la evolución durante mayor número de generaciones [Gathercole 1997] [Fuchs 1999].
- La tasa de cruces, es decir, el porcentaje de individuos de la siguiente generación que serán creados a partir de cruces de individuos de la anterior. Esta tasa suele ser alta, generalmente supera el 90%.
- La probabilidad de mutación. Esta probabilidad suele ser muy baja (menor de 0.05).
- Los algoritmos de cruce, selección y mutación utilizados.
- El criterio de parada del algoritmo. Como ya se ha explicado, este puede ser:
  - ✓ Número máximo de generaciones.
  - ✓ Haber alcanzado un valor de ajuste aceptable.

- ✓ Convergencia de la población.

2.2.2 Programación Genética

2.2.2.1 Orígenes

La programación genética surge como una evolución de los algoritmos genéticos tradicionales, manteniendo el mismo principio de selección natural. Lo que ahora se pretende es resolver los problemas mediante la inducción de programas y algoritmos.

Ya en las primeras conferencia sobre algoritmos genéticos, la primera desarrollada en la Universidad de Carnegie Mellon en 1985, y la segunda en Hillsdale en 1987, se puede encontrar dos artículos que, sin usar explícitamente el nombre de programación genética (que fue introducido por John R. Koza), pueden ser considerados como precursores en la materia: "A Representation for the Adaptive Generation of Simple Sequential Programs" [Cramer 1985] y "Using the Genetic Algorithm to Generate Lisp Source Code to solve the Prisoner's Dilemma" [Fujiki 1987].

El primero de estos artículos plantea un sistema adaptativo para la generación de pequeños programas secuenciales. Para ello, hace uso de dos lenguajes: JB (que representa los programas en forma de cadenas de números) y TB (versión evolucionada de JB, pero con estructura de árbol para representar programas). El objetivo principal del artículo es conseguir una forma de representar programas que, por un lado, permita la aplicación de los operadores genéticos clásicos (mutación, cruce, inversión) y que, por otro lado, produzca sólo programas "bien formados", incluso cuando se apliquen dichos operadores sobre los programas. No es importante que todos los programas que se puedan obtener sean útiles (de eso ya se encargarán los criterios de selección): sólo importa que estén dentro del espacio de programas sintácticamente correctos.

La gran importancia de este artículo se halla en la constatación de la importancia que tiene la representación de los programas para su manipulación. Los problemas que plantea el lenguaje JB pueden ser eliminados si se usa TB y su estructuración en forma de árboles. Este hecho, que en principio puede ser considerado poco relevante, ha demostrado ser de gran utilidad en trabajos posteriores, y, de hecho, todo el material actual que hay sobre programación genética se basa en la representación arbórea.

El segundo artículo trata sobre la resolución de un problema clásico: el dilema del prisionero. En él, dos prisioneros son interrogados por separado. Cada uno debe decidir

si delatar o no al otro, y en función de la declaración de ambos obtienen puntos. Si ambos delatan al otro, obtienen cada uno 1 punto. Si ninguno delata al otro, obtienen cada uno 3 puntos. Si uno delata y el otro no, el delator obtiene 4 puntos y el otro 0. Al cabo de un número determinado de interrogatorios (rondas) el que obtenga más puntos gana. Este problema ha sido usado como modelo en multitud de situaciones, desde el nivel de las relaciones personales hasta las negociaciones entre grandes corporaciones.

El interés del artículo se centra en el apartado dedicado a la representación del conocimiento. En teoría, el sistema debería ser lo suficientemente flexible como para encontrar una solución única y novedosa, aunque en la práctica suele ser necesario, e incluso deseable, proporcionar cierto tipo de conocimiento sobre la solución para agilizar el proceso de búsqueda. En el sistema de Hicklin, por ejemplo, este conocimiento es representado en forma de programa, generado por medio de una serie de producciones. El grado de flexibilidad del sistema y el grado de conocimiento incorporado dependerá en cómo sean definidas dichas producciones. Para el artículo, en concreto, se usó un conjunto restringido de expresiones LISP, que demostró ser de gran utilidad y funcionalidad.

2.2.2.2 Codificación de programas

La mayor diferencia entre los AA.GG. y la PG es la forma de codificación de la solución al problema. En PG la codificación se realiza en forma de árbol, de forma similar a como los compiladores leen los programas según una gramática prefijada.

2.2.2.2.1 Elementos del árbol

Al haber una representación de árbol, existirán dos tipos de nodos:

- Terminales, u hojas del árbol. Son aquellos que no tienen hijos. Normalmente se asocian con valores constantes o variables.
- Funciones. Son aquellos que tienen uno o más hijos. Generalmente se asocian con operadores del algoritmo que se quiere desarrollar.

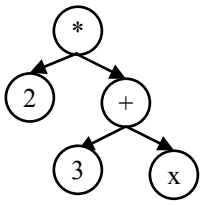


Figura 2.13. *Árbol para la expresión 2*(3+x)*

En la figura 2.13 se puede ver un ejemplo de árbol, que representa el programa f(x) = 2*(3+x). Tiene como nodos no terminales los correspondientes al producto y a la suma, y como terminales los correspondientes a los valores 2 y 3 y la variable x.

Una parte fundamental del funcionamiento de la programación genética es la especificación del conjunto de elementos terminales y funciones antes del inicio del proceso evolutivo. Con los nodos que se le especifique, el algoritmo construirá los árboles. Por tanto, es necesario un mínimo proceso de análisis del problema para configurar el algoritmo, puesto que hay que decirle qué operadores puede utilizar. Como regla general, es conveniente ajustar el número de operadores sólo a los necesarios, puesto que la adición de elementos que no sean necesarios no provocará que no se encuentre la solución, pero sí que el algoritmo tarde más en encontrarla.

A la hora de especificar los conjuntos de elementos terminales y funciones, es necesario que estos conjuntos posean dos requisitos, que son *cerradura* y *suficiencia*. El requisito de suficiencia dice que la solución al problema debe poder ser especificada con el conjunto de operadores especificados. El requisito de cerradura dice que debe ser posible construir árboles correctos con los operadores especificados.

Dado que el proceso del construcción de árboles es un proceso basado en el azar, muchos de los árboles construidos no serán correctos no por no seguir las reglas de la gramática sino por la aplicación de operadores (nodos no terminales) a elementos que no están en su dominio. Por esta razón no se aplican estos operadores directamente, sino una modificación de ellos en la que se amplía su dominio de aplicación. El ejemplo más claro es el operador de división, cuyo dominio es el conjunto de números reales excepto el valor cero. Ampliando su dominio, se define un nuevo operador (%):

$$\%\,(a,b) \begin{cases} 1 & \text{si } b = 0 \\ a/b & \text{si } b \neq 0 \end{cases}$$

A esta nueva operación se denomina *operación de división protegida*, y, en general, cuando se crea una nueva operación, que extiende el dominio de otra, se denomina *operación protegida*.

2.2.2.2.2 Restricciones

Existen dos tipos principales de restricciones:

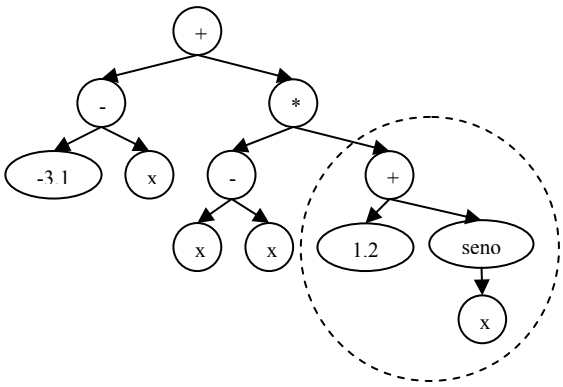
- Tipado.
- Altura máxima del árbol.

Para establecer reglas sintácticas en la creación de árboles, es posible especificar reglas de tipado [Montana 1995]: se establece el tipo de cada nodo (terminales y funciones), y para los no terminales (funciones) el tipo que debe tener cada hijo. De esta forma se especifica la estructura que deben seguir los árboles.

Al especificar el tipo de cada nodo, se está especificando una gramática que va a ser la que siga el algoritmo para la construcción de árboles. Esta gramática permitirá que los árboles tengan la estructura deseada.

Los tipos más usados son aquellos que tienen que ver con la realización de operaciones aritméticas: reales y enteros. Sin embargo, son también muy usados otros como el tipo booleano o el tipo sentencia. Este último se utiliza cuando se quiere desarrollar un programa que sea una secuencia de mandatos y designa nodos que no devuelven nada, sino que su evaluación se basa en los efectos laterales que provoca su ejecución.

La restricción de altura evita la creación de árboles demasiado grandes y fuerza una búsqueda en soluciones cuyo tamaño se acota de antemano. Con esta restricción se evita que los árboles posean mucho código redundante y el crecimiento excesivo de los árboles [Soule 1997] [Soule 1998]. El crecimiento excesivo de los árboles es un fenómeno conocido con el nombre de *bloat*. Este fenómeno se produce de forma espontánea al avanzar el proceso evolutivo, puesto que los árboles evolucionan generando partes que no influyen en su comportamiento. Esto se produce para paliar los efectos nocivos de los operadores de cruce y mutación, puesto que cuantas más partes inútiles tenga un árbol, menos probabilidades habrá de que este sea modificado cuando se le aplique un operador de cruce o mutación. De esta forma, los individuos se protegen a sí mismos. Un ejemplo de una parte inútil de un árbol sería, en un individuo que representa una expresión matemática, un subárbol al cual se va a multiplicar por cero, como se puede ver en la figura 2.14.



-3.1 – x + ((x - x) * (1.2 + seno(x)))

Figura 2.14. *Ejemplo de árbol con una rama inútil*

2.2.2.3 Funcionamiento

2.2.2.3.1 Algoritmo principal

El funcionamiento es similar al de los algoritmos genéticos: se basa en la generación de sucesivas generaciones a partir de las anteriores. Este algoritmo se puede ver en la figura 2.15 [Koza 1992].

Tras la creación inicial de árboles (generalmente serán aleatorios), se construyen sucesivas generaciones a partir de copias, cruces y mutaciones de los individuos de cada generación anterior.

Para poder aplicar la PG será necesario especificar dos elementos fundamentales:

- ✓ Conjunto de terminales y funciones. Dada la forma diferente de codificación que tiene la PG frente a los AA.GG., será necesario especificar qué elementos se pueden utilizar para construir los árboles.
- ✓ Función de ajuste. Indica la bondad de cada individuo.

2.2.2.3.2 Generación inicial de árboles

El primer paso en el funcionamiento del algoritmo es la generación de la población inicial. En la creación de la generación 0, cada árbol se creará de forma más o menos aleatoria, dependiendo del algoritmo, teniendo en cuenta las restricciones que existen en los árboles. Dado que los árboles son aleatorios, los individuos de esta población en general representan soluciones malas al problema.

Para la creación de un árbol existe una gran variedad de algoritmos, pero son tres los más utilizados [Koza 1992]: parcial, completo e intermedio.

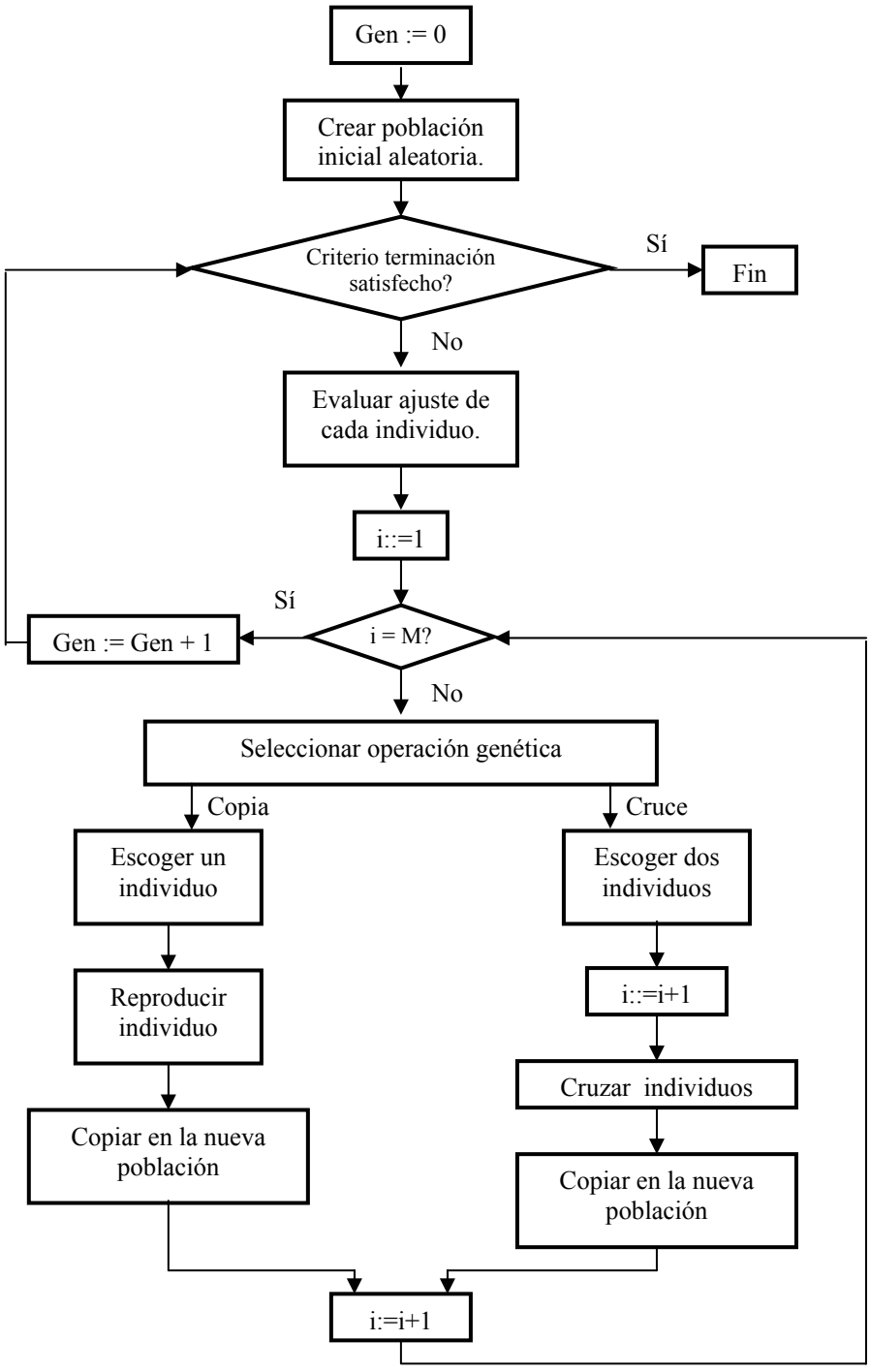


Figura 2.15. *Diagrama de flujo de programación genética*

El algoritmo de creación parcial genera árboles cuya altura máxima no supera la especificada. El algoritmo es el siguiente, dada la altura máxima y los conjuntos de elementos terminales (T) y funciones (F):

```
GeneraÁrbol (altura, F, T)
begin
    if altura=1 then
           Asignar como raíz del árbol un elemento aleatorio de T
    else
           Asignar como raíz del árbol un elemento aleatorio de F∪T
    Para cada hijo de la raíz,
        Asignar a la raíz como hijo el subárbol generado con
           GeneraÁrbol (altura-1, F, T)
    end
```

En este algoritmo, cada hoja tendrá como máximo la profundidad especificada. Para el caso de generar árboles tipados, será necesario mantener la restricción del tipado, con lo que el algoritmo es el siguiente:

```
GeneraÁrbol (altura, F, T, tipo)
    begin
           if altura=1 then
               Asignar la raíz del árbol como un elemento aleatorio de T
                   del tipo especificado.
           else
                   Asignar la raíz del árbol como un elemento aleatorio de
                          F∪T del tipo especificado.
           Para cada hijo de la raíz,
               Asignar a la raíz como hijo el subárbol generado con
                   GeneraÁrbol (altura-1, F, T, tipo de ese hijo)
    end
```

El algoritmo completo genera árboles cuyas hojas están todas a un determinado nivel, pues genera árboles completos. El algoritmo es muy similar al anterior:

```
GeneraÁrbol (altura, F, T)
    begin
           if altura=1 then
               Asignar la raíz del árbol como un elemento aleatorio de T
           else
           Asignar la raíz del árbol como un elemento aleatorio de F
           Para cada hijo de la raíz,
               Asignar a la raíz como hijo el subárbol generado con
                   GeneraÁrbol (altura-1, F, T)
    end
```

Para el caso de utilizar las propiedades de tipado, se realiza la modificación como anteriormente, dando lugar a:

```
GeneraÁrbol (altura, F, T, tipo)
    begin
            if altura=1 then
                Asignar la raíz del árbol como un elemento aleatorio de T
                    del tipo especificado.
            else
            Asignar la raíz del árbol como un elemento aleatorio de F
                    del tipo especificado.
            Para cada hijo de la raíz,
                Asignar a la raíz como hijo el subárbol generado con
                    GeneraÁrbol (altura-1, F, T, tipo de ese hijo)
    end
```

Sin embargo, es necesario tener en cuenta que al utilizar tipado se está forzando a la utilización de un conjunto de reglas sintácticas y esto puede provocar que sea imposible generar árboles de la altura especificada con los conjuntos de terminales y funciones dados.

Por ejemplo, con el conjunto de terminales T y funciones F

$$T = \{ X, Y, 3, 5 \} \qquad F = \{ if, > \}$$

Sería imposible generar un árbol completo de tipo real de altura 2, puesto que para ello la raíz debería ser el elemento "if", con su primer hijo de tipo booleano y como operador booleano solo hay un nodo no terminal, el que representa la relación de mayor, y se necesita un nodo terminal para garantizar el cumplimiento de la restricción de altura. En este ejemplo los conjuntos F y T cumplen la condición de cerradura, pues es posible construir árboles correctos, pero existe una incompatibilidad en el algoritmo completo con la restricción de altura 2 al generar árboles iniciales causada por la gramática que se está implementando.

El algoritmo de creación intermedio es una mezcla de los dos anteriores, creado para que exista mayor variedad en la población inicial, y con ello mayor diversidad genética. Este algoritmo se basa en ejecutar los anteriores alternándolos y tomando distintas alturas para crear todos los elementos de la población. El algoritmo es el siguiente: dado un tamaño de población M y una altura máxima A:

```
for i:=2 to A do begin
    Generar M/(2*(A-1)) árboles de altura i con el método parcial
    Generar M/(2*(A-1)) árboles de altura i con el método completo
end
```

Este método genera un porcentaje de 100/(A-1)% árboles nuevos de altura, variando entre 2 y A, de forma alternativa, completos y parciales.

En general, se evita que se generen árboles de altura 1; es decir, árboles que contengan solo un elemento terminal. En la práctica se modifican los algoritmos para que se evite esta posibilidad.

Estos algoritmos se basan fuertemente en el azar, y la única intervención del usuario está en la introducción de los elementos terminales y funciones. Sin embargo, existen muchos más algoritmos en los que la creación de árboles no es tan aleatoria. Por ejemplo, en [Luke 2000] se asigna una probabilidad de aparición a cada nodo y, de esta forma, se reduce el carácter aleatorio de la creación. Además, se orientan los árboles a que contengan más nodos de una clase que otra. Los algoritmos serán muy similares, con la salvedad de que la elección de los elementos seguirá siendo aleatoria, pero estará ponderada por esa probabilidad asignada.

2.2.2.4 Operadores genéticos

En la creación de una nueva generación se aplican operadores en los que se generan y modifican los árboles. Estos operadores son resultado de adaptar los existentes en los algoritmos genéticos tradicionales a la programación genética, modificándolos para adaptarlos a la codificación en forma de árbol.

Los operadores más utilizados son:

- Cruce.
- Reproducción.
- Selección.
- Mutación.

2.2.2.4.1 Cruce

El principal operador es el de cruce. En él, dos individuos de la antigua población se combinan para crear otros dos individuos nuevos.

Después de seleccionar a dos individuos como padres, se selecciona un nodo al azar en el primero y otro en el segundo de forma que su intercambio no viole ninguna de las restricciones: los nodos deben ser de igual tipo y los árboles nuevos deben seguir manteniendo la altura máxima. El cruce entre los dos padres se efectúa mediante el intercambio de los subárboles seleccionados en ambos padres.

La restricción de altura conlleva a que si se está tomando una altura máxima de A y se selecciona un nodo de un árbol para cruce que está a una profundidad P, del segundo árbol se descartarán para el cruce todos aquellos nodos cuyos subárboles tengan una altura mayor que A-P, puesto que el resultado del cruce daría lugar a un árbol de altura mayor que A. De la misma forma, si el nodo seleccionado en el primer árbol representa a un subárbol de altura A', del segundo se descartan aquellos cuya profundidad sea mayor que A-A', puesto que de insertar el subárbol seleccionado del primer árbol en ese hueco, se violaría igualmente la restricción de altura máxima.

La figura 2.16 muestra un ejemplo de dos árboles seleccionados para cruce. En este caso, la altura máxima que se utiliza es de 5.

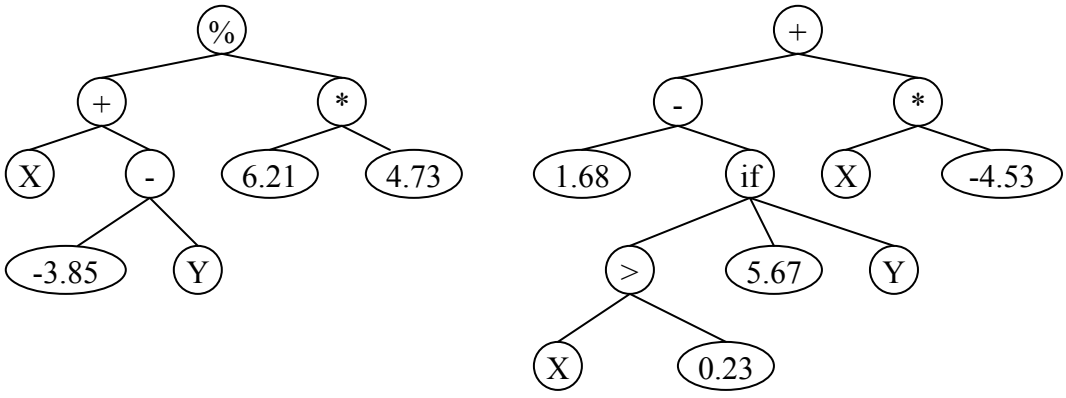


Figura 2.16. *Árboles seleccionados para cruce*

En primer lugar, se selecciona de forma aleatoria un nodo en el primer árbol. En este caso el nodo "+", que representa a un subárbol de altura 3. En el segundo árbol se descarta aquellos nodos de tipo distinto, en este caso el nodo ">", por ser de tipo booleano (figura 2.17).

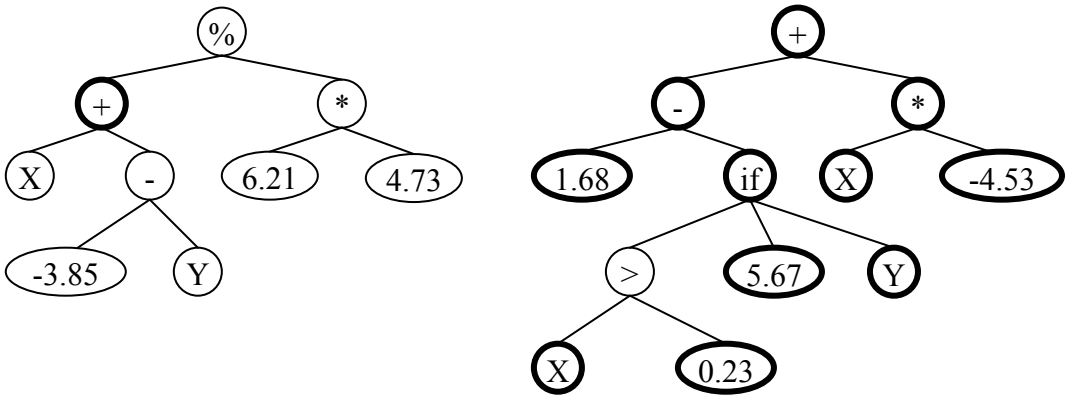


Figura 2.17. *Se selecciona nodo en el primer progenitor y se aplica restricción de tipo en el segundo*

En el segundo árbol se descartan aquellos nodos que nos llevarían a violar la restricción de altura tras el intercambio. Se descarta la raíz del árbol porque llevaría a un árbol demasiado alto en el primer progenitor y los nodos de altura 4 y 5 (es decir, los dos últimos niveles) porque llevarían a un árbol demasiado alto en el segundo progenitor (figura 2.18).

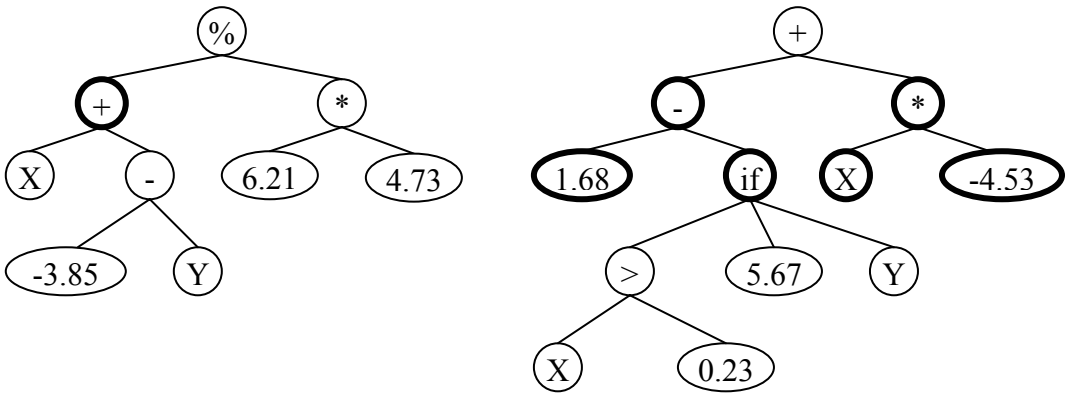


Figura 2.18. *Se descartan nodos en el segundo progenitor que violen la restricción de altura máxima*

En el segundo árbol se selecciona un nodo de los restantes, en este caso el operador "if", y se intercambian los nodos (figuras 2.19 y 2.20).

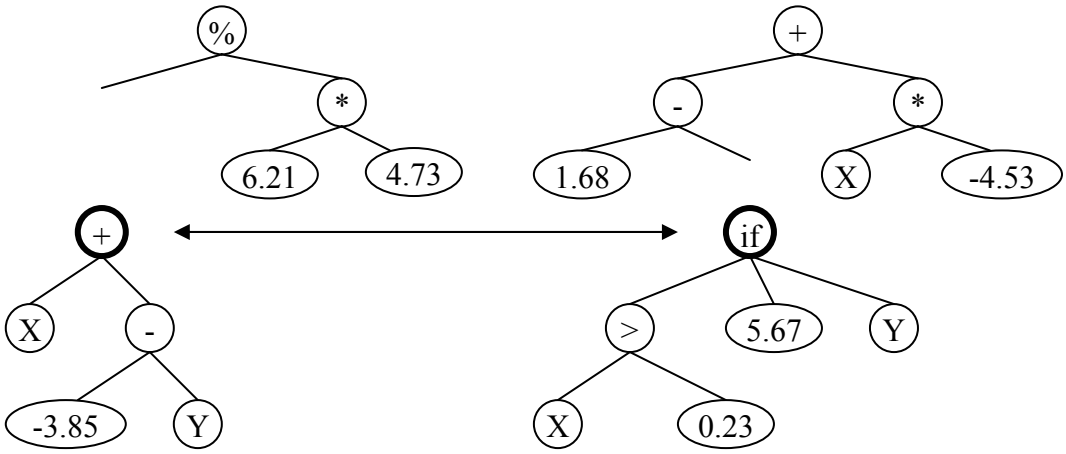


Figura 2.19. *Se selecciona un nodo de los restantes*

Finalmente, se introducen los árboles nuevos en la nueva generación.

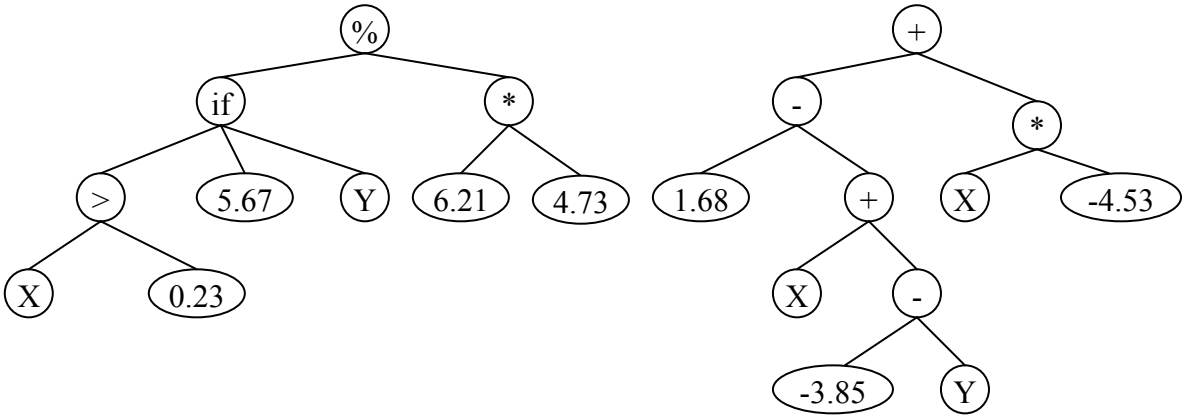


Figura 2.20. *Se intercambian los subárboles y se introducen en la nueva población*

La operación de cruce es sexual en el sentido en que se necesitan dos individuos para generar individuos nuevos.

Este operador es el auténtico motor del funcionamiento de la programación genética y provoca la combinación de resolución de subproblemas (cada subárbol puede interpretarse como una forma de resolver un subproblema) para la resolución del problema principal.

Se ha observado [Soule 1998] que la mayoría de los cruces provocan la generación de individuos peores, así como la aparición de mucho código redundante dentro de los árboles. Este código redundante previene los posibles efectos nocivos de otros operadores más destructivos como el de mutación, pero provoca un crecimiento exagerado de los árboles en poco tiempo. Por ello, una solución es utilizar *cruces no destructivos*: los árboles generados por la operación de cruce son insertados en la nueva

generación si son mejores que sus padres. En caso contrario, se insertan en la nueva generación copias de los padres.

Una ventaja de este tipo de cruces sobre los cruces de los algoritmos genéticos tradicionales es que al cruzar dos padres iguales, los hijos en general son distintos a los padres (y distintos entre ellos). Esto no ocurría en el cruce en los algoritmos genéticos, en los que en este caso, cuando se cruzaban padres idénticos, los hijos eran iguales a los padres.

Por ejemplo, si se cruzan los nodos señalados de los árboles iguales presentes en la figura 2.21.

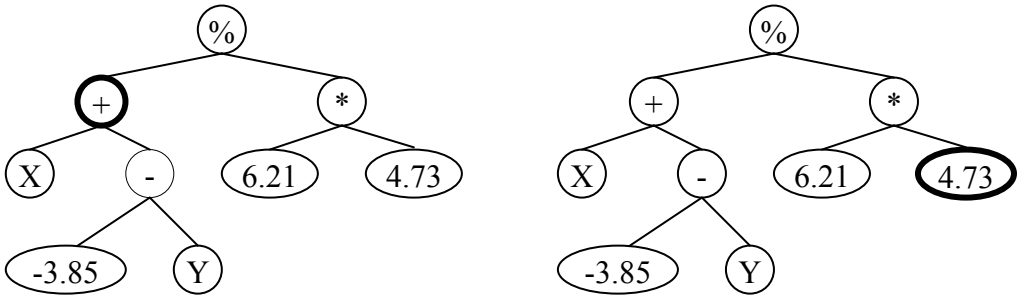


Figura 2.21. *Árboles iguales como ejemplo de cruce*

En este caso, se obtienen los árboles de la figura 2.22, que son diferentes.

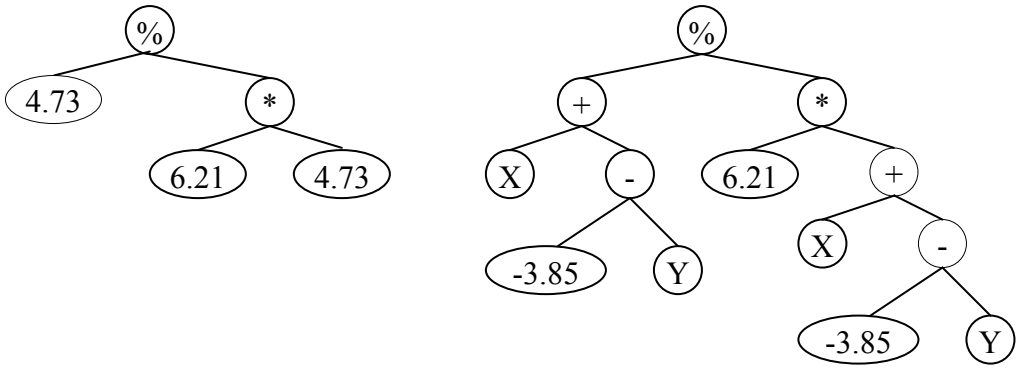


Figura 2.22. *Resultado de realizar un cruce entre árboles iguales*

Al realizar un cruce, el nodo escogido en ambos árboles no se suele tomar al azar: generalmente se asigna una probabilidad de que el nodo tomado sea no terminal. Esta probabilidad suele ser alta (sobre 0.9), puesto que en un árbol la mayoría de los nodos son terminales, y si no se toma esta probabilidad la mayoría de los cruces no son más que permutaciones de elementos terminales entre distintos árboles.

El operador de cruce aquí descrito se realiza de forma igual para todos los individuos. Sin embargo, existe variantes adaptativas de estos operadores [Angeline 1996] en las que el propio algoritmo se modifica, así como numerosas variantes que tienen como base este algoritmo [Aguirre 1999] [Pereira 1999].

2.2.2.4.2 Reproducción

La reproducción simplemente es la copia de individuos en la nueva generación. Esta operación es asexual en el sentido en que se genera un individuo a partir de un individuo anterior.

Esta operación, junto con la de cruce, son las que se utilizan más frecuentemente, y entre ellas tiene una especial predominancia la de cruce. De hecho, el porcentaje de individuos nuevos generados a partir de cruces suele ser superior al 90%, mientras que el resto son generados mediante copias. El aumento del número de individuos generados por copias aumenta el peligro de predominancia de un individuo sobre el resto de la población, y que finalmente tras varias generaciones toda la población converja hacia ese individuo. Esta es una situación indeseable, puesto que se ha perdido por completo la diversidad genética que se tenía al principio y ello conlleva que la búsqueda en el espacio de estados sólo se lleve a cabo en una determinada zona, lo cual es lo contrario que se pretende.

El operador de cruce es por tanto el principal operador utilizado para la generación de los nuevos árboles y exploración del espacio de estados [Poli 1998].

2.2.2.4.3 Selección

Al igual que en el caso de los AA.GG., la selección de individuos debe de realizarse favoreciendo el escoger individuos sobre todo de los mejores de la población, pero permitiendo también que algunos de los peores sean escogidos. Dado que en esta operación no influye la forma de codificación de los individuos, los algoritmos que se utilizan son los mismos que para el caso de AA.GG., explicados anteriormente.

2.2.2.4.4 Mutación

El operador de mutación provoca la variación de un árbol de la población. Este operador suele usarse con probabilidad muy baja (menos que 0.1) antes de introducir un individuo en la nueva generación.

Existen dos tipos principales de mutación: mutación en la que se varía un solo nodo y mutación en la que se varía una rama entera del árbol.

En el primer caso, conocida por mutación puntual, la mutación actúa de la siguiente manera:

1. Se escoge un nodo al azar del árbol.
2. Se escoge al azar un nodo del conjunto de terminales o no terminales, del mismo tipo que el seleccionado, con el mismo número de hijos y de forma que sus hijos sean del mismo tipo.
3. Se intercambia el nodo antiguo del árbol por el nuevo, manteniendo los mismos hijos que el antiguo.

Dado que cada rama del árbol representa una solución a un subproblema y el no terminal que las une representa la forma de combinar esas soluciones, si se realiza este tipo de mutación sobre un elemento no terminal se estará provocando que las soluciones se combinen de distinta forma. Este tipo de mutación apenas se usa. La figura 2.23 ilustra un ejemplo de mutación de este tipo. En este ejemplo, se ha mutado el árbol cambiando el nodo destacado por otro del mismo tipo y mismo número de hijos.

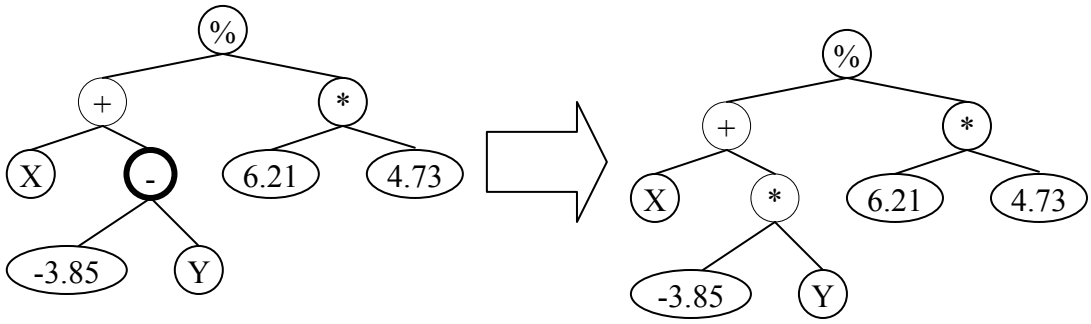


Figura 2.23. *Ejemplo de mutación puntual*

En el segundo tipo de mutación, se efectúan cambios mayores en el árbol. La operación es la siguiente: se escoge un nodo al azar del árbol, se elimina todo el subárbol que cuelga de ese nodo, se crea un nuevo subárbol del tipo y altura adecuados y se pone en su lugar.

Al cambiar una rama entera, lo que ahora se cambia del árbol es la forma de resolver el subproblema. En la figura 2.24 puede verse un ejemplo de un árbol sobre el que se va a aplicar esta operación de mutación. En él puede apreciarse un nodo destacado, que será el que sufra el proceso de mutación.

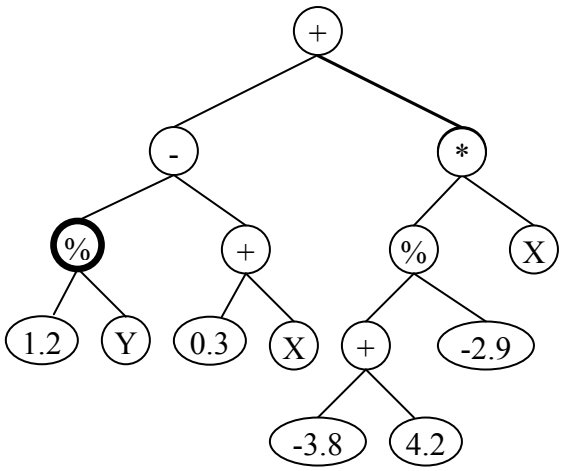


Figura 2.24. *Árbol sobre el que se realizará mutación*

Para realizar la mutación, será necesario eliminar el nodo seleccionado (y el subárbol que representa) y crear un subárbol nuevo para insertarlo en ese lugar. Para no violar la restricción de altura, el subárbol nuevo deberá tener una altura máxima de 3 (figura 2.25).

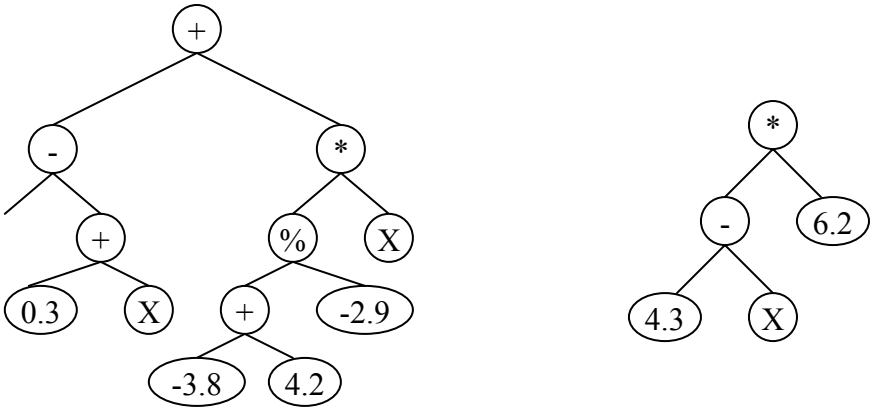


Figura 2.25. *Árbol de mutación y subárbol nuevo generado*

Finalmente, para terminar el proceso, se coloca el subárbol en el hueco dejado por el nodo eliminado (figura 2.26).

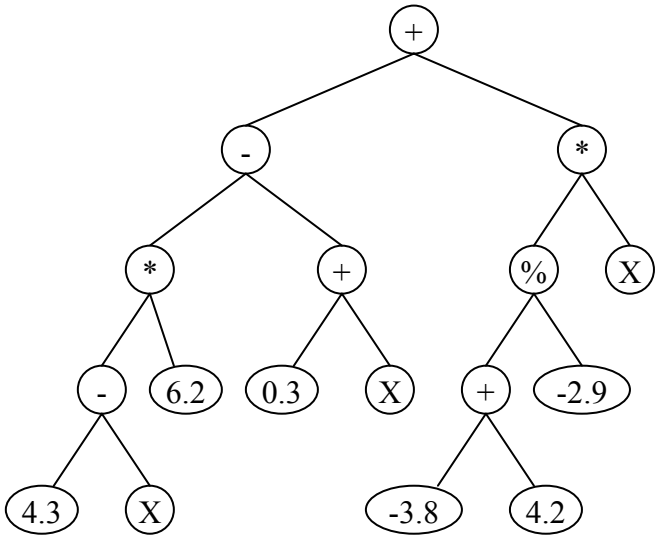


Figura 2.26. *Árbol resultante de la mutación*

El operador de mutación provoca en ese individuo un salto en el espacio de estados, comenzando una búsqueda distinta en otra zona. La mayoría de las mutaciones son destructivas, es decir, el individuo empeora, y por eso se utilizan con una probabilidad muy baja, para conseguir variedad genética. Existen estudios sobre la evolución sin el uso de cruces, en los que la mutación juega un papel fundamental [Chellapilla 1997], en los que se utilizan distintos tipos de mutaciones, pero los resultados siguen siendo peores que utilizando cruces.

2.2.2.5 Evaluación

Al igual que en los AA.GG., es necesario medir la bondad de cada individuo para realizar la ejecución del proceso evolutivo. De esto se encarga la función de ajuste, que cuantifica la bondad del fenotipo representado por el árbol genético. Existen diversas formas de ajuste, que son las mismas que las explicadas en la sección 2.2.1.4 para el caso de AA.GG. y, por lo tanto, son igualmente válidas para PG.

Como se ha explicado anteriormente, uno de los problemas de la PG es el crecimiento excesivo de los árboles para protegerse a sí mismos de los efectos nocivos de los operadores genéticos de cruce y mutación. Para evitar este problema, puede incluirse un factor de parsimonia en el cálculo del ajuste [Soule 1997] [Soule 1998]. Esta técnica se puede usar para reducir la complejidad del cromosoma que está siendo evaluado, y funciona mediante la penalización en el ajuste del individuo i de la siguiente forma:

$$f(i) = P(i) + \alpha \cdot s_i$$

Donde P(i) es una medida de la bondad del individuo (en este caso, peor cuanto más positivo), α es el nivel de parsimonia y s_i es el tamaño (número de nodos) del individuo. Con este coeficiente se está penalizando el número de nodos de un árbol, y su valor máximo suele ser de 0´1. Con este valor, se necesitará que el árbol tenga 10 nodos para incrementar en una unidad el valor de ajuste. Sin embargo, un valor tan alto ya es muy dañino en la evolución, y se suelen tomar valores menores (0.05, 0.01, etc), dependiendo del rango de valores en los que se espera que estén los ajustes de los individuos.

Dado que es un valor que se suma al valor de ajuste, que se quiere minimizar, constituye una penalización, lo cual provoca que el proceso evolutivo busque árboles sencillos que permitan resolver el problema en cuestión.

2.2.2.6 Parámetros

Ya que el funcionamiento de la PG es similar al de los AA.GG., los parámetros que controlaban el funcionamiento en éstos también realizan la misma función en la PG. Por lo tanto, los parámetros de tamaño de población, tasa de cruces, probabilidad de mutación, etc., explicados anteriormente, también están presentes aquí. Sin embargo, y dada la distinta naturaleza de la forma de codificación que tiene la PG, que ha obligado a realizar modificaciones a los operadores genéticos, surge una serie de parámetros especiales de PG, que son los siguientes:

- Altura máxima del árbol. Indica la altura máxima que van a poder alcanzar los árboles durante el proceso evolutivo. Este parámetro se ajusta también de forma proporcional a la complejidad del problema, de forma similar al anterior: un tamaño de población elevado exige elevar la altura máxima, pues, si no, existirá un gran número de individuos repetidos en la población, con predominio de uno de ellos, y los problemas que ello acarrea (fundamentalmente, pérdida de variedad genética y convergencia prematura del algoritmo). Para ajustar este parámetro y el anterior, y valorar la complejidad del problema, es conveniente tener en cuenta el número de elementos que tienen los conjuntos de elementos terminales y no terminales, así como el número de variables y analizar el problema a solucionar.
- La altura máxima que tendrán los árboles iniciales.

- La altura máxima que tendrán los subárboles creados por una mutación de subárbol. Para una de estas mutaciones, el subárbol generado tendrá una altura máxima que será la mínima entre este parámetro y la altura permitida por la restricción de altura máxima en el nodo seleccionado.
- Algoritmos de selección, mutación y creación. Sin embargo, y al contrario de lo que pasaba con los AA.GG., el algoritmo de cruce suele ser fijo y utilizarse siempre el mismo.
- En los cruces, se utiliza también una probabilidad de selección de nodo no terminal, que se usa para escoger nodos del árbol: con esa probabilidad se escogerá un no terminal antes que un terminal ([Koza 1992], [Angeline 1996]).
- El coeficiente de "parsimonia" para penalizar árboles grandes.

2.2.2.7 Aplicaciones

La programación genética es una técnica muy versátil y adaptable, y ha sido utilizada en una gran variedad de campos. Con seguridad su aplicación más utilizada ha sido en el campo de la regresión simbólica: el objetivo es encontrar expresiones que relacionen unas variables de entrada con salidas deseadas. En este campo, la programación genética ha hallado expresiones que relacionan magnitudes físicas que son tan buenas, o incluso mejores, que las existentes hasta aquel momento en el campo [Babovic 2001a] [Babovic 2001b]. También ha sido capaz de redescubrir relaciones matemáticas ya existentes, así como relaciones nuevas. Sin embargo, esta técnica no se limita a inducir expresiones matemáticas, sino que su capacidad para construir árboles la convierten en una poderosa técnica de optimización de árboles, y todo lo que pueda ser representado como tales. Algunos campos en los que ha sido aplicado son:

- ✓ Predicción en series financieras. Un ejemplo es el artículo publicado por Butler en 1995, en el que se utiliza la programación genética para generar reglas para un sistema experto con el objetivo de realizar apuestas en carreras de caballos [Butler 1995]. Un año antes, ya se había publicado otro artículo en el que se había utilizado esta técnica con el mismo fin, apostar en carreras de caballos, con buenos resultados [Perry 1994].

- ✓ Modelización de series en general, no solo financieras [Rivero 2005]. Howard Oakley ha utilizado programación genética para modelizar series caóticas basadas en las ecuaciones de Mackey-Glass en datos verdaderos [Oackley 1994a] [Oackley 1994b].
- ✓ Diseño de filtros. En los artículos de Oakley de los años 1993 y 1994 se compara el método de búsqueda con heurística (algoritmo genético convencional) con la programación genética para construir un filtro que elimine el ruido que se obtienen en los datos que se muestrean [Oakley 1993] [Oakley 1994a]. La pila de filtro que se obtuvo usando programación genética apareció como la respuesta apropiada. Actualmente se usa en una aplicación de Sistema Láser.
- ✓ Robótica. En este campo las aplicaciones han sido muy variadas. En el trabajo de Spencer de 1994, se describe como ha sido utilizado para generar automáticamente un programa que permita a una criatura de 6 patas andar y arrastrarse, con un operador adicional de perturbación [Spencer 1994]. En el campo de la robótica, la programación genética ha sido muy utilizada en el control de robots, pero también en el comportamiento (por ejemplo, en los trabajos de Reynolds se utilizó para evitar obstáculos [Reynolds 1992] [Reynolds 1994]) y en planificación o incorporación del dominio en el comportamiento de los robots.
- ✓ Reconocimiento de patrones y clasificación. Usando patrones del mundo real, se consiguieron clasificadores que formaran propiedades de los datos, con resultados mejores que los obtenidos con otros medios, como la utilización de árboles ID-3, o el uso de un perceptrón multicapa con un algoritmo de aprendizaje de *backpropagation*. Un trabajo teórico es el publicado de Tackett en 1994, en el que se estudia un problema de clasificación utilizando dos estructuras toroidales interconectadas como dos eslabones de una cadena [Tackett 1994]. El problema era clasificar una serie de puntos dados en un espacio tridimensional para ver si pertenecen a la estructura. La distribución de los puntos no puede ser linealmente separable por una regla perceptrón, ni pueden ser cubiertos los puntos por conos o hiperconos, ni encerrados por un par de funciones base radiales.

- ✓ Redes neuronales. Se ha aplicado la programación genética tanto para el diseño de redes neuronales [Gruau 1992] [Gruau 1994], como para la creación de nuevos algoritmos de aprendizaje [Segovia 1997].
- ✓ Arte. La programación genética se ha aplicado para la generación de imágenes y música de forma semiautomática, con la participación de humanos que evalúen la creación. Sin embargo, en la creación musical la participación humana ha sido posible automatizarla en gran medida [Johanson 1997] [Johanson 1998].
- ✓ Análisis de bases de datos. Se ha utilizado para distintos objetivos, como por ejemplo construcción de respuestas, clasificación o inducción de reglas, en el campo de la minería de datos [Freitas 1997].

Estos son solo algunos ejemplos de aplicaciones, pero existen innumerables campos de trabajo abiertos, como son creación de métodos de programación, aplicaciones en el procesamiento del lenguaje natural, en realidad virtual, síntesis de circuitos analógicos y digitales, algoritmos de compresión, procesado de imágenes [Rivero 2004], etc.

CAPÍTULO

3 ESTADO DE LA CUESTION

3.1 Generación de RR.NN.AA. mediante CE

El desarrollo de RR.NN.AA. es un tema que ha sido extensamente tratado con técnicas muy diversas. El mundo de los algoritmos evolutivos no es una excepción, y prueba de ello es la gran cantidad de trabajos que se han publicado en este aspecto usando diversas técnicas [Holland 1975] [Schwefel 1995] [Koza 1992] [Yao 1999] [Nolfi 2002a] [Nolfi 2002b] [Ritchie 2003] [Cantú-Paz 2005] [Kim 2005] [Rivero 2006a] [Rivero 2006b] [Davoian 2006] [Seys 2006] [Guillen 2007] [Raudys 2007]. Estas técnicas siguen la estrategia general de un algoritmo evolutivo: una población inicial de diferentes tipos de genotipos, cada uno de ellos codificando distintos parámetros (típicamente, los pesos de las conexiones, la arquitectura de la red y las reglas de aprendizaje, de forma separada o conjunta) es creada de forma aleatoria. Esta población es evaluada para determinar la bondad de cada individuo. Posteriormente esta población se hace evolucionar por medio de distintos operadores genéticos (replicación, cruce, mutación, etc.) de forma repetida hasta que se cumpla un determinado criterio de terminación (como por ejemplo, que se ha obtenido un individuo lo suficientemente bueno, o que se ha alcanzado un número máximo de generaciones predeterminado).

En general, el mundo de la generación de RR.NN.AA. mediante algoritmos evolutivos se divide en tres grandes grupos: evolución de pesos, de arquitecturas y de reglas de aprendizaje. La evolución de los pesos de las conexiones supone una aproximación adaptativa al entrenamiento, especialmente en aquellos entornos en los que se utilizaban algoritmos de entrenamiento basados en la minimización de gradiente, los cuales presentan grandes dificultades. La evolución de las topologías permite a las redes adaptar sus topologías a las diferentes tareas designadas sin la intervención humana, y por lo tanto son una aproximación al diseño automático de RR.NN.AA. dado

que se desarrollan ambos, estructuras y pesos a la vez. La evolución de las reglas de aprendizaje puede ser vista como un proceso de "aprender a aprender" por parte de las RR.NN.AA. en el cual la adaptación de las reglas se obtiene por medio de la evolución. Es también una forma de obtener nuevas reglas de aprendizaje.

3.1.1 Evolución de los pesos

La evolución de los pesos parte de una red con una topología ya fijada. En este caso, sólo preocupa el problema de establecer, mediante el entrenamiento, los pesos de las conexiones de la red, y este proceso generalmente se plantea como un problema de minimización del error de la red. Tomado, por ejemplo, como función de ajuste del individuo el Error Cuadrático Medio entre las salidas deseadas y las obtenidas por la red. La mayoría de los algoritmos de entrenamiento, como el algoritmo de retropropagación del error (*backpropagation*, en adelante BP) [Rumelhart, 1986], están basados en minimización del gradiente. Esto ha hecho que tengan varios inconvenientes [Sutton, 1986] [Whitley, 1990], el principal es que con bastante frecuencia el algoritmo se queda estancado en un mínimo local de la función de error y es incapaz de encontrar un mínimo global si la función de error es multimodal o no diferenciable (o ambas cosas). Una forma de superar estos problemas es realizar el entrenamiento mediante un algoritmo evolutivo [Whitley, 1990], es decir, formular el proceso de entrenamiento como la evolución de los pesos de las conexiones en el entorno definido por la arquitectura de la red y la tarea a resolver.

La aproximación evolutiva al entrenamiento de los pesos consiste en dos grandes fases. La primera se basa en decidir la representación de los pesos de las conexiones, es decir, si tendrán la forma de cadenas de valores binarios o no. La segunda fase es en sí misma el proceso evolutivo en el cual se ajustan los valores de estos pesos. Los algoritmos genéticos tradicionales [Holland, 1975] [Goldberg, 1989] utilizan una forma de codificación de los genotipos en forma de cadenas de valores binarios. De esta forma, han surgido muchos trabajos que codificaban los valores de los pesos mediante una concatenación de los valores binarios que los representan [Whitley, 1990] [Srinivas 1991] [Garis 1991] [Janson 1993]. La gran ventaja de esto radica en su simpleza y generalidad, es decir, es muy sencillo y rápido aplicar los operadores de cruce uniforme y mutación clásicos sobre una cadena de valores binarios. El inconveniente de utilizar este tipo de codificación es el problema de la permutación. Este problema se plantea al

considerar que el orden en el que se toman los pesos en el vector provoca que redes equivalentes puedan corresponderse con individuos totalmente diferentes, con lo que el operador de cruce se vuelve muy ineficiente. Es decir, si el operador de cruce se aplica a dos redes funcionalmente idénticas, pero estructuralmente diferentes, se obtendrán descendientes no útiles.

Lógicamente, también ha surgido la codificación de los valores de los pesos en forma de concatenación de números reales, cada uno de ellos asociado a un peso determinado [Menczer 1992] [Greenwood 1997] [Fogel 1995] [Fogel 1997] [Yan 1997] [Dorado, 1999]. Mediante operadores genéticos diseñados para trabajar con este tipo de codificaciones, dado que los existentes para cadenas de bits no pueden ser usados aquí, varios resultados [Montana 1989] [Bartlett 1990] muestran que este tipo de codificación produce mejores resultados que BP y con mayor eficiencia y escalabilidad. Otros trabajos en los que se hacen evolucionar vectores de números reales que codifiquen pesos utilizan técnicas de programación evolutiva (PE) o estrategias evolutivas (EE), y están pensadas para hacer optimización continua [Topchy 1997] [Greenwood 1997] [Sarkar 1997].

Sin embargo, un inconveniente que presenta la codificación real frente a la continua es que el espacio de búsqueda es extremadamente más grande, dado que este tipo de codificación es más preciso que con cadenas de valores binarios [Michalewicz 1996].

También presenta el mencionado problema de la permutación. Para evitar el problema de la permutación, han surgido algunos trabajos en los que se hace evolucionar los pesos sin codificarlos como un individuo, sino que es la propia red la que es un individuo de la población. Esto se ha utilizado para hacer evolucionar, por ejemplo, redes tipo MLP [Castillo 2001] o RBF [Rivas 1999].

3.1.2 Evolución de las arquitecturas

Por su parte, la evolución de las arquitecturas incluye la generación de la estructura topológica; es decir, de la topología y la conectividad de las neuronas, y la función de transferencia de cada neurona de la red. La arquitectura de una red es de suma importancia para poder aplicar con éxito la RNA puesto que cada arquitectura tiene un impacto muy significativo en la capacidad de proceso de la red. De esta forma, por una parte, una red con pocas conexiones y función de transferencia lineal puede no

ser capaz de resolver un problema que otra red con otras características (distinto número de neuronas, conexiones o tipos de funciones) sí podría resolver. Por otra parte, una red con un número muy elevado de conexiones y nodos no lineales podría sobreentrenarse y aprender el ruido presente en el entrenamiento como parte del mismo, sin llegar a discriminarlo, y no llegar a tener una buena capacidad de generalización [Yao 1999]. Por tanto el diseño de una red es algo crucial, y esta tarea la realizaba clásicamente un experto humano en función de su propia experiencia, a base de "ensayo y error", probando con un conjunto distinto de arquitecturas. El diseño automatizado de arquitecturas ha sido posible gracias a la aparición de algoritmos constructivos y destructivos [Frean 1990] [Sietsma 1991]. En grandes líneas, un algoritmo constructivo comienza con una red mínima (con un número pequeño de capas, neuronas y conexiones) y sucesivamente añade nuevas capas, nodos y conexiones si son necesarios, durante el entrenamiento. Por su parte, un algoritmo destructivo realiza la operación contraria, es decir, comienza con una red máxima y elimina capas, nodos y conexiones innecesarias durante el entrenamiento. Sin embargo, estos métodos (basados en algoritmos *Hill Climbing* o de ascenso de colinas) son bastante susceptibles de caer en un óptimo local [Angeline 1994].

El diseño de una arquitectura óptima para una RNA puede ser formulado como un problema de búsqueda de arquitectura donde cada punto representa una arquitectura. Dado este espacio de búsqueda, existen varias características que hacen a los algoritmos evolutivos mejores candidatos para realizar esta búsqueda que los algoritmos constructivos y destructivos mencionados anteriormente [Miller 1989]:

1. El espacio de búsqueda es muy amplio, puesto que el número posible de nodos y conexiones no tiene límites.
2. El espacio de búsqueda no es diferenciable, puesto que los cambios en el número de nodos o conexiones son discretos y pueden tener un efecto discontinuo en el comportamiento de la red.
3. El espacio de búsqueda es complejo y presenta ruido, puesto que el paso de una arquitectura a su correspondiente comportamiento es indirecto, muy epistático y dependiente del método de evaluación utilizado.
4. El espacio de búsqueda es engañoso, puesto que arquitecturas similares pueden exhibir diferentes comportamientos.

5. El espacio de búsqueda es multimodal, puesto que diferentes arquitecturas pueden exhibir comportamientos similares.

Para desarrollar RR.NN.AA. mediante un algoritmo evolutivo, es necesario decidir cómo codificar una red en el genotipo de forma que sea utilizable por los operadores genéticos [Dorado, 1999] [Nolfi 2002b] [Cantú-Paz 2005]. Para ello, han surgido diferentes formas de codificación de RR.NN.AA.

3.1.2.1 Métodos de codificación directa

En la primera forma de codificación, codificación directa, hay una correspondencia de uno a uno entre los genes y la representación fenotípica [Miller 1989]. La forma de codificación más típica consiste en una matriz $C=(c_{ij})$ de tamaño NxN que representa una arquitectura de N nodos, donde c_{ij} indica la presencia o ausencia de conexión entre el nodo i al j. Se podría usar $c_{ij}=1$ para indicar conexión y $c_{ij}=0$ para indicar ausencia de conexión. De hecho, c_{ij} podría tomar valores reales en lugar de booleanos para representar el valor del peso de la conexión de la neurona "i" a la "j" y, de esta forma, hacer evolucionar simultáneamente arquitectura y conexiones [Marin 1993] [Alba 1993] [Kothari 1996]. Las restricciones que se requieran en las arquitecturas pueden ser fácilmente incorporadas en este esquema de representación. Por ejemplo, una red *feed-forward* tendría coeficientes no nulos solamente en el triángulo superior derecho de la matriz.

Estos tipos de codificación son generalmente muy sencillos y fáciles de implementar. Sin embargo, tienen una gran cantidad de inconvenientes. Uno de ellos, por ejemplo, es la escalabilidad. Dado que el tamaño del genotipo es proporcional a la complejidad del genotipo correspondiente, el espacio de búsqueda del proceso evolutivo crece exponencialmente con el tamaño de la red y con ello el tiempo computacional necesario [Kitano, 1990]. Una forma de reducir el tamaño de las matrices es usar el conocimiento del dominio. Por ejemplo, si se sabe que dos capas estarán completamente conectadas en una red *feed-forward*, su arquitectura podría ser codificada con sólo el número de capas ocultas y el número de nodos en cada capa.

Otro problema de las formas de codificación directa es la imposibilidad de codificar estructuras repetidas (como una red compuesta por varias subredes con una conectividad local similar) de una forma compacta. De hecho, en estos tipos de codificaciones los elementos que se referencian repetidamente a nivel de fenotipo

(como puede ser el caso de una neurona) deben de repetirse a nivel de genotipo. Esto no sólo afecta a la longitud del genotipo, sino también a la capacidad de evolucionar de los individuos. Un fenotipo con estructuras referenciadas múltiplemente implica que su genotipo tendrá esas estructuras repetidas a lo largo del mismo, con lo que cambios producidos durante el proceso evolutivo en una de estas estructuras deberán de ser redescubiertos individualmente en cada uno del resto de estructuras repetidas.

Por último, otro problema de este tipo de codificación es la permutación. Varias redes distintas pero funcionalmente equivalentes que tengan un orden distinto en sus nodos ocultos tendrán una representación genotípica distinta. Por tanto, la posibilidad de producir una red que ofrezca un buen comportamiento como resultado del cruce de las mismas es muy bajo. Por esta razón, algunos investigadores han evitado realizar cruces en este tipo de codificación, y se han basado solamente en mutaciones en la evolución de arquitecturas [Yao 1997] [Yao 1998]. Existen trabajos en los que se propone una forma de codificación en la cual se evita este problema de permutación, pero con pocos y no muy buenos resultados experimentales [Thierens 1996].

3.1.2.2 Métodos de codificación indirecta

En contraposición con los tipos de codificación directa, existen los tipos de codificación indirecta. En ellos, con el objetivo de reducir la longitud de los genotipos, solamente algunas características de una arquitectura se codifican en el cromosoma. Dentro de este tipo de codificación existen varios tipos de representaciones, de naturaleza tan distinta como basadas en parámetros, fractales, etc.

En primer lugar hay que mencionar las representaciones paramétricas. En ellas la red puede ser representada por un conjunto de parámetros como el número de capas ocultas, el número de nodos en cada capa, el número de conexiones entre dos capas, etc. Existen varias formas de codificar estos parámetros en un cromosoma [Harp 1989] [Harp 1990] [Dodd 1991] [Hancock 1990]. A pesar de que la representación paramétrica puede reducir la longitud del cromosoma, el algoritmo evolutivo hace una búsqueda en un espacio limitado dentro de todo el posible espacio de búsqueda que representa todas las posibles arquitecturas. Por ejemplo, si se codifican solamente el número de nodos ocultos presentes en la capa oculta, se asume una red *feed-forward* con una capa oculta. En general este tipo de representación es más adecuada cuando se sabe qué tipo de arquitectura se está buscando.

Otro tipo de codificación no directa se basa en un sistema de representación en forma de reglas gramáticas [Vonk 1995] [Yao 1995]. En este sistema la red está representada por un conjunto de reglas las cuales dan lugar a una matriz que representa la red. Estas reglas tienen forma de reglas de producción, con una parte antecedente y otra consecuente. Cada parte izquierda de las reglas (excepto el inicio) es, a su vez, un elemento de la parte derecha de otra regla, y da lugar a otra submatriz. De esta forma, a medida que se aplican las reglas, la matriz que conforma la red va aumentando de tamaño hasta llegar a su tamaño final. Este método ha generado buenos resultados [Kitano, 1990]. Sin embargo, esto tiene varias limitaciones. Este método necesita que se predefina el número de pasos en los que se aplicarán reglas. Además, no permite la existencia de reglas recursivas, y una representación genotípica compacta no implica una representación fenotípica compacta, es decir, una arquitectura compacta.

Otro tipo de codificaciones, más inspirado en el mundo de la biología, se denominan "métodos de crecimiento" (*growing methods*). En ellos el genotipo ya no codifica directamente una red, sino que contiene un conjunto de instrucciones. La decodificación del genotipo consistirá en la ejecución de dichas instrucciones, lo cual provocará la construcción del fenotipo [Nolfi, 1994] [Husbands, 1994]. En estos modelos, una red estaba representada en un espacio de dos dimensiones en el que están dispuestas las neuronas y los axones. Durante un proceso de crecimiento, estos axones van creciendo y bifurcándose, y cuando una de estas bifurcaciones de un axón de una neurona alcanza a otra neurona distinta, se establece una conexión entre ambas neuronas. Después del proceso, las neuronas aisladas (sin conexión con otras, y por lo tanto no funcionales) se eliminan. El crecimiento y bifurcación de un axón se da sólo si la activación de una neurona es mayor que un umbral establecido genéticamente. Este mecanismo se basa en la idea de que la información sensorial que viene del entorno tiene un papel crítico en la maduración de la conectividad del sistema nervioso y, más específicamente, que el proceso de maduración es sensible a la actividad de cada neurona por separado [Purves 1994]. Este método permite al proceso evolutivo seleccionar topologías de redes que son adecuadas a la tarea a realizar por la red. De hecho, si algunos aspectos de la tarea pueden variar durante el proceso evolutivo, los genotipos evolucionados muestran una habilidad para convertirse en estructuras fenotípicas diferentes que se adapten a las condiciones actuales.

En los organismos biológicos, el sistema nervioso se crea en tres fases: la génesis y proliferación de diferentes clases de neuronas mediante la duplicación celular y su diferenciación, la migración de las neuronas hacia su destino final, y el crecimiento de neuritas (axones, dentritas). Los métodos de crecimiento explicados anteriormente caracterizan sólo la última de estas tres fases. Sin embargo, se han desarrollado una serie de intentos de incluir otros aspectos de este proceso en los experimentos basados en computación evolutiva. En general, estas formas de codificación se denominan "codificación celular" (*cellular encodings*).

En varios trabajos se extiende el modelo descrito ([Nolfi, 1994]) añadiéndole una división de celdas y una etapa de migración a la etapa ya existente de crecimiento axonal [Cangelosi, 1994] [Dellaert, 1994]. En este caso el genotipo es una serie de reglas que rigen el proceso de la división de celdas (una celda sencilla es reemplazada por dos celdas hijas) y migración (las nuevas celdas se pueden mover en un espacio 2D). Por tanto, el proceso de decodificación del genotipo al fenotipo comienza con una sola celda que sufre una serie de procesos de duplicación y migración, produciendo como resultado un conjunto de neuronas dispuestas en un espacio 2D. Estas neuronas hacen crecer sus axones y establecen conexiones hasta que se forma un controlador neuronal.

En otro trabajo representativo se propone un esquema de codificación de RR.NN.AA. basado también en la duplicación celular y el proceso de diferenciación [Gruau, 1994]. El proceso de decodificación comienza con una célula que sufre un número de procesos de duplicación y transformación para dar lugar a una red de neuronas artificiales completa. El genotipo será también un conjunto de reglas que rigen los procesos de división de celdas (una celda es reemplazada por dos celdas hijas) y transformación de las mismas (nuevas conexiones pueden ser añadidas y los pesos de las conexiones pueden ser cambiados). Por lo tanto, en este modelo los enlaces se establecen durante el proceso de duplicación celular.

Las instrucciones contenidas en el genotipo se representan en una estructura de árbol binario con lo que, en la práctica, se está usando el algoritmo de PG. Durante el proceso de decodificación, este árbol es recorrido comenzando por la cima del mismo y siguiendo cada ramificación. El nodo inicial representa la celda inicial que, tras los procesos de duplicación y diferenciación, dará lugar a la red. Cada uno de los nodos del árbol representa las operaciones que deberían ser aplicadas a la celda correspondiente y

los dos subárboles del mismo especifican las operaciones que serán aplicadas a las dos celdas hijas. De esta forma se construye la red, recorriendo el árbol y aplicando los operadores correspondientes. En el árbol los operadores terminales representan celdas que no sufrirán ningún otro efecto de duplicación. En algún trabajo, además, se considera el caso de genotipos formados por árboles en los cuales los nodos terminales de uno pueden apuntar a otros árboles [Gruau, 1994]. Este mecanismo permite que el fenotipo tenga subredes neuronales repetidas mediante la reutilización de la misma información genética. Este método de codificación tiene dos ventajas. La primera de ellas es que genotipos compactos pueden producir redes bastante complejas. La segunda ventaja es que el proceso evolutivo puede explotar el hecho de que un fenotipo pueda hacer uso repetidamente de subestructuras que hayan sido codificadas en una única parte del genotipo. Este método particular se denomina ADNS (*Automatic Definition of Neural Subnetworks*).

Dentro de las codificaciones no directas, es necesario nombrar otro tipo, basado en el uso de subconjuntos fractales de un plano [Merrill 1991]. Según este estudio, la representación fractal de las arquitecturas es biológicamente más plausible que la representación en forma de reglas. Se usan tres parámetros que toman valores reales para especificar cada nodo en una arquitectura: un código de borde, un coeficiente de entrada y un coeficiente de salida. Este método podría ser considerado como más cercano a un método de codificación directa que a uno indirecto.

Por último, y dentro de los métodos de codificación indirecta, existen otros métodos que se diferencian enormemente de los ya descritos. Andersen describe una técnica en la que cada individuo de la población representa un nodo oculto, en lugar de una arquitectura [Andersen 1993]. La red se construye capa a capa, es decir, las capas ocultas se añaden una por una si la arquitectura actual no puede reducir el error de entrenamiento por debajo de cierto umbral. Cada capa oculta se construye automáticamente a través de un proceso evolutivo que emplea un algoritmo genético. Este método tiene la limitación de que sólo puede construir redes *feed-forward*, y, además, suelen surgir varios nodos con una funcionalidad muy similar, con lo que esta redundancia debería ser eliminada. Otros trabajos parecidos son tratados por Smith con similares resultados [Smith 1994] [Smith 1997].

Sin embargo, a pesar de la existencia de una gran cantidad de métodos para hacer evolucionar redes mediante técnicas evolutivas, muchos autores se plantean la cuestión

de hasta qué punto el operador de cruce es útil [Castillo 2002]. Aquellos autores que están a favor de su uso, argumentan que debería de ser usado siempre que la representación de las soluciones permita que se formen "bloques de construcción" [Holland 1975], es decir, esquemas que representan una buena solución a una parte del problema a resolver y que pueden ser tratados independientemente y recombinados mediante este operador. Por contra, otros autores afirman que, si bien el operador de cruce ayuda en la búsqueda, esto no quiere decir que esté efectivamente recombinando bloques constructivos, sino que sus efectos son similares a los que provocaría un operador de mutación que provocase grandes cambios (macromutación) [Land 1998]. Un ejemplo de operador de macromutación es el descrito por Castillo, que sustituye una neurona oculta aleatoria por otra, inicializada con pesos aleatorios [Castillo 1999] [Castillo 2000].

En general todos estos métodos para evolucionar arquitecturas sólo realizan ese trabajo, es decir, no ajustan los valores de los pesos de las conexiones. Estos valores tienen que ser aprendidos después de haber encontrado una buena arquitectura para la red. Un gran inconveniente de la evolución de arquitecturas sin pesos es que la evaluación de las mismas presenta mucho ruido. En otras palabras, el ajuste de una arquitectura es ruidoso e impreciso porque el ajuste del genotipo (arquitectura sin pesos) se realiza mediante una aproximación del ajuste de un fenotipo (arquitectura con pesos). En este proceso hay dos fuentes de ruido:

- La primera es la inicialización aleatoria de los pesos, puesto que diferentes inicializaciones darán lugar a diferentes resultados de entrenamiento. Por lo tanto, el mismo genotipo tendrá diferentes valores de ajuste dependiendo de las diferentes configuraciones iniciales de los pesos.
- De la misma forma, diferentes algoritmos de entrenamiento darán lugar a diferentes valores de ajuste a pesar de haber podido comenzar con la misma configuración inicial de pesos.

Respecto a la inicialización de los pesos, esta es clave para obtener una convergencia rápida en MLP, ya que, dependiendo del punto del espacio de búsqueda del que se parta (y ese punto está determinado por el conjunto de pesos) se obtendrán mejores o peores soluciones en el entrenamiento [Thimm 1995]. Existen diversos métodos de inicialización de pesos, siendo el más sencillo de ellos el hacer una

inicialización aleatoria [Kolen 1990]. Fahlman propuso, tras un estudio experimental, usar un rango inicial entre [-4.0, 4.0] y [-0.5, 0.5] dependiendo del problema [Fahlman 1988]. En otro trabajo se demuestra teóricamente que los nodos con pesos con valores altos son más propensos a sufrir saturación (los cambios que se realicen a dichos pesos no afectarán prácticamente a la salida del nodo) [Lee 1993]. Otros autores proponen métodos no aleatorios de inicialización [Denoeux 1993] [Kim 1997], que necesitan hacer un preprocesado de los patrones de aprendizaje o de los pesos mismos de la red previo a la aplicación del algoritmo de aprendizaje, lo cual incrementa el coste computacional.

Para reducir este efecto de ruido en la evaluación de una arquitectura, ésta debe de ser entrenada muchas veces con diferentes configuraciones iniciales aleatorias de pesos. La media de los resultados obtenidos tras cada entrenamiento es lo que se usa para estimar el ajuste medio de ese genotipo. Sin embargo, este método aumenta el tiempo de computación dramáticamente, con lo que el proceso evolutivo resulta muy ineficiente. En general, este problema se debe a que no existe una correspondencia uno a uno entre genotipo y fenotipo, es decir, un mismo genotipo puede dar lugar a una gran cantidad de distintas redes que exhiban comportamientos distintos.

Una forma de solucionar este problema es hacer evolucionar arquitecturas y pesos simultáneamente [Alba 1993] [Angeline 1994] [Gruau 1992] [Srinivas 1991] [Yao 1995] [Yao 1997] [Yao 1998]. En este caso, cada individuo de la población es una red totalmente especificado con sus valores de pesos. Dado que ahora hay una correspondencia de uno a uno entre genotipo y fenotipo, la evaluación del ajuste de los individuos es precisa.

Otra característica importante a tener en cuenta es que estos métodos en general evolucionan arquitecturas, o bien de forma conjunta arquitecturas y pesos. La función de transferencia de cada nodo de la arquitectura se asume que ha sido previamente fijada por un experto humano, y que es la misma para todos los nodos de una red (como mínimo para todos los nodos de una misma capa), a pesar de que la función de transferencia ha demostrado tener un impacto significativo en el funcionamiento de las redes [DasGupta 1992] [Lovell 1992]. Se han desarrollado pocos métodos que hagan evolucionar también la función de transferencia [Stork 1990] [White 1993] [Liu 1996] [Hwang 1997] [Sebald 1998] y, por tanto, incluyan esta dentro del genotipo, y estos han tenido poca repercusión en el mundo del desarrollo de RR.NN.AA. con CE.

Sin embargo, en un trabajo significativo en el que sí se hace evolucionar la función de transferencia de los nodos se utiliza un AG de dos capas para diseñar la arquitectura de una RNA y realizar su entrenamiento [Dorado, 1999]. Entre otras cosas, durante el proceso evolutivo se fijan los parámetros de los elementos de proceso, entre ellos la función de transferencia. Junto con el ajuste de los parámetros de la arquitectura y entrenamiento de la misma, en este trabajo se realiza también un diseño del conjunto de entrenamiento óptimo a partir de series temporales, complementando así los métodos existentes para la discriminación de variables de entrada.

Otro aspecto interesante es el uso de técnicas de búsqueda local en la evolución de los individuos, con lo que efectivamente se están utilizando algoritmos híbridos en los que se aplican mecanismos "lamarckianos" y basados en el "efecto baldwin". La teoría de Lamarck [Lamarck 1809], a pesar de haber sido posteriormente rechazada por los biólogos, establece que las características adquiridas durante la vida de un individuo se transmiten genéticamente a la descendencia. Aplicado a un algoritmo evolutivo, esto quiere decir que se puede aplicar una técnica de búsqueda local para refinar un individuo y hacer que este tienda a su óptimo local, y que esta modificación del individuo permanezca en su "código genético" [Merz 1997] [Ross 1999]. Por su parte, Baldwin [Baldwin 1896] [Waddington 1942] sugirió que si el aprendizaje ayuda a sobrevivir, aquellos organismos con más capacidad de aprendizaje tendrán más posibilidades de reproducirse, con lo cual los genes responsables del aprendizaje incrementarán su frecuencia en la población. De esta forma, la capacidad de aprender ayuda a los mecanismos a responder ante cambios de su entorno (aspectos que cambian demasiado rápido para que la evolución los asimile en el código genético), e incluso de forma indirecta provoca que esas adaptaciones acaben codificadas en el código genético. Aplicado a un algoritmo evolutivo, esto se realiza de una forma parecida al mecanismo "lamarckiano", mediante una búsqueda local para mejorar el valor de su función de evaluación, pero en este caso sin modificar el código genético del individuo [Hinton 1987] [Boers 1995]. Estas dos técnicas han sido aplicadas de una forma muy similar al desarrollo de RR.NN.AA. pero con resultados bastante dispares y dependientes del problema a resolver [Castillo 2001] [Whitley 1994]. Existen varios trabajos en los que se comparan estrategias baldwinianas y lamarckianas [Ku 1997] [Houck 1997] [Julstrom 1999]. En general se puede concluir que el uso de una estrategia "lamarckiana" implica una mejora en velocidad del algoritmo, pero con la

posiblidad de convergencia a un óptimo local, puesto que puede hacer que ciertos individuos pasen a dominar la población por la ventaja adquirida en cierto momento y continúen siendo los mejores individuos hasta el final de la simulación [Oliveira 1999] [Castillo 2001].

3.1.3 Evolución de la regla de aprendizaje

Otra aproximación interesante al desarrollo de RR.NN.AA. mediante CE es la evolución de la regla de aprendizaje. Esta idea surge porque un algoritmo de entrenamiento tiene distinto funcionamiento al ser aplicado en redes con arquitecturas distintas. De hecho, y dado que el conocimiento que se tiene "a priori" de la red suele ser bastante escaso, es preferible desarrollar un sistema automático para adaptar la regla de aprendizaje a la arquitectura y el problema a resolver [Castillo 2002].

Existen varias aproximaciones a la evolución de la regla de aprendizaje [Crosher 1993] [Turney 1996] [Baxter 1992], aunque la mayoría se basan sólo en cómo el aprendizaje puede modificar o guiar la evolución, y en la relación entre la evolución de la arquitectura y de los pesos de conexión. Realmente son pocos los trabajos que se centran en la evolución de la regla de aprendizaje en sí misma [Bengio 1990] [Bengio 1992] [Ribert 1994].

Uno de los enfoques más comunes se basa en ajustar los parámetros del algoritmo BP: tasa de aprendizaje y momento [Kim 1996] [Belew 1991]. Algunos autores proponen métodos en los que se usa un proceso evolutivo para encontrar estos parámetros mientras se deja la arquitectura constante [Patel 1996] [Kim 1996]. Otros autores, en cambio, proponen codificar estos parámetros del algoritmo BP junto con la arquitectura de la red dentro de los individuos de la población [Harp 1989] [Merelo 1993].

Debido a la complejidad que supone realizar una codificación de todas las posibles reglas de aprendizaje, es necesario establecer ciertas restricciones que simplifiquen dicha representación. De esta forma, se limitará también el espacio de búsqueda. Chalmers definió una regla de aprendizaje como una combinación lineal de cuatro variables y seis términos producto [Chalmers 1990]. Cada individuo de la población es una cadena binaria que codifica exponencialmente diez coeficientes más un término de escala. A partir de este trabajo han surgido otros posteriores llegando a resultados

similares [Crosher 1993] [Bengio 1992] [Baxter 1992], en los que se puede ver que la capacidad de aprendizaje de una RNA puede mejorarse mediante evolución.

3.2 PG basada en grafos

La estructura que toman las soluciones proporcionadas por al algoritmo de PG, como se ha descrito, adopta una forma de árbol. Esto permite resolver una gran cantidad de problemas distintos y permite encontrar soluciones que otras formas de codificación, como la utilizada en AA.GG. como cadenas de bits o números reales, no permiten obtener. A su vez, la codificación de problemas en forma de grafo permite resolver problemas que los árboles no han podido resolver. Es por esto que, desde poco después de la aparición de la PG, han surgido investigadores que han estudiado la posibilidad de utilizar grafos dentro de la PG para poder representar y resolver estos problemas.

Como primeras aproximaciones a la codificación con grafos, surgieron diversas soluciones que utilizan árboles con operadores especiales, con el objetivo de resolver problemas muy concretos, por ejemplo, para desarrollar autómatas de pila [Zomorodian 1995] o para desarrollar circuitos eléctricos. En este último campo, destacan varios trabajos en los que se utiliza PG clásica para desarrollar distintos tipos de circuitos eléctricos y filtros analógicos [Zan 2001] [Zan 2002] [Zan 2003]. Para ello, distintos operadores han tenido que ser creados para permitir a la PG representar estructuras tan complejas. Si bien los resultados son altamente satisfactorios, esta forma de codificación, utilizando este conjunto de operadores, es muy limitada y sólo permite resolver problemas en este ámbito.

En otras aproximaciones el uso de grafos mediante árboles se utiliza principalmente para el desarrollo de RR.NN.AA. mediante PG [Gruau 1992], [Gruau 1994] y [Luke 1996]. Estos trabajos utilizan los operadores del árbol de PG para ir creando grafos a medida que se ejecutan dichos operadores del árbol. De esta forma, existirán operadores para crear nuevos nodos, crear aristas entre nodos o invertir el sentido de una arista, por ejemplo. Estas codificaciones se denominan “cellular encoding” o “edge encoding”, y tienen algunos inconvenientes. En primer lugar, el fenotipo representado (es decir, el grafo que se va a obtener) es demasiado dependiente del orden de ejecución de los operadores del árbol. Un subárbol dentro de un individuo puede resultar en un subárbol totalmente diferente después de haber sido cruzado a otro individuo, con lo que es deseable utilizar un sistema de cruce que preserve mejor el

fenotipo en la operación de cruce. Además, este tipo de codificaciones producen un número muy alto de nodos interconectados, lo cual puede no ser muy deseable en muchos dominios.

Teller describe un sistema llamado PADO que utiliza una pila y discriminadores lineales para obtener programas paralelos utilizados en la clasificación de señales e imágenes [Teller 1996]. Todos estos métodos utilizan tipos especiales de paralelismo en el uso de grafos, y no se pueden considerar como una generalización natural de la PG al mundo de los grafos.

De una forma similar a este último trabajo, en otros trabajos se utilizan ya grafos como sistema de codificación, utilizando para ello una memoria indexada y una pila, que se usan para transferir datos entre nodos [Kantschik 1999a] [Kantschik 1999b]. Además, los nodos se dividen en dos partes, acción y ramificación. La parte de acción puede ser una constante o una función que se ejecutará cuando se llegue al nodo durante la ejecución del programa. Para realizar esta ejecución, esta función toma sus parámetros de la pila del sistema, y escribirá su resultado igualmente en la misma pila. Después, la parte de ramificación escogerá, bien a partir del contenido de la cima de la pila o de la memoria, qué nodo seguirá la ejecución entre los nodos a los que está conectado el nodo actual.

Otro sistema de PG en el que se utilizan grafos es un sistema llamado "Parallel Distributed Genetic Programming" [Poli 97]. En este sistema se utilizan grafos como forma de codificación, y estos se sitúan sobre una rejilla que contiene nodos activos e inactivos. Los nodos activos se conectan con nodos activos del nivel inferior, con la posibilidad de referenciar varias veces el mismo nodo para crear grafos, hasta llegar a las hojas. Se han definido nuevos operadores de cruce y mutación para el trabajo con estos grafos [Poli 97].

En una nueva aproximación al uso de grafos, denominada esta vez "linear-graph GP", se utilizaron nodos especiales que ejecutaban un conjunto de operaciones secuenciales y que terminaban en una ramificación condicional [Kantschik 2002]. Como resultado de esta ramificación, estos nodos apuntan a otros nodos del mismo tipo, pudiendo ser un nodo referenciado más de una vez. Si bien en este estudio se logra trabajar directamente con grafos, en principio este trabajo está pensado solo para

desarrollar programas de ejecución secuencial con forma de grafos, y no vale para resolver problemas más genéricos.

Posiblemente unos de los trabajos más representativos en este campo son los descritos por Teller [Teller 2000a] [Teller 2000b]. En ellos se describe un sistema que utiliza grafos como forma de codificación, en lo que denominan "Neural Programming". En este sistema ya no hace falta ningún tipo de estructura externa como pilas para evaluar los programas, sino que se evalúan directamente como si fuesen árboles. Además, estos trabajos utilizan un sistema de créditos en las conexiones entre los nodos para escoger mejor los que serán usados en la combinación entre individuos. Sin embargo, este sistema solo funciona con grafos de naturaleza matemática (es decir, con nodos de tipo aritmético, trigonométrico, etc.), puesto que el sistema ha sido pensado para el procesado de señales e imágenes. De todas formas, el sistema desarrollado es uno de los más completos existentes.

3.3 Consideraciones

La aplicación de las técnicas de CE en el campo de las RR.NN.AA. ha experimentado una gran evolución en su investigación desde sus orígenes, con un gran éxito en los resultados obtenidos. Como ya se ha explicado, la aplicación de estas técnicas se ha realizado con enfoques distintos.

En primer lugar, el conjunto de técnicas basadas en el entrenamiento de RR.NN.AA. tienen como objetivo solamente fijar los valores de los pesos de las conexiones de forma evolutiva. De esta forma se evitan los problemas que tiene la aplicación de los algoritmos de optimización basados en gradiente descendente. En este aspecto, algunos de los trabajos más significativos [Castillo 2001] [Fogel 1997] muestran mejores resultados que los obtenidos con el algoritmo BP. Sin embargo, esta familia de herramientas parten de una topología de red predeterminada, lo cual obliga al experto a realizar cierto trabajo para crearla, posiblemente aportando conocimiento previo que tenga del entorno, y, seguramente, tenga que repetir este proceso de diseño y entrenamiento para encontrar una arquitectura que ofrezca buenos resultados, con lo que su aplicación se aleja del objetivo de esta Tesis.

Por su parte, las técnicas de desarrollo automatizado presentan una serie de graves inconvenientes. El principal de ellos es el alto coste computacional que tienen debido a que para evaluar una arquitectura es necesario entrenarla, con lo cual evaluar una

población completa de individuos, siendo cada individuo una arquitectura distinta, y proceder a su evolución, se convierte en un proceso muy costoso computacionalmente. Este problema se agrava, puesto que la evaluación de cada arquitectura, es decir, su entrenamiento, como ya se ha explicado, es un proceso que tiene mucho ruido, principalmente provocado por el carácter estocástico del proceso, es decir, muy dependiente del componente aleatorio (sobre todo en la inicialización de los pesos) y del algoritmo de entrenamiento utilizado. Esto conlleva mucha variabilidad en los resultados que se obtienen al evaluar cada individuo. Para minimizar esta componente aleatoria, se debe repetir el proceso de evaluación de cada individuo varias veces con el objetivo de devolver la media de estos valores como el resultado del ajuste del individuo. Esta repetición del proceso de entrenamiento hace que la complejidad del proceso global se dispare, haciéndolo altamente costoso.

Por otra parte, muchas de las técnicas de desarrollo de arquitecturas parten de una topología ya definida por el experto, lo cual vuelve a convertir el proceso en dependiente del experto y de su nivel de experiencia en el campo. Además, la mayoría de ellas establecen muchas restricciones sobre las redes que generan; por ejemplo, que deben de tener una arquitectura clásica en capas, un número máximo de neuronas ocultas, etc.

Dentro del mundo del desarrollo automatizado de arquitecturas destacan los trabajos de selección de variables [Yang 1998] [Ozdemir 2001] y algoritmos de poda [Reed 1993], explicados más extensamente en la sección 7. Como puede verse en esa sección, estas técnicas son algunas de las que mejores resultados ofrecen. Sin embargo, adolecen del mismo defecto, es decir, requieren de un trabajo previo del experto. En general, esta situación se da muy a menudo, para obtener mejores resultados es recomendable (a veces necesario) incluir conocimiento del problema y realizar un trabajo previo sobre la red, esto es, establecer una arquitectura inicial, una limitación, etc.

Por su parte, respecto al uso de grafos dentro de la PG, también existe una serie de trabajos realizados en este tema, si bien la cantidad no es, ni de lejos, tan alta como los de entrenamiento y desarrollo de RR.NN.AA. mediante técnicas evolutivas. En general el uso de grafos ha sido menos estudiado y, en la mayoría de veces, el desarrollo ha sido realizado de una forma no genérica, es decir, para una aplicación particular en un campo concreto. Esto es lo que ocurre, por ejemplo, con los trabajos de Zan [Zan 2001] [Zan

2002] [Zan 2003], en los que se usan grafos para el desarrollo de circuitos eléctricos y filtros analógicos. Como ya se ha mencionado, uno de los trabajos más significativos es el realizado por Teller [Teller 2000a] [Teller 2000b], en el que la codificación en forma de grafos se realiza de una forma "nativa". Sin embargo, adolece del mismo defecto, es decir, que sólo puede usarse para la resolución de problemas en el ámbito para el que ha sido creado, en este caso el procesado de imágenes y señales.

CAPÍTULO

4 HIPÓTESIS

Las RR.NN.AA. tienen unos sistemas de aprendizaje que han permitido resolver una gran cantidad de problemas complejos en diversas facetas (clasificación, clusterización, regresión, etc.) [McCulloch, 1943] [Orchad, 1993] [Haykin, 1999]. Presentan características interesantes que las convierten en una técnica muy potente y han llevado a muchos investigadores a usarlas en una gran cantidad de entornos distintos [Rabuñal 2004] [Rabuñal 2005].

Sin embargo, su uso plantea una serie de problemas, principalmente en el proceso de desarrollo de las mismas. El desarrollo de RR.NN.AA. se puede dividir en varias fases: desarrollo de la arquitectura, entrenamiento, validación y test de la red resultante.

El desarrollo de la arquitectura se refiere al proceso de determinar cuántas neuronas tendrá una RNA, en cuántas capas estará estructurada y cómo estarán interconectadas entre sí. Esta etapa es crucial en el desarrollo de una red, puesto que la topología tiene una gran importancia en el funcionamiento que tendrá la misma, dado que diferentes topologías permiten distintos niveles de representatividad, y cada una de ellas conlleva distintos problemas. Por esta razón, dada la inexistencia de métodos de desarrollo automatizados de topologías, esta parte tradicionalmente la desarrollaba un experto humano que, basándose en su propia experiencia, determinaba la estructura que tendría la red. Esta es la parte más dependiente del experto humano que usualmente tiene que realizar varios diseños de arquitecturas hasta encontrar una que pueda ofrecer buenos resultados.

El entrenamiento se refiere al proceso de fijar los valores de los pesos de las conexiones de dicha arquitectura. Este proceso es realizado por un algoritmo de

entrenamiento. El más común de ellos es el denominado algoritmo de retropropagación del error (*backpropagation*, en adelante BP) [Rumelhart, 1986]. Sin embargo, a pesar de ser un proceso automatizado, esta etapa precisa de la supervisión de un experto para controlar su correcta aplicación. Además, los algoritmos de entrenamiento suelen tener una inicialización aleatoria de los pesos de las conexiones, lo cual provoca que su aplicación en una misma red pueda provocar resultados dispares. Por esta razón el experto se ve forzado a aplicar el algoritmo de entrenamiento varias veces hasta que los resultados son satisfactorios. Por su parte, el proceso de validación se refiere a la evaluación de la red con un conjunto distinto, con el objetivo de controlar el proceso de entrenamiento de la red, o bien de tomar decisiones acerca de las redes resultantes.

Finalmente, y una vez que se ha escogido una red entre todas las topologías distintas que se han probado y todos los conjuntos de pesos resultantes del entrenamiento de la misma, se procede a realizar un test, sobre un conjunto de patrones que la red no ha visto, con el objetivo de probar el funcionamiento de la red en casos reales.

Este funcionamiento de las RR.NN.AA., y dado que la arquitectura de una red es dependiente del problema a resolver, hace que el proceso de diseño de esta arquitectura generalmente se realice mediante un proceso manual basado en la experiencia, es decir, es el experto el que debe probar con varias arquitecturas distintas hasta que encuentra una que es capaz de ofrecer buenos resultados tras el proceso de entrenamiento. Este es un proceso lento, marcado por el hecho de que determinar la arquitectura es un proceso manual, a pesar de que recientemente se han desarrollado técnicas de creación de RR.NN.AA. de forma más o menos automatizada.

Sin embargo, se ha observado que estas técnicas de desarrollo de RR.NN.AA. de forma automatizada no funcionan todo lo bien que cabría esperarse. Como ya se ha explicado, estas técnicas tienen un alto coste computacional y, en muchas ocasiones, siguen requiriendo la participación del experto humano, con lo que no se ha conseguido una independencia total con el mismo, aparte de las limitaciones que imponen a las redes que son generadas.

Como ya se ha dicho, en esta Tesis se propone un sistema que realice todas las etapas de desarrollo de RR.NN.AA. de forma conjunta, y, lo que es más importante, sin

necesidad alguna de intervención por parte del experto humano. Además, las redes que se generan no tienen ninguna restricción respecto a la topología.

Para llevar a cabo esto, se utiliza la técnica de PG. La capacidad de representación que tiene esta técnica, muy superior a la de los AA.GG. tradicionales, hace que sea posible codificar estructuras más complejas con la misma. Sin embargo, y como se explica más adelante, la forma de codificación que tiene, a pesar de permitir la codificación de RR.NN.AA., requiere el uso de estructuras externas, con lo que la representación de una RNA no se realiza de forma totalmente directa. Esto se pretende solucionar mediante el uso de grafos como forma de codificación, con lo que una RNA sí puede ser codificada de forma directa, sin necesidad de otras estructuras externas. Ello conlleva que los operadores genéticos sean más eficientes y, por lo tanto, se espera que los resultados sean mejores en este caso.

Se pretende conseguir un sistema que reúna las siguientes características:

- Debe de ser totalmente independiente del experto.
- No debe de imponer ninguna restricción sobre la topología de las redes que genere.
- Debe de ser eficiente.

CAPÍTULO

5 MODELO PROPUESTO

En este trabajo se propone el uso de PG para el desarrollo de RR.NN.AA. Como ya se ha explicado, su uso se plantea desde dos puntos de vista: desde un punto de vista clásico, en el que se utiliza la PG tradicional sin modificar el algoritmo, y utilizando la PG con ciertas modificaciones para permitir una codificación en forma de grafos.

Además, en este trabajo se propone el uso de AA.GG. para mejorar este proceso. La motivación del uso de AA.GG. es utilizarlos para optimizar los valores de los pesos de las conexiones y, con ello, hacer el proceso más eficiente.

Para poder utilizar PG para resolver cualquier problema, es necesario especificar dos elementos fundamentales que permiten su aplicación:

- Conjuntos de terminales y funciones. Determinan qué operadores existirán como nodos del árbol (o grafo) que representa el genotipo.
- Función de ajuste. Determina cómo será evaluado cada individuo y devuelve un valor que cuantifica la bondad del mismo.

5.1 Conjuntos de terminales y funciones

Los conjuntos de operadores, bien sean terminales o funciones, dependerán del uso de árboles o grafos como forma de codificación. Además, si se emplean AA.GG. para optimizar los pesos, esto variará también los posibles operadores existentes.

5.1.1 Uso de árboles

El desarrollo de RR.NN.AA. mediante el uso de PG se realiza gracias a la propiedad de tipado existente en la PG [Montana, 1995]. Esta propiedad permite

desarrollar estructuras que sigan una gramática determinada. Para el caso particular que ocupa este trabajo, la estructura debe de permitir la construcción de RR.NN.AA. Para ello, cada nodo del árbol de PG debe de tener un tipo, y, además, para aquellos que no sean hojas del árbol (es decir, que tengan hijos), será necesario establecer el tipo que debe de tener cada uno de sus hijos. Por tanto, en primer lugar, y con el objetivo de utilizar PG para generar RR.NN.AA., habrá que definir los tipos que se van a utilizar. De esta forma, se utilizarán los siguientes tipos [Rivero 2006b]:

- TRED. Este tipo identifica la red. Se utilizará sólo en la raíz del árbol.
- TNEURONA. Este tipo identifica a un nodo (o subárbol) como una neurona, bien sea oculta, de salida o de entrada.
- TREAL. Este tipo identifica al nodo (o subárbol) como un valor real. Se utiliza para fijar el valor de los pesos de las conexiones, es decir, un nodo que tenga este tipo será o bien una constante de punto flotante o bien un subárbol aritmético que define un valor real.

Con esos tres tipos ya se pueden construir redes. Se define ahora el conjunto de terminales y funciones. Su descripción es la siguiente:

- RNA. Es el único nodo que será de tipo TRED. Dado que se pide que el árbol generado por la PG sea de tipo TRED, este nodo será la raíz del árbol, no pudiendo aparecer en otra parte del árbol. Tendrá el mismo número de hijos que salidas se quiera que tenga la red, cada uno de estos hijos será de tipo TNEURONA, puesto que será una neurona o subred.
- n-Neurona. Conjunto de nodos que identifican a una neurona de n entradas. Estos nodos serán de tipo TNEURONA, y tendrán 2*n hijos. Los primeros n hijos serán de tipo TNEURONA, y designarán las neuronas o subredes que serán entradas a esta neurona. Los segundos n hijos serán de tipo TREAL, y contendrán el valor de los correspondientes pesos de las conexiones de las neuronas de entrada (los primeros n hijos) a esta neurona.
- Neurona_Entrada_n. Conjunto de nodos que definen una neurona de entrada que recibe su valor de activación de la variable de estímulo número n. Estos nodos serán de tipo TNEURONA, y no tendrán hijos.

- Por último, y para generar los valores de los pesos de las conexiones (que serán subárboles de los nodos n-Neurona), se necesita el conjunto de operadores aritméticos {+,-,*,%}, donde % designa la operación de división protegida (devuelve un resultado de 1 si el divisor es 0). Estos nodos realizarán operaciones entre constantes para dar lugar a nuevos valores. Por lo tanto, también se necesita añadir valores reales para que se puedan realizar estas operaciones. Estos valores reales se introducen mediante la adición de constantes aleatorias en el rango [-4, 4], como es recomendado en [Fahlman 1988].

Una descripción más formal de estos operadores puede verse en la tabla 5.1.

	Nombre	Tipo	Número de hijos	Tipos de los hijos
Conjunto de funciones	RNA	TRED	n	TNEURONA, …, TNEURONA
	n-Neurona	TNEURONA	2*n	TNEURONA, …, TNEURONA, TREAL, …, TREAL
	+, -, *, %	TREAL	2	TREAL, TREAL
Conjunto de terminales	Neurona_Entrada_n	TNEURONA	-	-
	[-4, 4]	TREAL	-	-

Tabla 5.1. *Conjunto de terminales y funciones del sistema de PG*

En la figura 5.1 puede verse una red sencilla que se puede construir con este conjunto de terminales y funciones. En ella, los nodos denominados "NE_x" corresponden a "Neurona_Entrada_x". La red acepta 4 entradas distintas y tiene dos salidas diferentes.

Este sistema de tipado permite la construcción de redes sencillas, pero tiene un gran inconveniente, que no permite la reutilización de parte de la red. Con el conjunto de operadores especificado, no es posible hacer que la salida de una neurona sea entrada a más de una neurona distinta (exceptuando las neuronas de entrada de la red), es decir, que una misma neurona no puede ser referenciada múltiples veces desde distintas partes de la RNA. Esto es un gran inconveniente, dado que se elimina una de las grandes ventajas de las RR.NN.AA., que es la reutilización de parte de su estructura. Las RR.NN.AA., dada la alta conectividad que suelen presentar, reutilizan de forma masiva resultados computados anteriormente, convirtiendo muchas partes de la red en bloques funcionales.

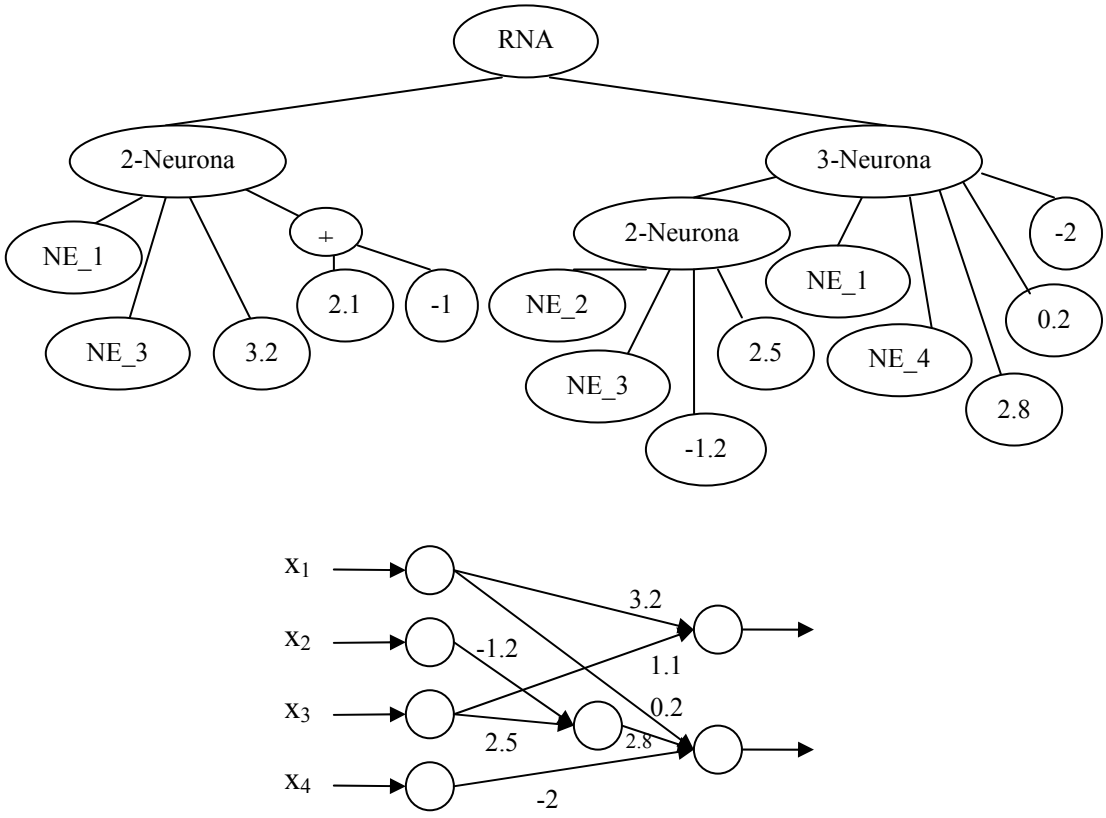


Figura 5.1. *Árbol de PG y su correspondiente RNA*

Para poder realizar esto, es decir, para poder referenciar una neurona como entrada desde más de un elemento de procesado, se ha ampliado el conjunto de terminales y funciones y se ha complicado el sistema para incluir una lista que permita referenciar nodos utilizados anteriormente. A medida que se evalúa el árbol y se va construyendo la red, se almacenan en esta lista las neuronas que se van añadiendo a la red y, por medio de operadores especiales, se extraen de ella para poder reutilizarlas. Para extraer neuronas de la lista con el objetivo de reutilizarlas, se contará además con un índice que apuntará a un elemento de la misma (es decir, una neurona), que será la siguiente en ser extraída tras la ejecución del operador correspondiente. De esta forma, se necesitan dos nuevos operadores, para extraer neuronas de la lista y para modificar la posición del índice. La adición de neuronas a la lista se realiza de forma automática, a medida que se va evaluando el árbol. La descripción de los nuevos operadores es la siguiente:

- "Pop". De tipo TNEURONA. Este nodo extrae la neurona de la lista que esté en la posición indicada por el índice. Este nodo sustituye a la evaluación de una neurona, puesto que devuelve una ya existente y, por lo tanto, no tiene hijos.

- "Forward". De tipo TNEURONA. Este nodo avanza una unidad el índice de la lista. Tiene un hijo, de tipo TNEURONA. Al contrario que el operador anterior, su evaluación no sustituye a la evaluación de una neurona, puesto que su hijo será una neurona. La evaluación de este nodo tendrá dos efectos: avanzar en una unidad el índice de la lista y devolver la neurona resultado de evaluar a su hijo.

Por lo tanto, en un árbol se puede encontrar, en lugar de un nodo "n-Neurona", un nodo "Pop", el cual, en lugar de crear una neurona, referenciará una creada anteriormente. También podrá encontrarse un nodo "Forward", que incrementará el índice de la lista, pero forzosamente su hijo será una neurona, por lo que devolverá una neurona, bien sea una nueva que acabe de crear, o una ya existente.

En la evaluación del operador "n-Neurona", por lo tanto, añade esta nueva neurona que se está creando a la lista. La evaluación de este operador implica la posible (y probable) creación de otras neuronas. El orden de adición de esta neurona a la lista con respecto a la evaluación de los hijos de este operador y, por tanto, creación de nuevas neuronas es crucial, y pueden darse dos casos:

- La neurona es creada y añadida a la lista antes de que sus hijos hayan sido evaluados y las respectivas neuronas creadas. En este caso, en el momento de evaluar un nodo de tipo TNEURONA presente en un subárbol de un hijo de esta neurona, podría aparecer, en lugar de una creación de neurona, un nodo "Pop", con lo que se referenciará una neurona de la lista. Dado que la neurona antecesora está presente en la lista, es posible que se referencie a esta neurona antecesora, con lo que se está creando un enlace recurrente. Por lo tanto, este orden de evaluación permite la creación de redes recurrentes.

- La neurona es creada, sus hijos evaluados y posteriormente añadida a la lista. En este caso, en la creación de los enlaces de las neuronas antecesoras, ésta no estará presente en la lista, con lo que no se podrán realizar conexiones recurrentes a neuronas antecesoras. Este es el caso de estudio del presente trabajo, en el que no se están desarrollando redes recurrentes.

Para una mejor comprensión de la evolución del conjunto de terminales y funciones, se incluye aquí un pseudocódigo en el que se puede ver cómo se evalúan cada uno de los nodos existentes en la red.

```
evaluar(nodo)
begin

  case nodo of:

  RNA:
    begin
      crear red vacía, con neuronas de entrada
      crear Lista vacía
      Indice = 1
      for i=1 to (numero de salidas de la red),
        neurona = evaluar(hijo i)
        establecer neurona como salida de la red
      endfor
      devolver red
    end

  n-Neurona:
    begin
      crear neurona
      for i=1 to n,
        neurona_entrada = evaluar(hijo i)
        peso_entrada = evaluar(hijo i+n)
        if (neurona_entrada ya es entrada de neurona) then
          actualizar peso de neurona_entrada con peso_entrada
        else

          establecer neurona_entrada como entrada a neurona con peso_entrada
        endif
      endfor
      añadir neurona a la lista
      devolver neurona
    end

  Neurona_Entrada_n:
    devolver neurona de entrada n

  operador aritmetico:
    begin
      f1 = evaluar(hijo 1)
      f2 = evaluar(hijo 2)
      devolver resultado de combinar f1 y f2
    end

  float f:
    devolver f

  Pop:
    devolver Lista[Indice]

  Forward:
    begin
      Indice = Indice + 1
      neurona = evaluar(hijo)
      devolver neurona
    end

  endcase
end
```

Un ejemplo de red que incluya estos operadores puede verse en la Fig. 5.2. Para simplificar, en esta figura se han suprimido los subárboles de operaciones aritméticas que fijan el valor de los pesos de las conexiones. Como en la figura anterior, los nodos denominados “NE_n” se refieren a “Neurona_Entrada_n”. En esta figura se puede ver la

red que va generando en sucesivos momentos de evaluación del árbol. La explicación de estos momentos, de forma correspondiente a cómo ha sido nombrada en la figura, es la siguiente:

a) Red generada antes de la evaluación del primer nodo "Forward". Se puede ver que se han creado dos neuronas, etiquetadas en este caso con los números 1 y 2, que aún no han sido conectadas mutuamente, pero que sí han sido añadidas a la lista.

b) Red generada después de evaluar ese nodo Forward. En este caso la neurona de entrada número 2 aún no ha sido conectada a la neurona etiquetada como "2" porque esto se realizará al final de la evaluación de esta neurona.

c) Red generada en la evaluación del nodo etiquetado como "5", antes de la evaluación de sus hijos. En este punto ya ha sido creado ese nodo, pero aun no ha sido introducido en la lista puesto que esto se realiza al final de la evaluación de este nodo. En este punto, además, ya se ha evaluado todo el subárbol izquierdo con lo que se ha generado una parte importante de la red.

d) Aquí se muestra la red durante la evaluación del nodo "5", después de evaluar el primer hijo. En este punto, como primer hijo se ha encontrado un nodo "Pop", con lo que no se ha creado ninguna neurona sino que, en su lugar, se ha referenciado una existente, la que estaba apuntada por el índice en la lista, en este caso la número 2.

e) Red generada tras la ejecución del segundo nodo "Forward". En este punto sólo se ha avanzado el índice de la lista, apuntando ahora al nodo "3", que será el que se devuelva en el siguiente nodo "Pop".

f) Red generada finalmente, tras la evaluación del último nodo "Pop" y la finalización de la evaluación de los nodos 5 y RNA.

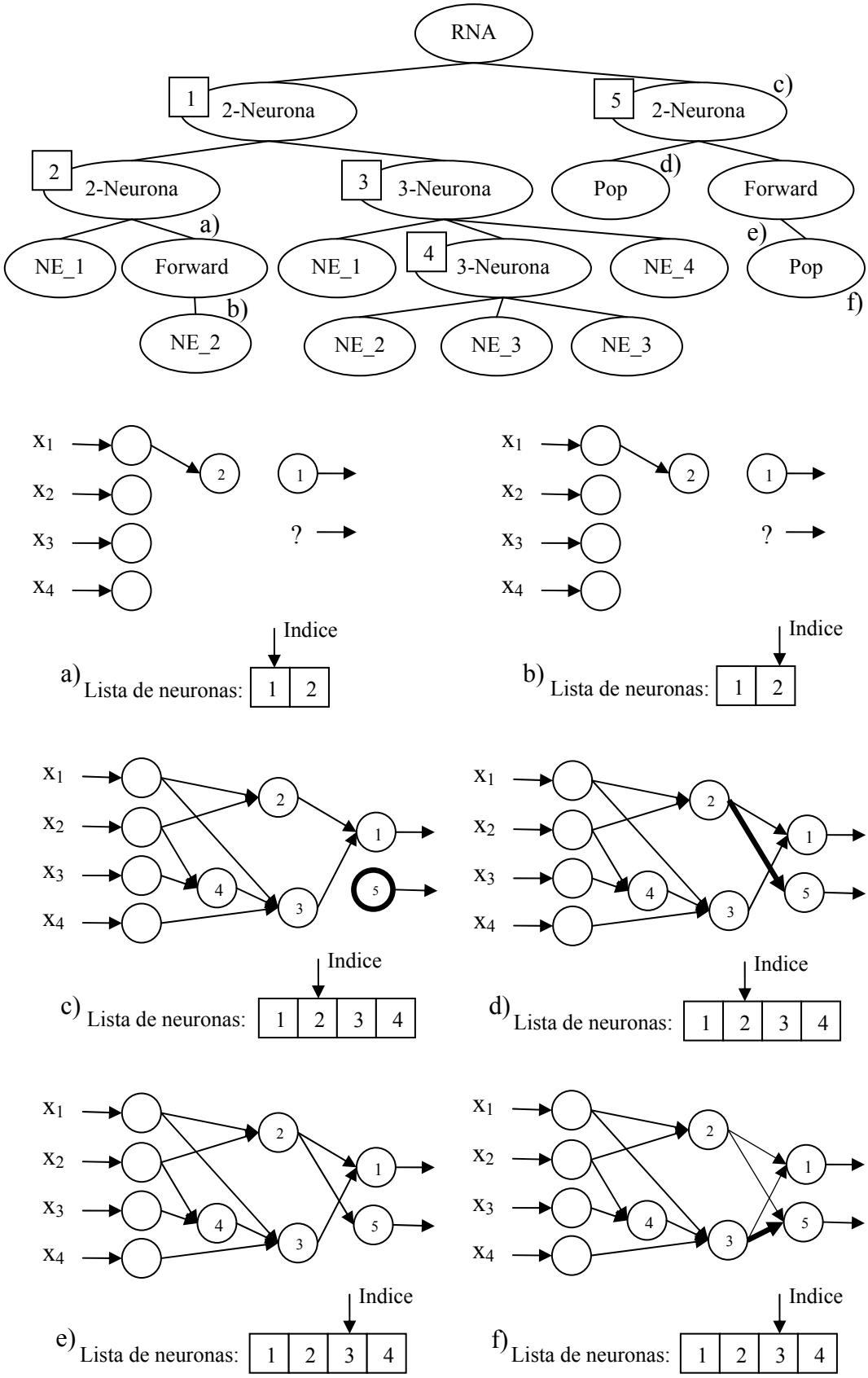


Figura 5.2. *Árbol con nodos "Forward" y "Pop", y las RR.NN.AA.*

Es importante tener en cuenta que, durante la creación de una neurona, en el proceso de creación o referenciación de neuronas hijas, es posible que aparezca como antecesora varias veces repetida alguna neurona, bien sea de entrada o alguna ya referenciada. En este caso, lógicamente, no se establece una nueva conexión de entrada por parte de ese elemento de procesado, puesto que ya existía uno, sino que se modifica el peso de esa conexión existente añadiéndole el valor del peso de la nueva conexión. Por lo tanto, una situación muy común es que un operador "n-Neurona" no esté referenciando a n neuronas distintas, sino que es posible que haya alguna repetida, especialmente si n es un valor alto. Es necesario limitar el valor de n, para saber cuál es el número máximo de antecesores que puede tener una neurona. Un valor muy alto seguramente provocará el efecto descrito, es decir, que no se hará un uso efectivo de todas esas entradas, sino que alguna se repetirá. Sin embargo, también es necesario tomar un valor alto para asegurar que se pueda tener un alto número de antecedentes.

5.1.2 Uso de grafos

Las representaciones de programas mediante grafos tienen la ventaja de que pueden ser más compactas, en términos del número de nodos, y más eficientes que las representaciones en forma de árbol. Un ejemplo de cómo el uso de grafos puede resultar en una representación más compacta se ilustra en la figura 5.3. Sin embargo, el manejo directo de grafos con PG presenta algunos problemas [Poli 96].

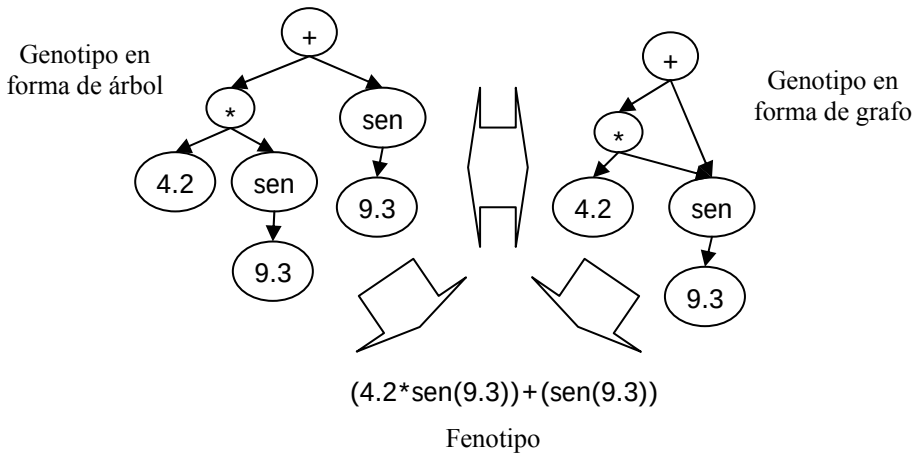


Figura 5.3. *Ejemplo de árbol y grafo como formas de codificación*

Existen diversas representaciones directas para los grafos en la teoría de grafos. Para cada una de ellas pueden imaginarse operadores que seleccionan un subgrafo en un padre o un subgrafo aleatorio. Sin embargo, no es fácil producir buenos operadores genéticos para las representaciones directas de grafos. En particular es difícil producir un operador que cumpla las siguientes propiedades:

1. Cuando los padres comparten una característica sus descendientes la hereden.
2. Cuando los padres tienen diferentes características sus descendientes puedan heredar dichas características.
3. Todo descendiente sea una solución válida.
4. El operador sea eficiente.

Las representaciones indirectas de grafos, como la codificación celular o la codificación de aristas, no presentan este problema ya que los operadores de PG estándar se pueden utilizar con dichas codificaciones. Sin embargo, estas representaciones requieren un paso adicional de decodificación genotipo a fenotipo antes de que se puedan interpretar los grafos, ya que la búsqueda no se realiza en el espacio de los posibles grafos sino en el espacio de los programas secuenciales que producen grafos [Galván 2004]. Cuando la evaluación de la función de ajuste implica cálculos complejos el paso de decodificación puede tener un efecto relativo limitado en términos de coste computacional.

En este trabajo en lugar de utilizar árboles para codificar los individuos en la PG se utilizan grafos acíclicos. Dos conceptos muy utilizados en la PG son el de raíz del árbol y el de altura. Estos conceptos no existen para grafos que no sean árboles. Sin embargo, para mantener una mayor similitud con la bibliografía tradicional, en este trabajo se utilizan las definiciones de raíz del grafo y altura del grafo. Se supone que se tiene identificado el orden en el que son generados los nodos del grafo. A partir de esto, se define:

- Raíz del grafo: un nodo es la raíz del grafo si es el primer nodo que ha sido generado en ese grafo. De esta forma, la raíz será el único nodo que no tiene predecesores.

- Altura del grafo: la altura de un grafo será la longitud del mayor camino que exista en el grafo y que cumpla la condición de que cada nodo perteneciente al camino haya sido generado después de todos sus predecesores.
- Subgrafo: dado un nodo, se dirá que el subárbol que tiene a dicho nodo como raíz es el conjunto de sucesores de dicho nodo generados con posterioridad a ese nodo.

5.1.2.1 Modificación de la PG

Para poder utilizar grafos como forma de codificación dentro de la PG, es necesario modificar los operadores genéticos que trabajan con los individuos, en este caso codificados en forma de grafos. En concreto, será necesario modificar los algoritmos de creación, cruce y mutación de árboles para adaptarlos al uso de grafos.

5.1.2.1.1 Creación

Para la generación de grafos con PG se deben cumplir las mismas restricciones comentadas anteriormente para la generación de árboles y, además, deben tenerse en cuenta las siguientes consideraciones:

- Debe permitirse la generación de grafos. Por lo tanto el método de generación de individuos debe ser menos restrictivo que en el caso de la generación de árboles, permitiendo que un nodo sea compartido por varios padres.
- En los grafos generados no deben existir circuitos.

En el método propuesto la generación de grafos se realiza, siguiendo una exploración en profundidad, de la siguiente forma:

1. Se comienza generando el nodo raíz, seleccionando aleatoriamente un nodo no terminal.
2. A continuación, realizando una exploración en profundidad del árbol que se está generando se selecciona, cada vez que es necesario generar un nodo, aleatoriamente un elemento del conjunto S. S es el conjunto formado por la unión de los siguientes conjuntos:
    - El conjunto de nodos terminales.
    - El conjunto de nodos no terminales (funciones).

- El conjunto de nodos generados y que no sean predecesores del nodo que se está generando.

Sin embargo ha de tenerse en cuenta que, al igual que se hace en el caso de utilizar sólo árboles, a la hora de elegir un nodo del conjunto S se puede restringir (según la altura que se quiera alcanzar) que sea un nodo terminal o una función. Si el nodo elegido no es una hoja ni un nodo generado previamente el proceso se repetirá recursivamente con cada uno de los hijos de ese nodo. Igualmente que se hacía con árboles, será necesario respetar la restricción de tipado, por lo que, antes de escoger un nodo del conjunto S, se elimina de este aquéllos que no tienen el tipo requerido por el nodo antecesor.

Al incluir en el proceso de selección el conjunto de nodos ya generados se permite que en la construcción aleatoria de individuos se generen grafos, si se selecciona como nuevo nodo uno de los pertenecientes a dicho conjunto, y que la construcción no se limite a la mera generación de árboles. Por otra parte, al impedir que los predecesores sean seleccionados y debido a que la generación se realiza en profundidad, se impide que existan circuitos en los grafos generados. Por lo tanto, el método propuesto cumple con todas las restricciones necesarias para la generación aleatoria de grafos y permite la generación de grafos que no sean árboles y que no tengan circuitos.

5.1.2.1.2 Mutación

Por su parte, al realizar la mutación de un individuo se sustituye uno de sus subgrafos por otro generado aleatoriamente. Dicho subgrafo es generado teniendo en cuenta las reglas enunciadas anteriormente. Además, hay que tener en cuenta que alguno de los nodos del subgrafo reemplazado podrían tener como predecesores nodos que permanecerán en el grafo tras la mutación. Un ejemplo se muestra en la figura 5.4. Si se selecciona para la mutación el nodo que contiene el operador de multiplicación y todo el subgrafo que representa, se entraría en el caso de tomar un subgrafo en el que algún nodo tiene un predecesor fuera del mismo (además, por supuesto del nodo raíz del subgrafo seleccionado). En este caso es el nodo “y” el que tiene un predecesor fuera, el nodo de suma. La dificultad que surge en casos de este estilo es qué hacer con ambos padres. Al menos uno de ellos debe mantenerse para asegurar que el grafo no queda disconexo. La solución utilizada en este trabajo se resume en dos reglas:

- ✓ Los nodos que sean predecesores del nodo 'raíz' del subgrafo que se va a mutar se mantienen como predecesores del nodo raíz del nuevo subgrafo generado.
- ✓ A aquellos nodos no pertenecientes al subgrafo mutado y cuyos sucesores sí pertenezcan a dicho subgrafo (y no sean el 'raíz' del subgrafo mutado), se les asignará aleatoriamente como dichos sucesores un nodo del nuevo subgrafo generado.

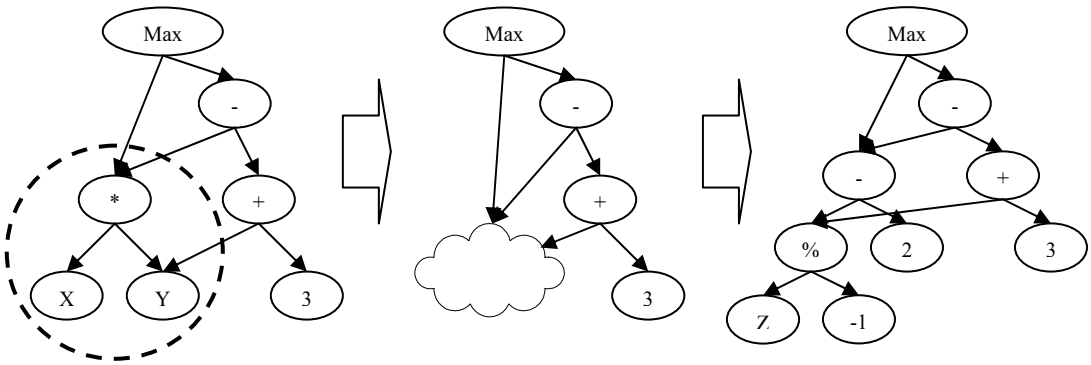


Figura 5.4. *Ejemplo de mutación*

El nuevo subgrafo se crea de una forma similar a la explicada en el algoritmo de creación de grafos. Un ejemplo de este proceso viene ilustrado en la figura 5.4.

5.1.2.1.3 Cruce

Finalmente, el algoritmo de cruce es el más complicado de modificar para su adaptación. Al igual que en los anteriores, deben de tenerse en cuenta y respetarse las restricciones mencionadas anteriormente. Sin embargo, al tratar con grafos surge una nueva dificultad. Al intercambiar dos subgrafos, puede ocurrir que alguno de los ancestros de los nodos que se intercambian permanezca en el individuo de origen. Un ejemplo de este tipo puede verse en la figura 5.5. En este ejemplo, tras el intercambio de los subgrafos, existen referencias desde un individuo a partes que están en otro distinto. Por lo tanto, si el intercambio se realizase como en el caso en el que se trabaja sólo con árboles, sin tener en cuenta ninguna otra consideración, se podría obtener un grafo codificado a través de varios individuos. De esta forma, la codificación de un individuo estaría entremezclada con la codificación de otros individuos, algo totalmente indeseable. Para evitar ese tipo de inconsistencias se ha modificado el método de cruce tradicional para adaptarlo al uso de grafos. La solución al problema de los predecesores

es similar a la empleada en el caso de la mutación. El método utilizado en este trabajo añade al método tradicional dos reglas más:

- ✓ Los nodos que sean predecesores del nodo 'raíz' de uno de los subgrafos que se van a intercambiar se ponen como predecesores del nodo raíz del otro subgrafo.
- ✓ A aquellos nodos no pertenecientes al subgrafo intercambiado y cuyos sucesores sí pertenezcan a dicho subgrafo (y no sean el 'raíz' de un subgrafo a intercambiar), se les asignará aleatoriamente un nodo del otro subgrafo seleccionado.

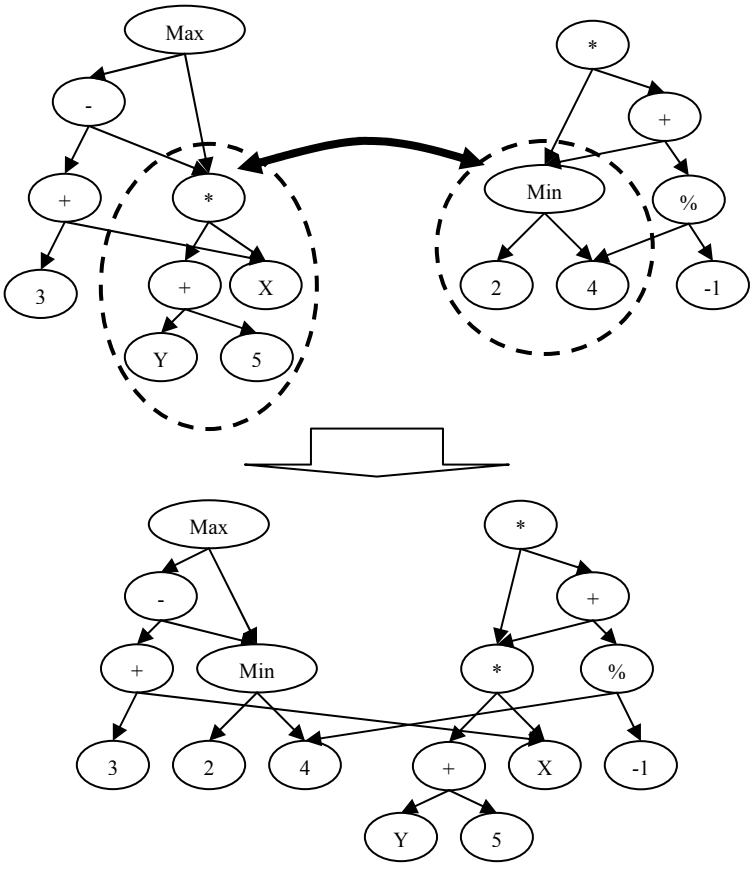


Figura 5.5. *Ejemplo de cruce sin tener en cuenta las restricciones*

Un posible resultado del cruce propuesto en la figura 5.5 puede verse en la figura 5.6. Debe tenerse en cuenta que, en el caso de usar árboles, una vez seleccionados los nodos raíces de los subárboles a intercambiar, el resto del proceso de cruce era determinista. En el caso en el que se utilicen grafos existe una diferencia fundamental y es que, aunque se hayan elegido los mismos nodos 'raíces' a intercambiar, el cruce no está completamente determinado. Esto se debe a que la presencia de predecesores de nodos no 'raíz' produce una asignación aleatoria de sucesores. Por supuesto, a la hora

de seleccionar los nodos que representarán los subgrafos a cruzar, debe de seguir respetándose las restricciones de tipado y altura en los grafos resultantes. Igualmente, estas restricciones deben de respetarse a la hora de designar nuevos sucesores a aquellos nodos que tengan hijos dentro de los subgrafos cruzados.

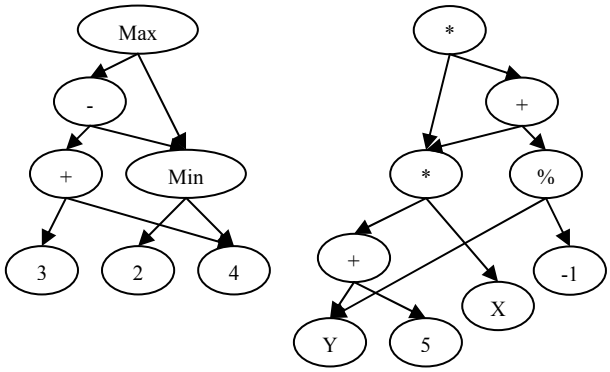


Figura 5.6. *Ejemplo de resultado del cruce*

5.1.2.2 Modelo

El proceso de codificación de RR.NN.AA. en grafos es mucho más sencillo que cuando se usaban árboles, puesto que no se necesita la estructura externa (lista de neuronas) ni los operadores especiales que operan con ella. Estos elementos eran necesarios cuando se usaban árboles para emular una estructura de grafos. Sin embargo, ya no es necesario emularla, puesto que ahora la codificación en forma de grafos permite la representación de las RR.NN.AA. de forma directa dentro de los individuos de PG.

El conjunto de operadores propuesto es, por lo tanto, más sencillo que el utilizado para árboles. Dado el sistema de tipado explicado en la sección 5.1.1, los conjuntos de terminales y funciones pueden verse resumidos en la tabla 5.1, y su descripción es la misma que para el caso de usar árboles. Es decir, los conjuntos de terminales y funciones son los mismos que para árboles, pero sin la adición de los operadores especiales de “Pop” y “Forward” [Rivero 2006a].

Con estos conjuntos, la representación de redes es mucho más directa. En la figura 5.7 puede verse un ejemplo en el cual se codifica la misma red que la representada en la figura 5.2 con árboles. Puede verse que en este caso el individuo es más sencillo, y no es necesario utilizar operadores especiales para poder representar las redes. Al igual que sucedió antes, y con el objetivo de simplificar la representación, los nodos etiquetados

como "NE_x" se corresponden con "Neurona_Entrada_x", y no se han mostrado los pesos de las conexiones de la red.

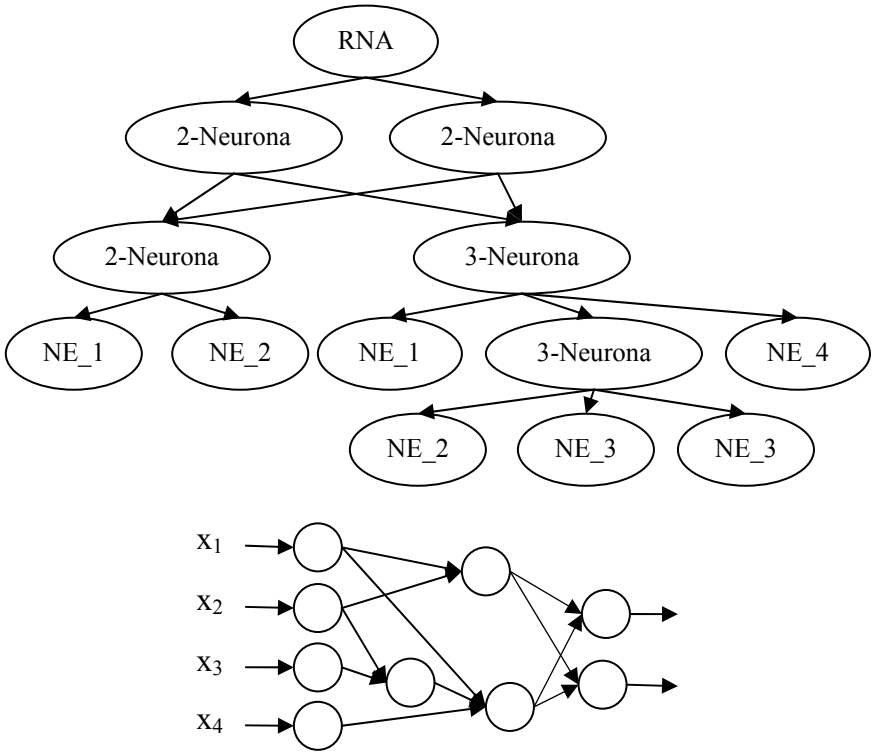


Figura 5.7. *Ejemplo de grafo que representa una RNA*

5.1.3 Optimización de los pesos mediante el uso de AA.GG.

Una vez el sistema ha sido configurado, bien sea mediante el uso de árboles o grafos como forma de codificación, es posible ampliarlo mediante un proceso de optimización de los pesos de las conexiones utilizando para ello un AG.

Este proceso es genérico y puede aplicarse para cualquier tipo de optimización de constantes en cualquier tipo de expresiones de PG [Cagnoni 2005]. En este caso, se utilizará en los valores de los pesos de las conexiones de las redes representadas por cada individuo de PG.

Para utilizar este proceso, se elimina del conjunto de funciones los operadores aritméticos. De esta forma, el proceso evolutivo se inicia de la misma forma; es decir, se crean las constantes iniciales de forma aleatoria en el rango seleccionado. Sin embargo, una vez evaluada esta población inicial, esta sufre un proceso de transformación: se toma un porcentaje F_{sel} de individuos de la población (los mejores) y, a cada uno de ellos, se le aplica un proceso de optimización de constantes. Este proceso de

optimización se aplica, como se ha dicho, tras la creación de la primera población, pero también tras la ejecución de N_{int} generaciones tras la última aplicación.

El proceso de optimización que se aplica se basa en tomar los pesos de las conexiones del individuo (que representa una RNA) y optimizar estos valores. Esto se realiza mediante la creación de un AG con codificación real, en el que cada gen representa el valor de un peso. Este AG se ejecuta y, como consecuencia de ello, se genera un proceso evolutivo de refinamiento de los pesos de la red. Tras la ejecución del mismo, el individuo de PG, es decir, la red, tomará unos valores de pesos distintos, optimizados, con un valor de ajuste mejor.

La evaluación de cada individuo del AG es similar a la evaluación de cada individuo de PG. En el caso del AG se toma el cromosoma que representa los pesos de las conexiones, se establecen los valores de estos pesos en la red que está siendo optimizada, y se evalúa la misma con el conjunto de entrenamiento. Por lo tanto, el valor del ajuste devuelto es el mismo que se calcula para los individuos de PG.

De una forma más esquemática, el proceso de optimización de constantes aplicado es el siguiente:

1. Se crea un AG cuya longitud del cromosoma será la del número de pesos que tiene la red representada por el individuo seleccionado. Cada uno de los genes representa un peso de la red.
2. Se inicializa la población del AG, de tamaño N_{ga} individuos. Esto se realiza de la siguiente forma:
    - ✓ Se introduce dentro de la población vacía el individuo representado por los pesos ya conocidos del individuo de PG que se está optimizando. Igualmente, también se conoce su ajuste, con lo que también se establece, por lo que no hace falta evaluarlo. Es decir, se inserta como individuo inicial del AG el conjunto de pesos ya existente.
    - ✓ Se generan de forma aleatoria los N_{ga}-1 individuos restantes de la población y se evalúan. La función de evaluación será la explicada anteriormente, es decir, establecer como pesos de las conexiones los valores de los genes del individuo genético a evaluar y devolver

como ajuste el resultado de evaluar la red en el conjunto de entrenamiento.

3. Se ejecuta el algoritmo genético de forma tradicional. El algoritmo se ejecuta un total de N_{gagen} generaciones o hasta que no hay cambio en el mejor ajuste durante 5 generaciones.

4. Una vez que finaliza la ejecución del algoritmo genético se toma, del mejor individuo encontrado durante este proceso, los valores de los genes y se establecen en las correspondientes constantes del individuo de PG (es decir, en el árbol o grafo) que representan los pesos de la red. Es decir, las mejoras halladas por el AG se guardan dentro del individuo de PG. Por lo tanto, este proceso de optimización puede verse como una estrategia "lamarckiana" [Ku 1997] [Lamarck 1809], en el que los cambios realizados persisten. También se cambia el valor del ajuste de ese individuo de PG al ajuste del individuo del AG.

Este proceso de optimización se realiza a un porcentaje pequeño (F_{sel}) de individuos de PG. Estos individuos tomados serán los mejores de la población, teniendo cuidado de no tomar ninguno que esté repetido. Una vez que ha finalizado la optimización, prosigue la ejecución del algoritmo de PG. Tras la ejecución de N_{int} generaciones de PG, este proceso se repite nuevamente con la población correspondiente de PG.

Por lo tanto, existen cuatro parámetros fundamentales que guían este proceso:

- N_{int}. Número de generaciones de PG que deben de pasar entre dos ejecuciones del proceso de optimización.
- F_{sel}. Porcentaje de individuos de PG al cual se aplica la optimización.
- N_{ga}. Tamaño de la población del AG.
- N_{gagen}. Número máximo de generaciones que se ejecuta el AG. El algoritmo se para tras la ejecución de N_{gagen} generaciones, o tras 5 generaciones sin mejora en el mejor individuo. Experimentalmente se ha observado que este parámetro no ejerce ninguna influencia sobre el proceso, puesto que la mayoría de las veces no se alcanzan las N_{gagen} generaciones para parar el proceso porque siempre ocurren 5 generaciones sin mejora en el ajuste antes de llegar a ese punto, incluso para valores muy bajos de este parámetro (menores incluso que 15 ó 10). Si se toman

valores menores, lo que ocurre es que se paraliza la búsqueda de constantes antes de que esta finalice, con lo cual el proceso de optimización resulta incompleto.

5.2 Función de ajuste

Una vez se ha evaluado un árbol, este genotipo se ha transformado en fenotipo, es decir, en una red con los valores de los pesos fijados y que ya puede ser evaluada, no necesita ser entrenada. El proceso evolutivo exige la asignación de un valor de ajuste a cada genotipo. Este valor de ajuste será el resultado de la evaluación de la red con el conjunto de patrones que representan el problema. En este caso, el resultado de la evaluación será el error cuadrático medio (ECM) de la diferencia entre las salidas de la red y las deseadas.

Sin embargo, este valor de error que se toma como valor de ajuste ha sido modificado para provocar que el sistema intente generar redes sencillas. Para ello, ha sido penalizado con un valor que se suma a este error. Este valor que se suma no es más que un valor de penalización multiplicado por el número de neuronas que tiene la red. De esta forma, y dado que el sistema evolutivo se ha diseñado para minimizar un valor de error, si se suma un valor al valor de ajuste se provocará que una red más grande obtenga un valor de ajuste peor, con lo que se favorece la aparición de redes sencillas, puesto que este valor de penalización que se suma es proporcional al número de neuronas de la RNA. El cálculo del ajuste final que se utiliza es el siguiente:

$$ajuste = ECM + N * P$$

donde ECM es el error cuadrático medio de la red en el conjunto de patrones de entrenamiento, N es el número de neuronas de la red y P es el valor de penalización al número de neuronas. La constante P tendrá un valor bajo, mucho menor que la unidad, y, como se mostrará en los experimentos, será crucial en la evolución del sistema. De esta forma, al hacer que una red con más neuronas pero idéntico funcionamiento tenga un valor peor de ajuste, se intenta que el sistema busque sistemas sencillos, con pocas neuronas.

Un problema muy común a la hora de generar RR.NN.AA. y entrenarlas consiste en el sobreentrenamiento de las mismas [Bishop, 1995] [Ripley, 1996], como ya ha sido explicado en la sección 2.1.4.2. Esto se produce por varias razones, principalmente por

entrenar una red con un número demasiado alto de conexiones para el problema que se quiere solucionar, o por realizar el entrenamiento durante un período de tiempo demasiado alto (por ejemplo, un número muy alto de ciclos en el algoritmo BP o de generaciones con un algoritmo evolutivo). Esto se puede notar cuando durante el proceso los resultados de entrenamiento siguen mejorando, mientras que los de test empiezan a empeorar. En ese caso, la red está perdiendo la capacidad de generalización. En este trabajo, el problema se intenta solucionar, ya que se desea que este efecto no se produzca.

Para evitar este problema, además del conjunto de entrenamiento, se utiliza otro conjunto de patrones distinto, con el objetivo de realizar una validación. La finalidad de esta validación es, principalmente, controlar el proceso de aprendizaje para evitar el sobreentrenamiento de las redes. Unidos a estos dos conjuntos, se necesita un tercero, de test, cuyo objetivo es evaluar la red resultante de todo el proceso de aprendizaje. En total, por lo tanto, se tienen 3 conjuntos distintos y disjuntos: entrenamiento, validación y test. Los conjuntos de entrenamiento y validación son los que dictan el entrenamiento. El conjunto de test se aplicará al final, a la red resultante de todo el proceso evolutivo, para evaluar el nivel de aprendizaje de la misma.

El proceso de entrenamiento se realiza hasta que el usuario decida parar, o bien hasta que se haya alcanzado un número máximo de generaciones fijado de antemano, o de ejecuciones de la función de ajuste, que es lo que se utiliza en los experimentos llevados a cabo en este trabajo.

El conjunto de validación se utiliza para calcular un nuevo valor de error. Este cálculo se realiza de forma conjunta al error de entrenamiento para cada nuevo individuo; es decir, para cada nueva red generada, o para una ya existente con un conjunto de pesos nuevos. Así, tras cada cómputo de entrenamiento seguirá uno de validación y cada red tendrá asociados esos dos valores de error: entrenamiento y validación. Sin embargo, el valor de ajuste no se modifica, sino que sigue dependiendo sólo del valor de entrenamiento, no del de validación. Este valor puede verse como una estimación del valor de test que ofrecerá la red, puesto que el conjunto de validación contiene patrones que no están presentes en el conjunto de entrenamiento.

El proceso de entrenamiento almacena un individuo, que será el que devuelve el sistema, y el cual se actualizará cuando se encuentre otro individuo con un mejor error

de validación. Es necesario almacenar este individuo porque no es el mejor individuo de la población de PG (que sería el de mejor ajuste), sino el de mejor error de validación. Y puede haber sido generado por el proceso evolutivo en cualquier generación. En resumen, el entrenamiento discurre de la siguiente manera:

1. Se inicializa y ejecuta el algoritmo de PG.
2. En la evaluación de cada individuo se calculan dos valores:
   - ✓ Error de entrenamiento. Se utiliza para calcular el valor del ajuste, tras aplicarle la penalización por el número de neuronas.
   - ✓ Error de validación. Se utiliza para controlar el sobreentrenamiento del proceso.
3. Tras la evaluación de cada individuo, si este tiene un menor error de validación que el individuo almacenado, se sustituye este por el que acaba de ser evaluado.
4. En todo momento, la salida del sistema será este individuo almacenado; es decir, el que menor error de validación ha alcanzado en todo el proceso.
5. Se ejecuta este proceso hasta que se haya alcanzado un número prefijado de ejecuciones de la función de ajuste o generaciones de PG, o bien el usuario decida terminar el proceso.
6. Una vez finalizado el proceso de entrenamiento, el individuo devuelto podrá ser sometido a la fase de test, en la que se evaluará con patrones que no ha visto durante el entrenamiento ni la validación.

Este proceso puede verse gráficamente en la figura 5.8.

El entrenamiento se realiza de una forma similar a la realizada por la técnica de parada temprana [Prechelt 1996] [Prechelt 1998], puesto que una red con mejores valores de entrenamiento y peor validación indica que está siendo sobreentrenada.

Es importante tener en cuenta que, a pesar de que este valor de validación no influye en el cálculo del ajuste y los patrones de validación no están presentes en el conjunto de entrenamiento, puede pensarse que, en la práctica, es como realizar un test a las redes. Sin embargo, esto no es correcto, puesto que al devolver el sistema la red que mejor resultado de validación haya obtenido, de alguna forma se hace intervenir el

conjunto de validación en el proceso global de generación de RR.NN.AA. y, por tanto, también se hace tender de alguna forma las redes resultantes a este conjunto, lo que no se debe hacer con el conjunto de test. Éste sólo debe de ser aplicado para realizar el cómputo del test a la red final resultante de todo del proceso global.

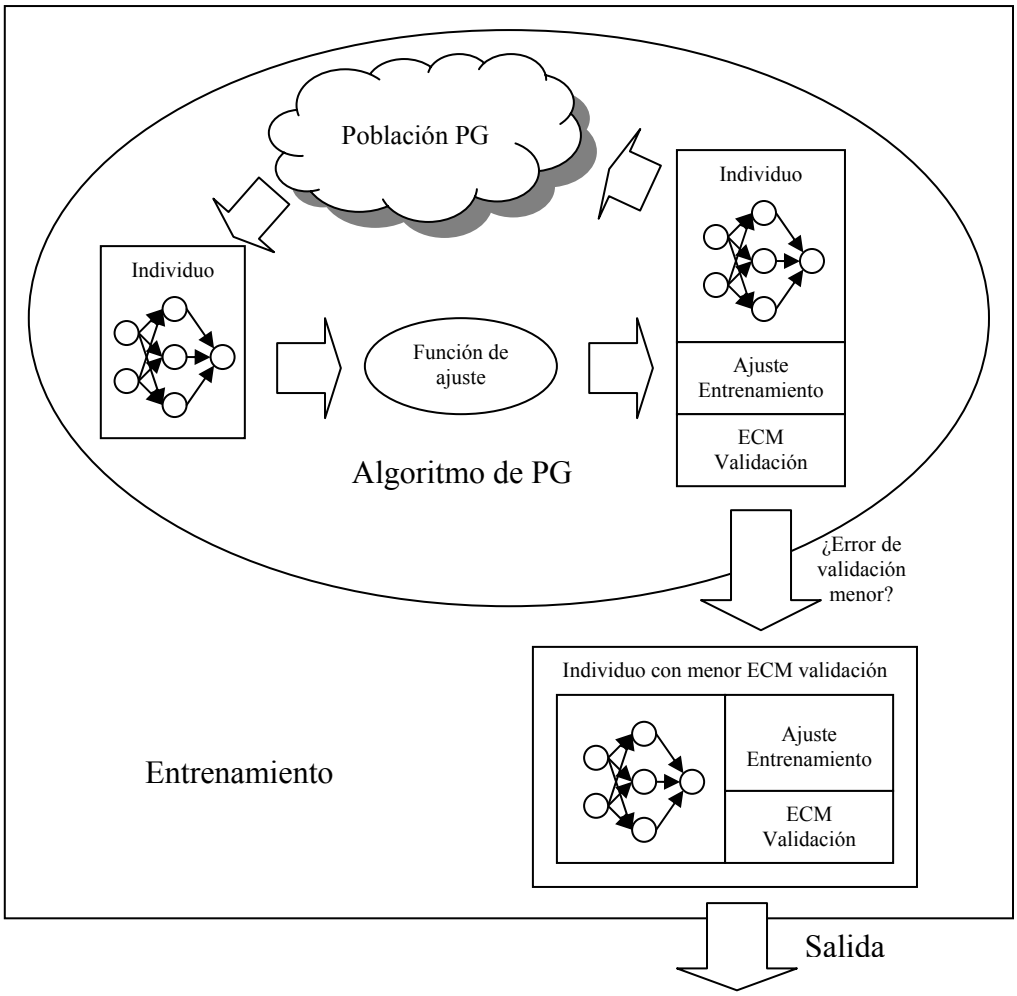


Figura 5.8. *Proceso de entrenamiento seguido*

5.3 Parámetros

El sistema aquí descrito acepta una serie de parámetros. Estos se pueden dividir en tres grupos distintos:

- Parámetros propios de la PG. Este grupo abarca los parámetros típicos existentes en un proceso evolutivo guiado por el algoritmo de PG: algoritmos de creación, cruce y mutación, tamaño de la población, tasa de cruces, probabilidad de mutación, etc.

- Parámetros que limitan el tamaño de las redes que se generan. En este grupo entran fundamentalmente 3 parámetros, ya explicados:
    - Número máximo de entradas de cada neurona.
    - Altura máxima del árbol o grafo.
    - Penalización al número de neuronas. Si bien este parámetro no impone ningún límite al tamaño máximo de las redes, guía notablemente el proceso evolutivo haciendo que se generen redes sencillas. Si este parámetro toma valores bajos, se generarán redes con un alto número de neuronas y conexiones, dando lugar a problemas de sobreentrenamiento. Al contrario, si toma valores demasiado altos, el sistema creará redes con pocas neuronas, incluso sin neuronas ocultas, lo cual lleva a obtener redes que no resuelven el problema.
- Parámetros que controlan el proceso de optimización de pesos mediante AA.GG. En este grupo, como ya se ha descrito, figuran los parámetros F_{sel}, N_{int} y N_{ga}.

Debido a la existencia de tantos parámetros, y dado que existe una alta dependencia de sus valores con los resultados que se obtienen y la eficiencia del sistema, así como una gran dependencia de los valores de los parámetros entre sí, es posible pensar que fijar un valor para los mismos no es una tarea sencilla y que es necesaria la participación de un experto humano para este fin.

Sin embargo, y para cumplir con uno de los objetivos principales de esta Tesis, que es la independencia con el experto, se han realizado diversos experimentos con problemas de distinta complejidad con el objetivo de probar la eficiencia del sistema y para establecer un conjunto de valores para esos parámetros que ofrezcan buenos resultados en problemas de diversa naturaleza y con una complejidad muy distinta.

CAPÍTULO

6 PRUEBAS REALIZADAS

6.1 Problemas a resolver

Para verificar el funcionamiento de este sistema, se ha probado este en la solución de problemas de diversa naturaleza. Todos estos problemas proceden de base de datos tomadas del UCI [Mertz, 2002]. Esta es una colección de bases de datos disponible en internet, de acceso público, pensadas para ser usadas por la comunidad de investigadores en técnicas de aprendizaje máquina para el análisis empírico de los algoritmos que utilizan.

6.1.1 Apendicitis

El primer problema que se intenta resolver es de diagnóstico de apendicitis. Para ello, se cuenta con una base de datos con 106 casos. Se dispone sólo de 8 atributos para realizar la predicción, y son los siguientes:

- (X1) WBC1
- (X2) MNEP
- (X3) MNEA
- (X4) MBAP
- (X5) MBAA
- (X6) HNEP
- (X7) HNEA
- (X8) CRP

6.1.2 Cáncer de mama

El segundo problema es el de clasificación de cáncer de mama, en dos posibles tipos: benigno y maligno. Para ello se cuenta con una base de datos tomada del UCI con 699 datos, con 458 benignos (65.5%) y 241 malignos (34.5%), cada uno caracterizado por 9 atributos, los cuales, a pesar de que toman valores discretos en el rango 1-10, se consideran continuos. Los 9 atributos son los siguientes:

- (X1) *Clump thickness* 1 – 10
- (X2) *Uniformity of cell size* 1 – 10
- (X3) *Uniformity of cell shape* 1 – 10
- (X4) *Marginal adhesion* 1 – 10
- (X5) *Single epithelial cell size* 1 – 10
- (X6) *Bare nuclei* 1 – 10
- (X7) *Bland chromatin* 1 – 10
- (X8) *Normal nucleoli* 1 – 10
- (X9) *Mitoses* 1 - 10

6.1.3 Flores Iris

El siguiente problema en el que será aplicado este sistema es el de clasificación de flores. Este problema fue originalmente planteado por Fisher en 1936 [Fisher 1936] como una aplicación en el campo de análisis discriminante y clusterización. Para este problema se cuenta con cuatro parámetros de naturaleza continua en los que se han recogido medidas en milímetros de cuatro características de las flores: longitud y anchura de pétalos y sépalos. Las medidas recogidas corresponden a 150 flores de tres especies distintas de iris: setosa, verginica y versicolor (50 de cada tipo). Por lo tanto, se tienen 150 datos con 4 características cada uno, para hacer una clasificación en tres posibles tipos, con lo que la red a generar tendrá tres salidas, cada una correspondiente a una clase distinta, y cada salida tomará un valor deseado de 1 si pertenece a esa clase, y de 0 en caso contrario.

6.1.4 Setas venenosas

Otro problema a resolver es el de clasificación de setas venenosas. En este problema el objetivo es determinar, a partir de un conjunto de 22 características de naturaleza simbólica, si una seta es venenosa o no. Para hacer eso, se trabaja con otra base de datos tomadas del UCI con 5644 datos. La descripción de las características es la siguiente:

- (X1) *Cap shape*: *bell, conical, convex, flat, knobbed, sunken*
- (X2) *Cap surface*: *fibrous, grooves, scaly, smooth*
- (X3) *Cap color*: *brown, buff, cinnamon, grey, green, pink, purple, red, white, yellow*
- (X4) *Bruises:* *true, false*
- (X5) *Odor:* *almond, anise, creosote, fishy, foul, musty, none, pungent, spicy*
- (X6) *Gill attachment:* *attached, descending, free, notched*
- (X7) *Gill spacing:* *close, crowded, distant*
- (X8) *Gill size:* *broad, narrow*
- (X9) *Gill color:* *black, brown, buff, chocolate, grey, green, orange, pink, purple, red, white, yellow*
- (X10) *Stalk shape:* *enlarging, tapering*
- (X11) *Stalk root:* *bulbous, club, cup, equal, rhizomorphs, rooted, missing*
- (X12) *Stalk surface above ring:* *ibrous, scaly, silky, smooth*
- (X13) *Stalk surface below ring:* *ibrous, scaly, silky, smooth*
- (X14) *Stalk color above ring:* *brown, buff, cinnamon, grey, orange, pink, red, white, yellow*
- (X15) *Stalk color below ring:* *brown, buff, cinnamon, grey, orange, pink, red, white, yellow*

- (X16) *Veil type:* *partial, universal*
- (X17) *Veil color:* *brown, orange, white, yellow*
- (X18) *Ring number:* *none, one, two*
- (X19) *Ring type:* *cobwebby, evanescent, flaring, large, one, pendant, sheathing, zone*
- (X20) *Spore print color:* *black, brown, buff, chocolate, green, orange, purple, white, yellow*
- (X21) *Population:* *abundant, clustered, numerous, scattered, several, solitary*
- (X22) *Habitat:* *grasses, leaves, meadows, paths, urban, waste, woods*

Estos valores fueron discretizados asignándole un número distinto a cada posible valor de cada variable. Esta base de datos es de un tamaño significativamente superior a las anteriores.

6.1.5 Enfermedades coronarias

En otro de los problemas que se quiere resolver el objetivo es detectar presencia o no de enfermedad de corazón. Estos datos, tomados igualmente del UCI, corresponden a medidas tomadas a 303 pacientes del centro médico V.A. de Cleveland. En este caso se tienen 13 entradas, y su descripción es la siguiente:

- (X1) *Age: years*
- (X2) *Sex: 1 = male; 0 = female*
- (X3) *Cp: chest pain type*
  - *1: typical angina*
  - *2: atypical angina*
  - *3: non-anginal pain*
  - *4: asymptomatic*
- (X4) *Trestbps: resting blood pressure (in mm Hg on admission to the hospital)*

- (X5) *Chol: serum cholestoral in mg/dl*
- (X6) *Fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)*
- (X7) *Restecg: resting electrocardiographic results*
    - *0: normal*
    - *1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)*
    - *2: showing probable or definite left ventricular hypertrophy by Estes' criteria*
- (X8) *Thalach: maximum heart rate achieved*
- (X9) *Exang: exercise induced angina (1 = yes; 0 = no)*
- (X10) *Oldpeak = ST depression induced by exercise relative to rest*
- (X11) *Slope: the slope of the peak exercise ST segment*
    - *1: upsloping*
    - *2: flat*
    - *3: downsloping*
- (X12) *Ca: number of major vessels (0-3) colored by flourosopy*
- (X13) *Thal: 3 = normal; 6 = fixed defect; 7 = reversable defect*

6.1.6 Ionosfera

El último problema a resolver, también tomado del UCI, consiste en clasificar mediciones de radar de la capa de la ionosfera. Estos datos de radar fueron tomados por un sistema en Goose Bay, Labrador, Canadá. Este sistema consiste en un conjunto de 16 antenas de alta frecuencia con un poder de transmisión total del orden de 6.4 kilowatios. Los objetivos fueron los electrones libres en la ionosfera. Las mediciones “buenas” son aquellas que muestran evidencia de algún tipo de estructura en la ionosfera. Por su parte, las “malas” son aquellas que no muestran tal evidencia, es decir, las señales pasan a través de la ionosfera. A partir de 34 atributos, se quiere predecir este tipo de presencia o no de estructuras. En este caso se cuenta con 351 instancias.

6.1.7 Resumen

En la tabla 6.1 se puede ver un pequeño resumen de las características más importantes de los 6 problemas que se van a resolver. Es importante tener en cuenta que, en la mayoría de ellos, sólo se quiere tener una salida. Sin embargo, en el problema de Iris, se quiere realizar una clasificación en tres posibles casos, con lo que se generarán redes con 3 posibles salidas, una para cada caso.

	Numero de entradas	Número de instancias	Número de salidas
Apendicitis	8	106	1
Cáncer de mama	9	699	1
Iris	4	150	3
Setas venenosas	22	5644	1
Enfermedad del corazón	13	303	1
Ionosfera	34	351	1

Tabla 6.1. *Resumen de los problemas a resolver.*

Los atributos de todas estas bases de datos han sido normalizados entre 0 y 1 y los ejemplos separados en dos partes, tomando un 70% de los datos para realizar entrenamiento y un 30% para realizar validación.

6.2 Parámetros utilizados

En este trabajo se estudia la influencia de distintos parámetros en el rendimiento del sistema. Como ya se ha descrito, existen tres grupos de parámetros distintos: los que controlan la ejecución del algoritmo, los que limitan la complejidad de las redes y los que controlan la optimización de los pesos (si se usa).

Se estudiará la influencia de estos parámetros para cada problema. El objetivo de este estudio será determinar un conjunto de valores que produzcan buenos resultados, no para un problema en particular, sino para cualquier problema que se plantee. En general lo recomendable será ajustar estos parámetros al problema que se desee resolver, pero uno de los objetivos de este trabajo será establecer valores que permitan resolver problemas arbitrariamente complejos. Por esta razón, se están utilizando problemas de complejidades muy diversas, para determinar qué valores de estos parámetros se muestran buenos para resolver tipos diferentes de problemas.

6.2.1 Parámetros de PG

En este trabajo se estudian los parámetros nuevos del sistema creado, es decir, el segundo y el tercer grupo. Para ello, y previamente al estudio de estos parámetros, se han realizado diversas pruebas con el objetivo de mantener fijos el resto de parámetros que afectan al funcionamiento del algoritmo de PG. Como resultado de estos experimentos, se ha concluido que la configuración que ofrece mejores resultados es la siguiente:

- Tamaño de la población: 1000 individuos.
- Tasa de cruces: 95%.
- Probabilidad de mutación: 4%.
- Probabilidad de selección de función: 95%.
- Algoritmo de selección: Torneo de 2 individuos.
- Algoritmo de creación: *Ramped Half&Half*.

6.2.2 Parámetros que limitan la complejidad de las redes

En primer lugar, se estudiarán los parámetros que limitan la complejidad de las redes. La altura máxima del árbol o grafo, así como el número de entradas de cada neurona son los parámetros que influyen en este aspecto. Un valor muy bajo de estos parámetros impedirá la creación de redes lo suficientemente complejas como para resolver los problemas. Sin embargo, un valor demasiado alto provocará la creación de redes demasiado grandes, lo cual traerá problemas de pérdida de eficiencia en el proceso evolutivo. De la misma forma, la penalización al número de neuronas, no impone ningún límite al tamaño máximo de las redes, pero sin embargo guía notablemente el proceso evolutivo haciendo que se generen redes sencillas. Los valores que toma este parámetro son claves para conseguir un buen comportamiento de las redes que se generen. En los experimentos realizados se han tomado los siguientes valores para los parámetros:

- Para la altura máxima del árbol, un valor de 6 parece adecuado para conseguir redes con una complejidad suficiente para resolver problemas de una complejidad arbitraria. Por esta razón, se han tomado valores para este parámetro de 4, 5 y 6. En experimentos previos realizados se han hecho pruebas

con valores de 7 o incluso superiores. Sin embargo, la complejidad de las redes que se generan y las consecuentes exigencias de recursos computacionales provocan una pérdida de eficiencia enorme, hasta tal punto que los requisitos computacionales requeridos son tan altos que es casi imposible el trabajo con esos valores.

- Con respecto al número máximo de entradas de cada neurona, igualmente parece que un valor de 12 proveerá de suficiente complejidad para resolver cualquier problema. Por esta razón, se realizaron pruebas con este parámetro tomando valores desde 3 hasta 12. De igual forma que con la altura máxima, se realizaron experimentos con valores superiores a 12, pero los recursos computacionales exigidos por el sistema han provocado que estos valores no puedan ser tomados porque provocan graves pérdidas de eficiencia.
- Por su parte, para la penalización al número de neuronas se han tomado valores que varían de 0 hasta 0.1. Los experimentos demuestran que el valor de 0.1 es un valor muy alto, que no permite obtener redes satisfactorias.

Una vez establecidos valores para los parámetros que limitan la complejidad de las redes, es posible utilizar el sistema. Sin embargo, existe la posibilidad de utilizar también el sistema de optimización de redes. Para ello, habrá que fijar los valores de otra serie de parámetros.

6.2.3 Parámetros relativos al proceso de optimización de los pesos

Si se desea utilizar el sistema de optimización de pesos, habrá que fijar los valores de los parámetros involucrados en el mismo. Estos son los siguientes:

- N_{int}. Número de generaciones de PG que se esperan antes de ejecutar el proceso de optimización de pesos. Se toman los valores 1, 5, 10, 20, 40, 60, 80, 100 generaciones.
- F_{sel}. Porcentaje de individuos de la población de PG a los que se aplica el proceso de optimización de constantes. Se toman los valores 1%, 5%, 10%, 25% y 50%.
- N_{ga}. Tamaño de la población del algoritmo genético. Se toman los valores de 10, 25, 50, 75, 100, 125, 150, 175 y 200 individuos.

6.3 Resultados

En esta sección se describen los experimentos realizados, en primer lugar, para probar el correcto funcionamiento del sistema, pero también, como ya se ha explicado, para conseguir una configuración de parámetros que ofrezca un buen comportamiento del sistema ante problemas de diferente complejidad.

Para todos los experimentos realizados, se han medido los resultados que se obtienen con respecto al esfuerzo computacional, es decir, con respecto al número de ejecuciones de la función de ajuste.

Una de las características de los sistemas de CE es que son fuertemente estocásticos, siendo esta componente aleatoria intrínseca lo que hace que los resultados que ofrezcan puedan llegar a tener una alta variabilidad. Para minimizar esta componente aleatoria, para cada experimento mostrado en adelante se han realizado 20 ejecuciones distintas, obteniendo finalmente como resultado la media de los valores obtenidos (ECM) en estas 20 ejecuciones. Para ello, se ejecutó el algoritmo hasta un máximo de 500.000 evaluaciones de la función de ajuste un total de 20 veces. Durante la ejecución del algoritmo, la salida del mismo fue el individuo que mejor valor obtuvo en la validación hasta ese momento.

Dado que los problemas han sido normalizados entre 0 y 1, en todos los experimentos realizados la función de transferencia utilizada (que permanece fija, no se evoluciona con la red y es la misma para todas las neuronas) es la sigmoidal, que devuelve valores entre 0 y 1.

6.3.1 Uso de árboles

En esta sección se hallan los valores de los parámetros cuando se emplea el sistema utilizando árboles como forma de codificación.

En primer lugar, es necesario fijar los valores de los parámetros del sistema en sí, sin usar la optimización de constantes para, una vez fijados, poder utilizar estos valores para ejecutar el sistema con dicha optimización.

6.3.1.1 Sin optimización de los pesos

6.3.1.1.1 Altura máxima y número máximo de entradas

Los primeros experimentos realizados son aquellos en los que se intenta determinar cuál es la complejidad idónea del sistema para resolver problemas de dificultad arbitraria. Como ya se ha descrito, esta complejidad se mide según el efecto de dos parámetros: altura máxima de los árboles y número máximo de entradas de las neuronas.

Con respecto a la altura máxima, se han tomado valores variando de alturas de 4 a 6. Una altura inferior a 4 no permitiría la complejidad necesaria para resolver estos problemas. De igual forma, el valor mínimo que se ha tomado como número máximo de entradas de una neurona es de 3 entradas, el cual ya se puede suponer que es demasiado bajo como para obtener buenos resultados.

La figura 6.1 muestra los resultados obtenidos en la validación para la resolución de los problemas anteriormente descritos. Para ello, se ejecutó el algoritmo hasta un máximo de 500.000 evaluaciones de la función de ajuste. Durante la ejecución del algoritmo, la salida del mismo fue el individuo que mejor valor obtuvo en la validación hasta ese momento. En todos estos resultados se ha mantenido un tamaño de población constante de 1000 individuos y una penalización al número de neuronas de 0, es decir, sin optimización.

En la figura 6.1 puede apreciarse que el sistema es bastante independiente de los parámetros. A partir de 5 entradas los resultados son equivalentes y algún problema (ionosfera o setas venenosas) se comporta peor con altura 4. De igual manera, el sistema no presenta una diferencia significativa para valores de número máximo de entradas de 6 a 12. Por lo tanto, el sistema se presenta robusto en esos rangos para los parámetros: [5-6] para la altura máxima y [6-12] para el número de entradas.

Para proseguir con el resto de experimentos, es necesario adoptar un valor para estos parámetros, que se mantendrá constante. Por motivos de eficiencia, y para reducir el coste computacional, se ha adoptado una altura de 5 y 6 entradas como máximo a cada neurona como valores para estos parámetros. Estos valores han generado buenos resultados en todos los problemas.

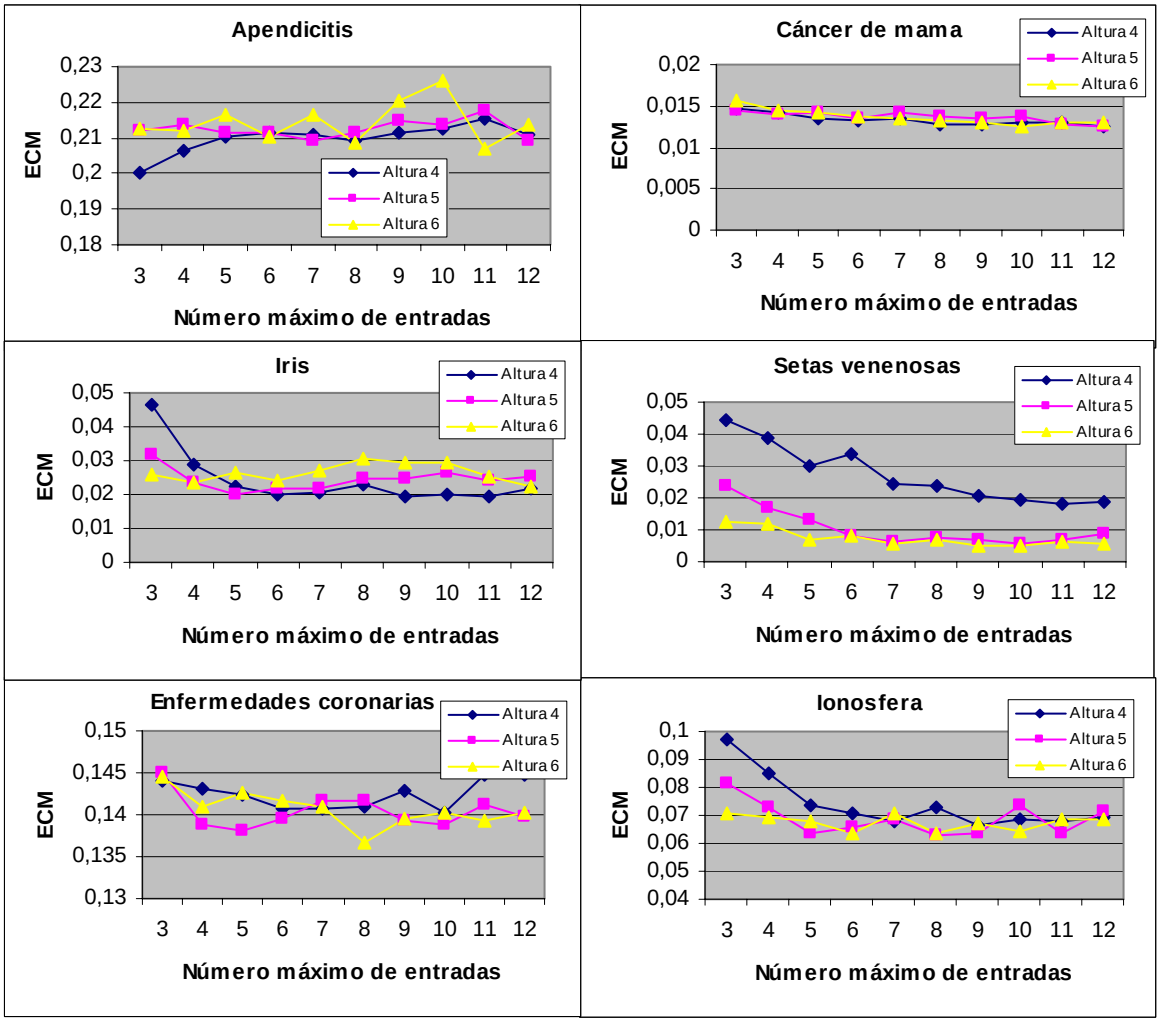


Figura 6.1. *Resultados de validación para distintos valores de altura y número de entradas*

6.3.1.1.2 Penalización al número de neuronas

Se han realizado también experimentos con un parámetro que es de vital importancia, la penalización al número de neuronas. Como se ha dicho antes, el valor de este parámetro se suma al valor de la función de ajuste multiplicado por el número de neuronas que tiene la RNA. De esta forma, se intenta obtener redes sencillas. Los valores que se han probado para este parámetro son 0.1, 0.001, 0.0001, 0.00001 y 0. El valor de 0 es el que se ha tomado para los experimentos realizados hasta el momento, puesto que se desea evaluar el sistema sin usar la penalización, para, posteriormente, estudiar el efecto de este parámetro.

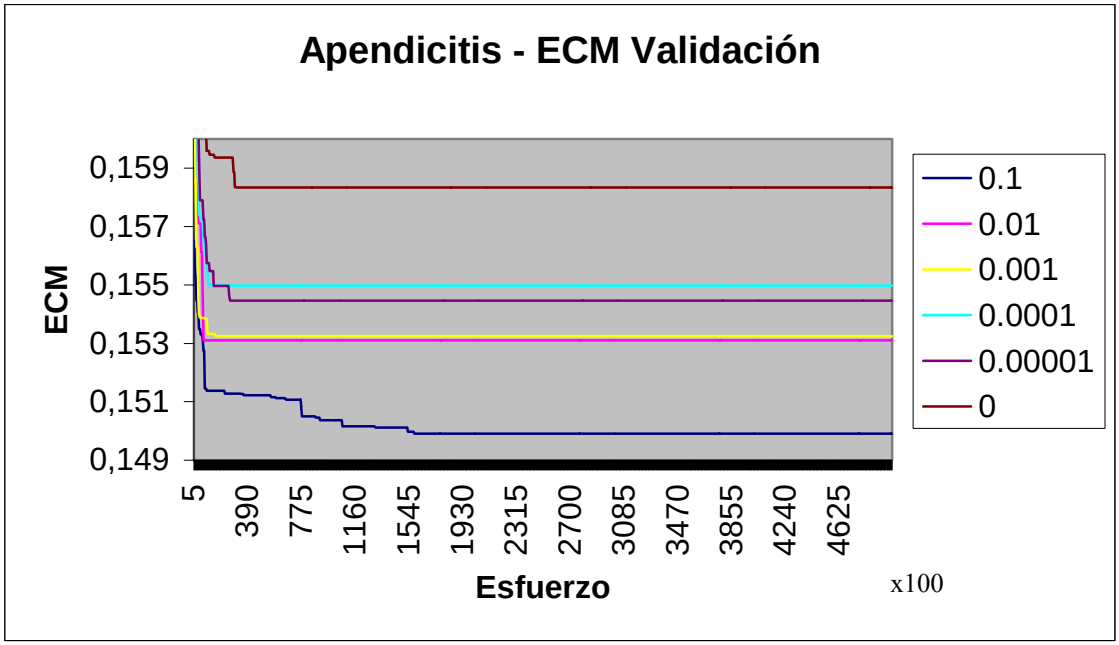


Figura 6.2. *ECM para distintos valores de penalización en el problema de apendicitis*

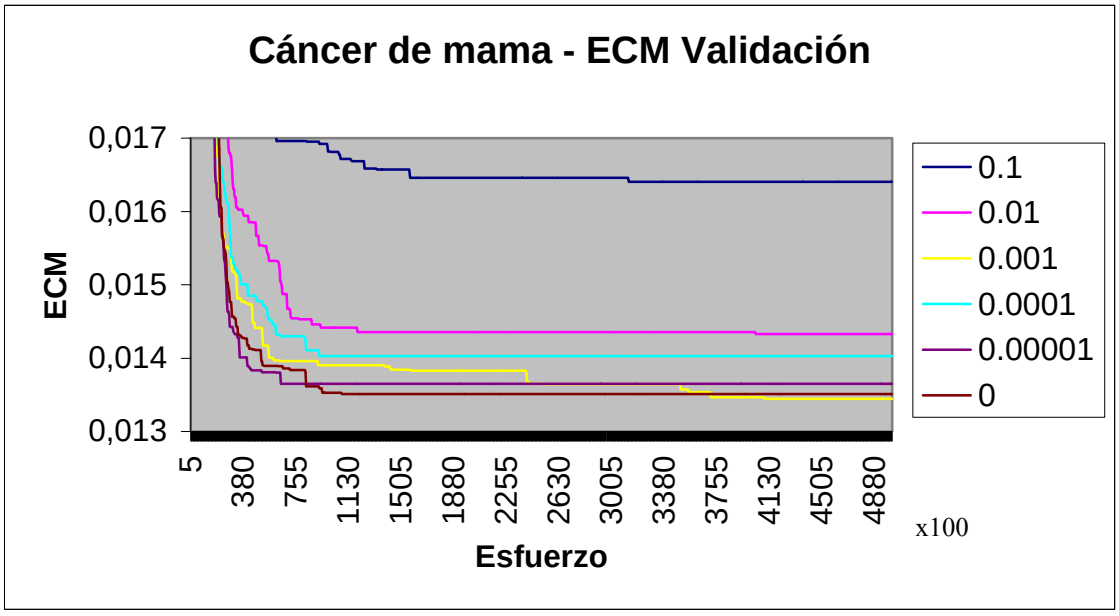


Figura 6.3. *ECM en validación para distintas penalizaciones en el problema de cáncer de mama*

En las figuras 6.2 a 6.7 se puede ver la evolución seguida por el sistema, en los distintos problemas, con los distintos valores de penalización. En estas gráficas se muestran los valores del ECM (sin sumar la penalización) de la validación correspondiente al mejor individuo encontrado por el algoritmo. En el eje X se muestra el esfuerzo computacional a que se refiere cada valor, medido como el número de ejecuciones de la función de ajuste, hasta un máximo de 500000 ejecuciones. En estas

gráficas puede observarse claramente que el error obtenido es menor cuanto menor es la penalización Estos valores de error son el resultado de realizar la media de los errores obtenidos tras realizar 20 ejecuciones distintas para cada valor de penalización en cada problema.

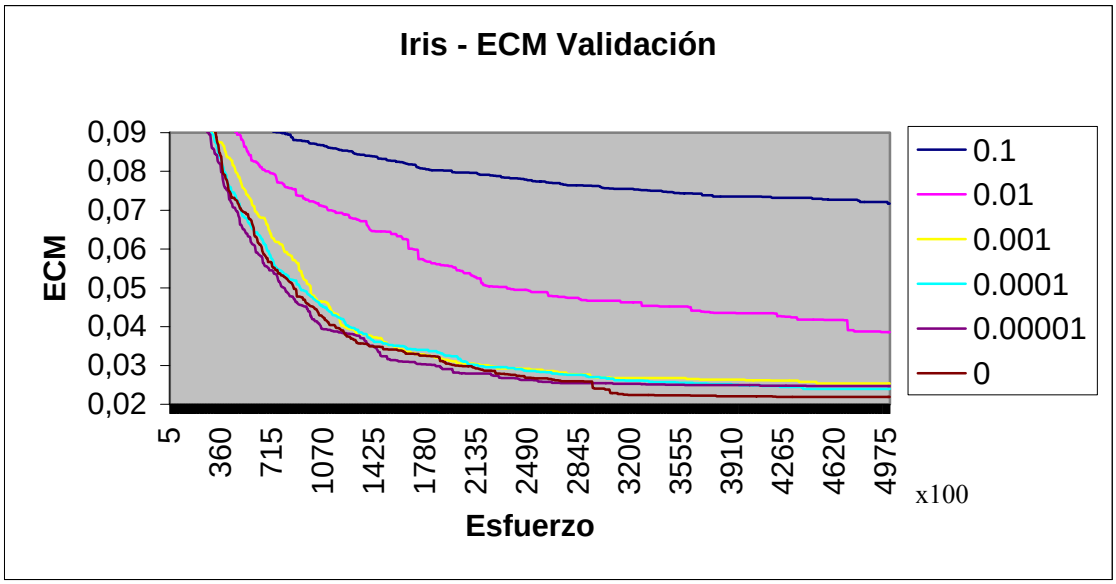


Figura 6.4. *ECM en la validación para distintos valores de penalización en el problema de iris*

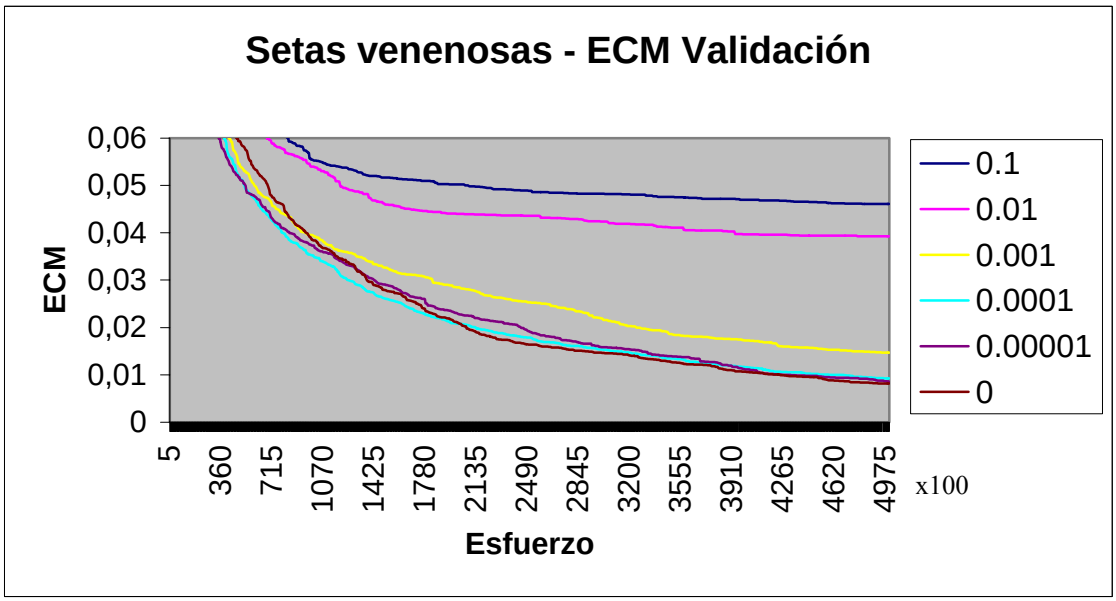


Figura 6.5. *ECM en validación para distintas penalizaciones en el problema de setas venenosas*

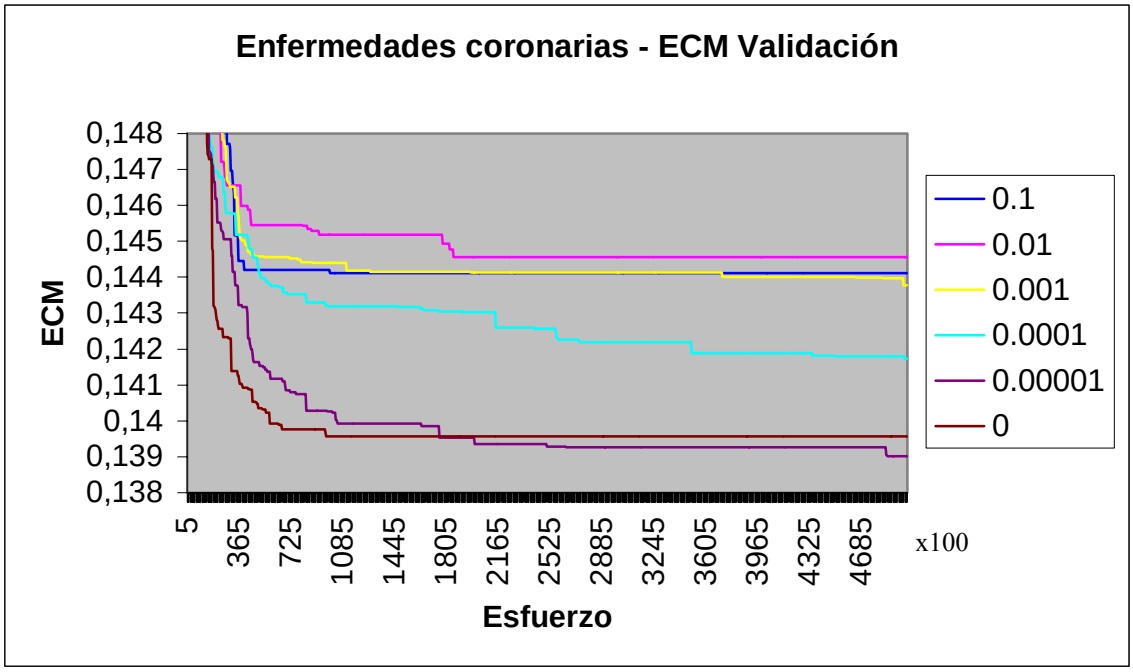


Figura 6.6. *ECM en la validación para distintos valores de penalización en el problema de enfermedad del corazón*

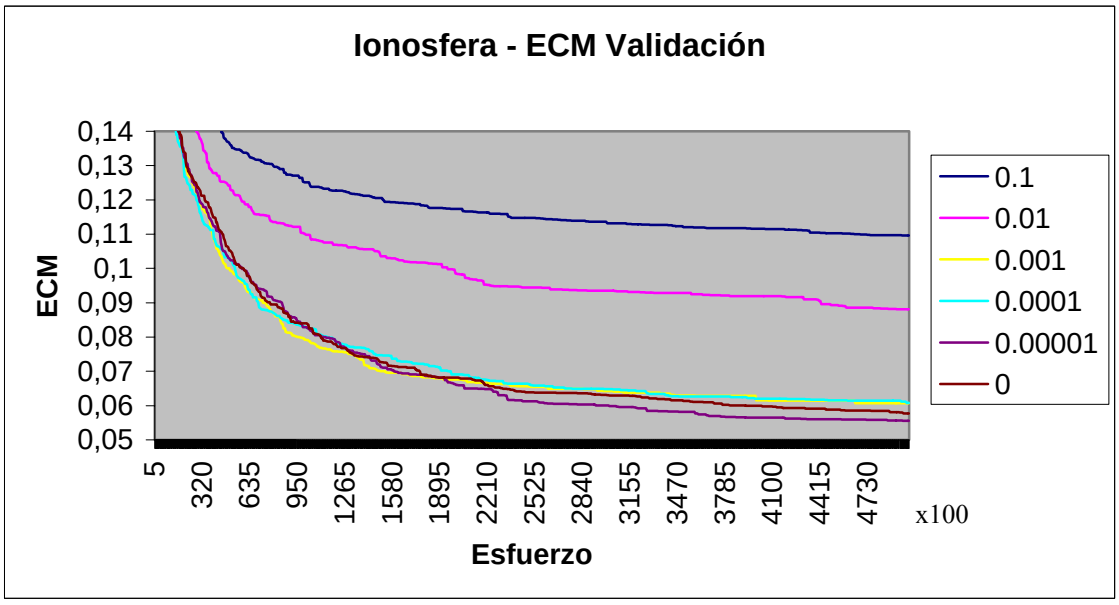


Figura 6.7. *ECM en validación para distintos valores de penalización en el problema de ionosfera*

En estas figuras puede apreciarse el coste computacional necesario para obtener esas soluciones al problema. Mientras que en los problemas de setas venenosas e ionosfera, por ser los más complejos, el sistema seguía mejorando tras la ejecución de 500.000 veces la función de ajuste, en otros problemas, como predicción de cáncer de

mama o apendicitis, la mejor solución se había encontrado mucho antes de este punto, con lo que el resto del proceso de entrenamiento sólo serviría para producir un efecto de sobreentrenamiento. Para evitar esto, y como se ha dicho, se ha mantenido la salida del sistema como el individuo que mejor resultado ha producido en validación.

En las figuras 6.8 a 6.13 puede verse el número de neuronas que tenían las redes que daban los mejores resultados (que corresponden a los valores de error de las figuras 6.2 a 6.7). Lógicamente, el número de neuronas de la red es menor conforme mayor es la penalización aplicada. Con una penalización muy alta (0.1), el sistema solo generará redes con muy pocas neuronas. Con una penalización demasiado baja, el sistema genera redes con muchas neuronas, pero, como se ha visto anteriormente, el error obtenido es menor.

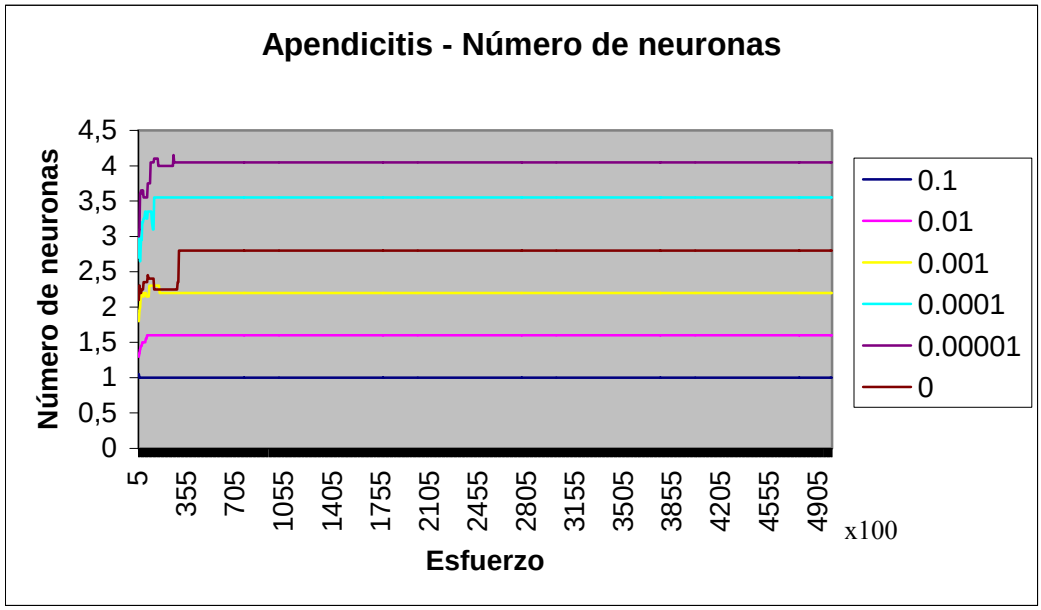


Figura 6.8. *Número medio de neuronas para distintas valores de penalización en el problema de apendicitis*

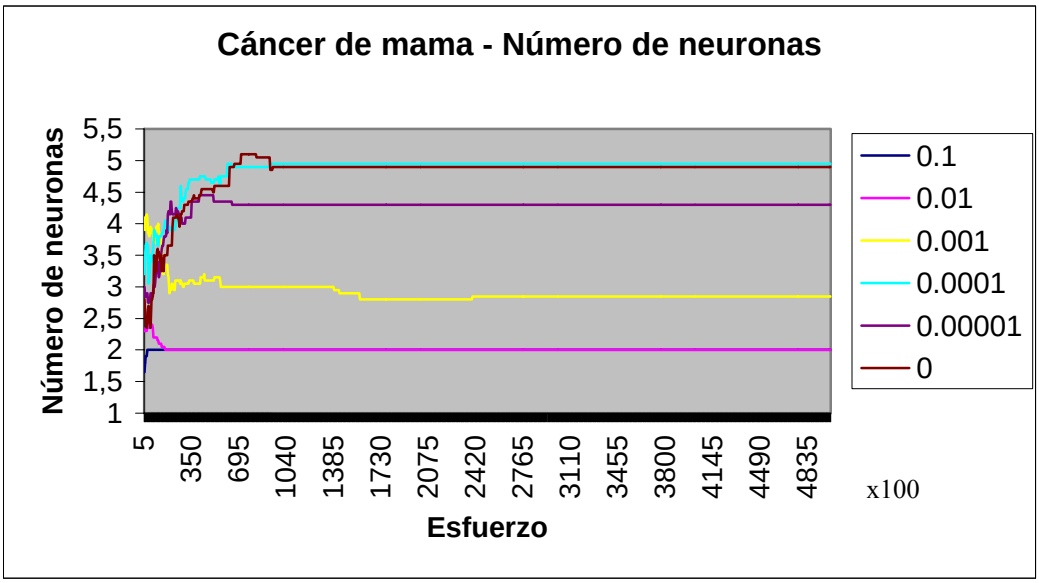


Figura 6.9. *Número medio de neuronas para distintos valores de penalización en el problema de cáncer de mama*

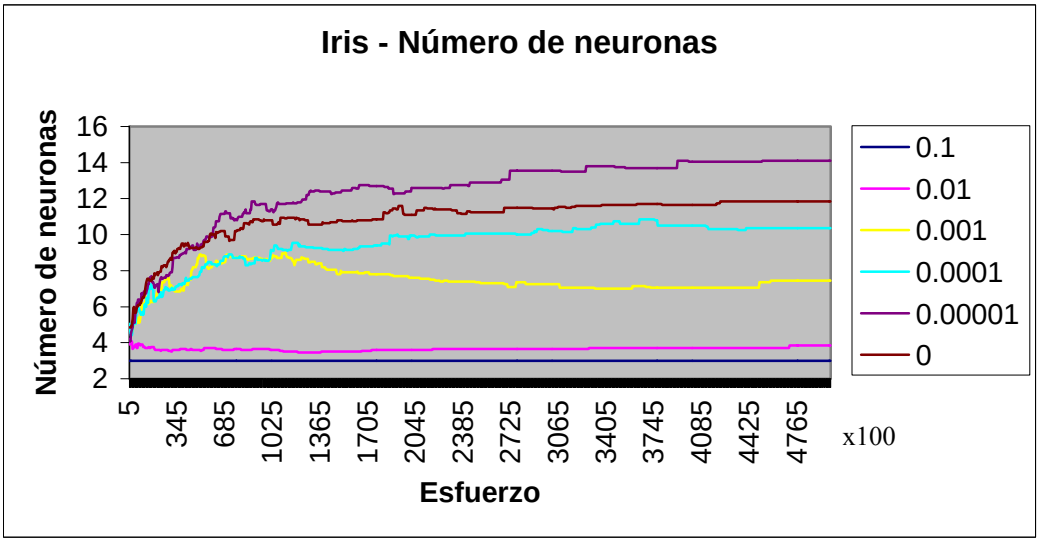


Figura 6.10. *Número medio de neuronas para distintos valores de penalización en el problema de flores iris*

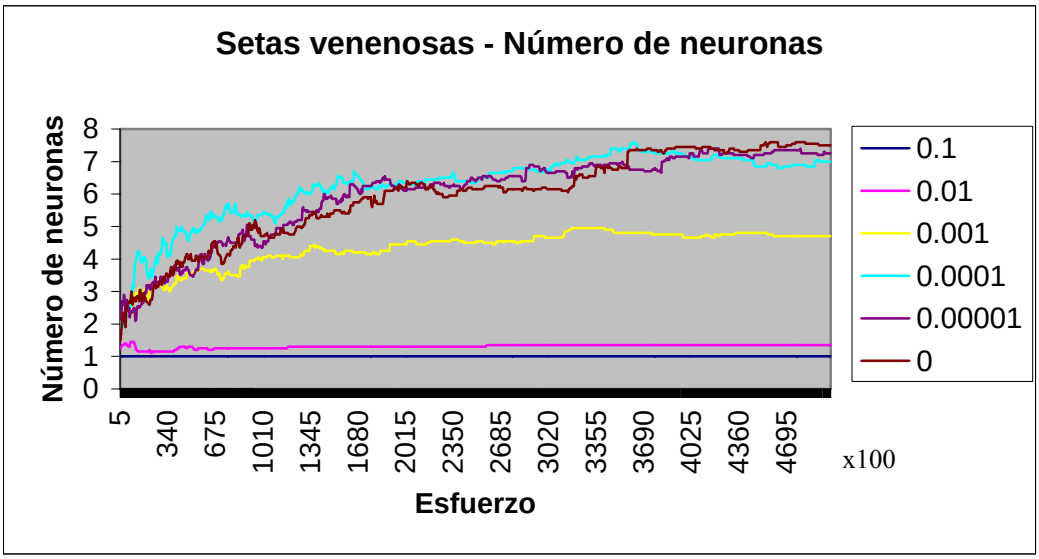


Figura 6.11. *Número medio de neuronas para distintos valores de penalización en el problema de setas venenosas*

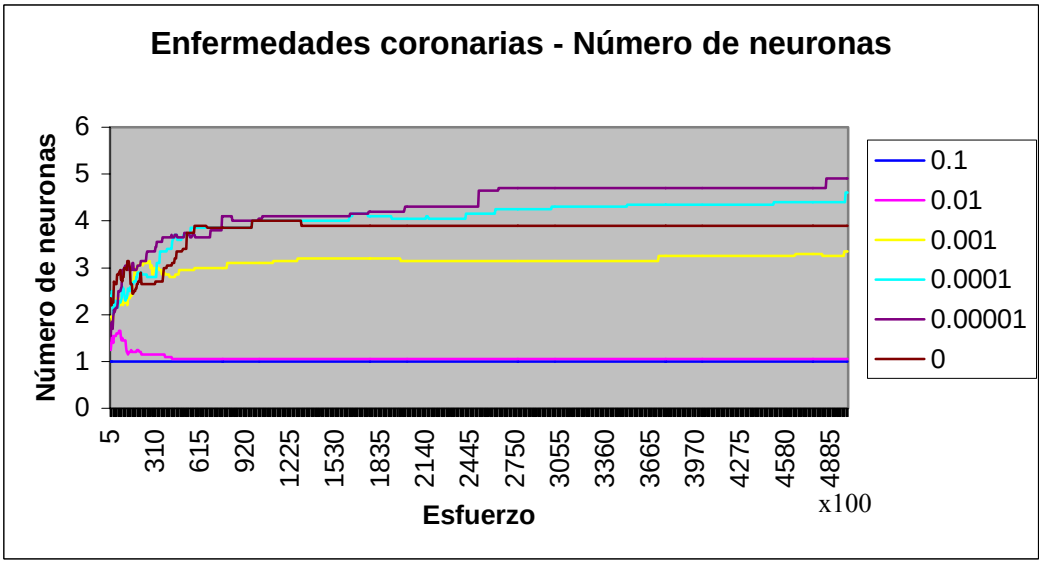


Figura 6.12. *Número medio de neuronas para distintos valores de penalización en el problema de enfermedades de corazón*

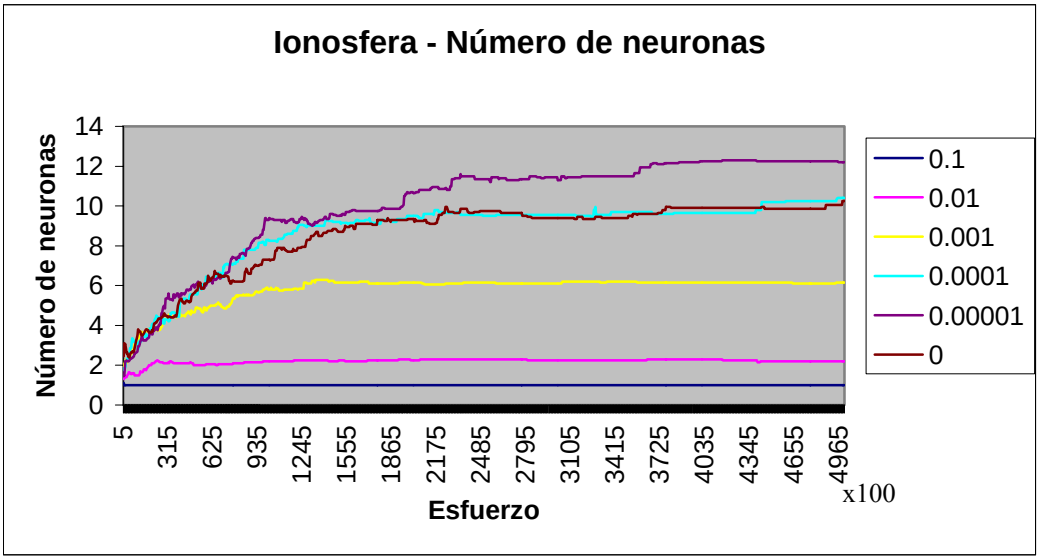


Figura 6.13. *Número medio de neuronas para distintos valores de penalización en el problema de ionosfera*

La tabla 6.2 muestra un resumen de todas estas gráficas. En ella se muestran el número de neuronas y de conexiones medio obtenido para cada valor de penalización en cada problema. Además, se puede ver también las medias de errores cuadráticos medios obtenidos en el entrenamiento y en la validación después de las 500000 ejecuciones de la función de ajuste, en 20 experimentos distintos realizados. El número de neuronas que se muestra se refiere a neuronas ocultas y de salida, sin contar neuronas de entrada, puesto que en estas no se realiza ningún procesamiento.

A la vista de esta tabla, sólo uno de los problemas ha ofrecido buenos resultados con un valor alto de penalización. En el resto de los problemas, los mejores resultados han sido obtenidos con valores bajos de este parámetro, llegando un punto a partir del cual las diferencias en errores no son muy significativas. En general, puede decirse que valores menores de 0.001 no producen grandes variaciones en los resultados, con lo que el sistema se mantiene robusto en los mismos, pudiéndose escoger cualquier valor en ese rango [0-0.001]. En adelante se adoptará un valor de penalización intermedio, de 0.00001, que ha mostrado buenos resultados en la mayoría de los problemas.

		0.1	0.01	0.001	0.0001	0.00001	0
Apendicitis	Neuronas	1	1.6	2.2	3.55	4.05	2.8
	Conexiones	3.05	3.45	5.5	9.6	11.6	7.9
	Entrenamiento	0.080083	0.068735	0.059800	0.055485	0.055139	0.052832
	Validación	**0.149913**	0.153103	0.153251	0.154981	0.154466	0.158337
Cáncer de mama	Neuronas	2	2	2.85	4.95	4.3	4.9
	Conexiones	6.05	6.75	10.35	16.95	14.45	16.95
	Entrenamiento	0.054054	0.021042	0.019398	0.021153	0.021932	0.020549
	Validación	0.016404	0.014327	**0.013446**	0.014027	0.013650	0.013508
Iris	Neuronas	3	3.85	7.35	10.35	11.85	14.1
	Conexiones	8.8	10.7	21.23	33	34.25	41.95
	Entrenamiento	0.063018	0.037535	0.023231	0.021895	0.018216	0.019512
	Validación	0.071735	0.038576	0.025357	0.023964	0.024657	**0.021883**
Setas venenosas	Neuronas	1	1.35	4.7	7	7.25	7.5
	Conexiones	4.35	6	18.15	27.65	27.4	29.35
	Entrenamiento	0.049114	0.041824	0.015163	0.009725	0.008761	0.008441
	Validación	0.046069	0.039260	0.014725	0.009256	0.008493	**0.008143**
Enfermedades coronarias	Neuronas	1	1.05	3.35	4.6	3.9	4.9
	Conexiones	3.85	4.15	12.45	17.15	14.05	17.9
	Entrenamiento	0.111395	0.110298	0.090913	0.087188	0.085101	0.098505
	Validación	0.144119	0.144564	0.143769	0.141717	**0.139023**	0.139564
Ionosfera	Neuronas	1	2.2	6.1	10.4	10.25	12.2
	Conexiones	4.55	8.4	24.65	41	41.45	48.4
	Entrenamiento	0.109343	0.080765	0.041037	0.038350	0.037155	0.038113
	Validación	0.109653	0.088133	0.060709	0.060848	**0.055604**	0.057700

Tabla 6.2. *Número medio de neuronas, conexiones y errores de entrenamiento y validación para cada valor de penalización y cada problema*

6.3.1.2 Optimizando los pesos con AA.GG.

Una vez se han tomado valores para el sistema en sí, incluyendo aquellos que afectan al funcionamiento del algoritmo de PG (como tamaño de población o probabilidad de mutación), y aquellos que afectan a la complejidad de las redes que genera el sistema, ya puede ser ejecutado con estos parámetros en una gran variedad de problemas distintos. Sin embargo, si se desea utilizar el sistema de optimización de constantes, será necesario realizar el mismo proceso para establecer qué valores de los parámetros son los idóneos para resolver problemas de diferente complejidad.

6.3.1.2.1 Rangos de los pesos

En primer lugar será necesario establecer qué rangos de valores pueden tomar los pesos de las conexiones. Cuando no se usa optimización de constantes, esto no es necesario, puesto que, tomando constantes en el rango [-4, 4], recomendado en [Fahlman 1988], estas sufren diversas operaciones como consecuencia de ser parte de

árboles aritméticos para dar lugar a los valores de los pesos. Por esta razón, los valores resultantes pueden exceder este rango inicial de [-4, 4].

Sin embargo, esto no ocurre cuando se utiliza el sistema de optimización, puesto que ya no existen estos subárboles aritméticos, con lo que los pesos de las conexiones son directamente los valores de las constantes, que estarán en el rango especificado. Por esta razón ahora cobra especial importancia el rango de las constantes, porque limita enormemente el rango que pueden tomar los pesos, y eso antes no ocurría, puesto que los valores de los pesos podían salir del rango inicial si era necesario.

Por lo tanto, los primeros experimentos están orientados a establecer el rango en el que estarán los pesos de las conexiones. En estos experimentos se mantiene el resto de los parámetros con un valor constante de N_{int}=5, F_{sel}=25, N_{ga}=50 y N_{gagen}=50, es decir, que se aplica el método cada 5 generaciones del algoritmo de PG, sobre el 25% de la población (es decir, sobre 250 individuos), y con un tamaño de población del AG de 50 individuos y una ejecución de un máximo de 50 generaciones del mismo. Sin embargo, como ya se ha explicado, este último parámetro apenas tiene influencia en la ejecución del sistema.

	Apendicitis	Cáncer de mama	Iris	Setas venenosas	Enfermedades coronarias	Ionosfera
5	0,153296	**0,014033**	0,074318	0,050959	**0,138473**	0,094282
10	0,157146	0,01438	0,048462	0,037134	0,146487	0,089621
15	0,16138	0,015018	**0,029359**	0,033486	0,151675	0,089435
20	0,153216	0,01572	0,035118	0,029051	0,152173	**0,076892**
25	0,151568	0,015873	0,038101	0,028358	0,154135	0,083339
30	0,149415	0,017777	0,034181	0,02622	0,162617	0,077314
35	0,147177	0,01625	0,030693	0,02878	0,154824	0,078194
40	**0,144174**	0,01656	0,039733	0,027036	0,159892	0,077313
45	0,151048	0,016888	0,035441	0,024127	0,159613	0,084639
50	0,149694	0,018019	0,042472	0,024493	0,159991	0,081943
100	0,150528	0,017954	0,04139	0,027326	0,163021	0,080822
150	0,153138	0,018709	0,046441	0,025322	0,167827	0,084301
200	0,155381	0,018322	0,034405	0,02324	0,17407	0,083326
250	0,152794	0,019594	0,042601	0,025882	0,160385	0,086791
300	0,151446	0,019263	0,03632	0,023081	0,165749	0,085072
350	0,153894	0,020215	0,036755	0,026852	0,166101	0,085676
400	0,151724	0,019775	0,039067	**0,022761**	0,163752	0,084519
450	0,158917	0,020332	0,045935	0,02361	0,172902	0,081753
500	0,154574	0,020171	0,044522	0,025749	0,164713	0,09047

Tabla 6.3. Errores en validación cometidos para diferentes valores de pesos

Estos experimentos se realizan tomando [-X, X] como intervalo en los que estarán los valores de los pesos. Los valores de X, es decir, el valor absoluto de los límites inferior y superior, tomados para el intervalo son 5, 10, 15, 20, 25, 30, 35, 40, 45, 50,

100, 150, 200, 250, 300, 350, 400, 450 y 500, valores que son suficientemente variados como para realizar un estudio a fondo con valores pequeños y grandes.

En la tabla 6.3 puede verse diferentes valores de ECM cometidos tomando distintos valores para el rango de los pesos.

Como puede verse, el sistema es robusto e independiente del valor que se tome para este parámetro, puesto que las diferencias encontradas en los resultados son muy pequeñas. Gráficamente, esto puede verso en la figura 6.14.

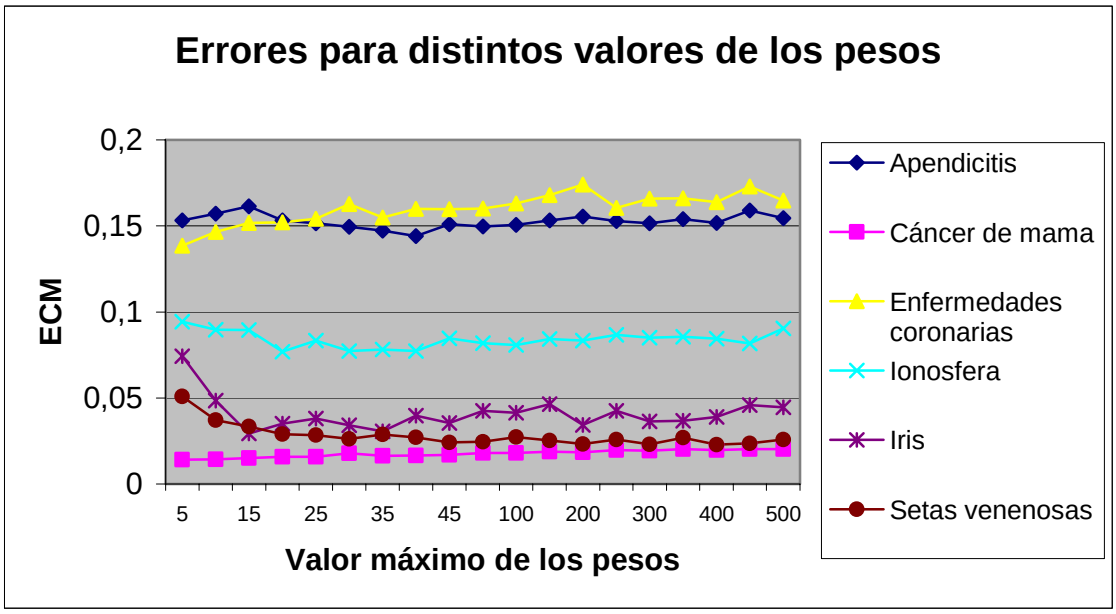


Figura 6.14. *Errores en validación para distintos valores de los pesos*

En esta gráfica puede verse que no existe una tendencia clara en este aspecto, puesto que el sistema se mantiene estable para un rango de valores de los pesos. A la vista de esta gráfica, valores menores de 15 producen malos resultados para ciertos problemas (iris, setas venenosas), y los superiores a 35 también provocan la obtención de peores resultados para otros problemas (iris, enfermedades coronarias, ionosfera). Dentro de este rango [15-35], el sistema se mantiene estable y ofrece un buen comportamiento. Por esta razón, para efectuar el resto de experimentos, se tomó un valor intermedio que está dentro de ese rango y que muestra buenos resultados para todos los problemas. El valor tomado es de 20; es decir, los pesos se tomarán en un intervalo de [-20, 20].

6.3.1.2.2 Número de individuos a optimizar y frecuencia de la optimización

En esta sección se estudian los parámetros que dictan cuántas veces se aplicará el algoritmo de optimización de pesos. Dado que es este algoritmo el que realiza modificaciones en los valores de los pesos, puesto que ahora ya no existen subárboles aritméticos y evolutivamente el único cambio en los pesos consiste en el intercambio con otros pesos de otro individuo, este proceso de optimización se convierte en algo esencial para el desarrollo de las redes. Por esta razón ha de estudiarse minuciosamente cuántas veces se debe de ejecutar, puesto que esto influye notablemente en los resultados obtenidos y en la eficiencia del sistema, ya que si se ejecuta demasiadas veces se provoca una pérdida de eficiencia. Por esta razón, se estudia de manera conjunta el efecto de los dos parámetros que rigen este proceso, N_{int} y F_{sel}, que controlan cada cuánto se lanza este proceso y sobre cuántos individuos se realiza. Para la realización de estos experimentos se mantuvieron unos valores constantes para el resto de los parámetros no estudiados hasta el momento, es decir, N_{ga}=50 y N_{gagen}=50.

Las figuras 6.15 a 6.20 muestran una comparativa, para cada problema, de los resultados obtenidos para distintos valores de ambos parámetros.

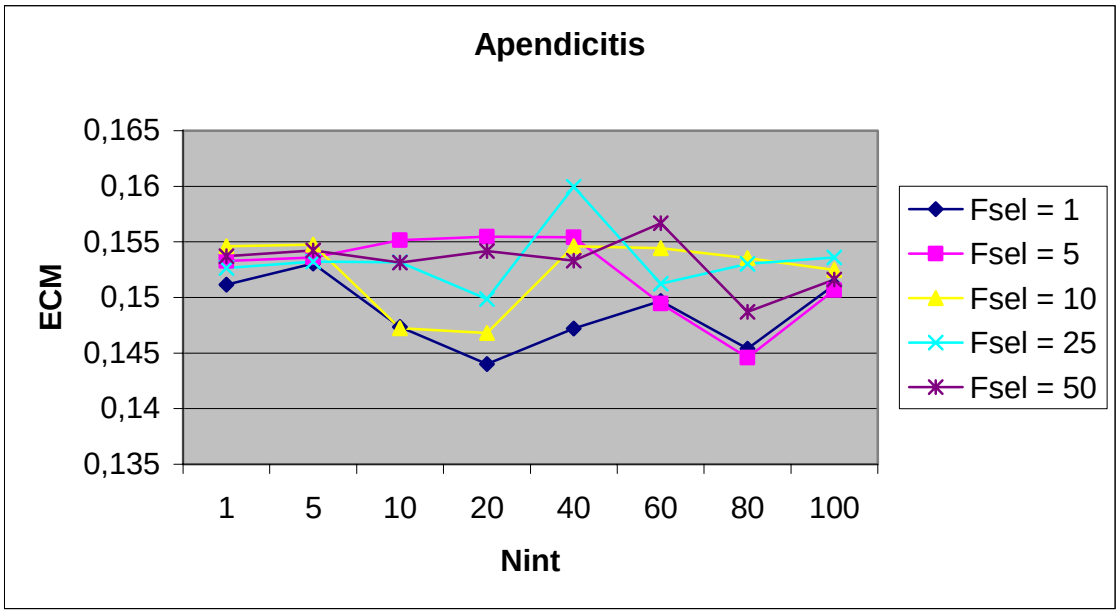


Figura 6.15. *Comparativa de los errores con diferentes valores de N_{int} y F_{sel} para el problema de apendicitis*

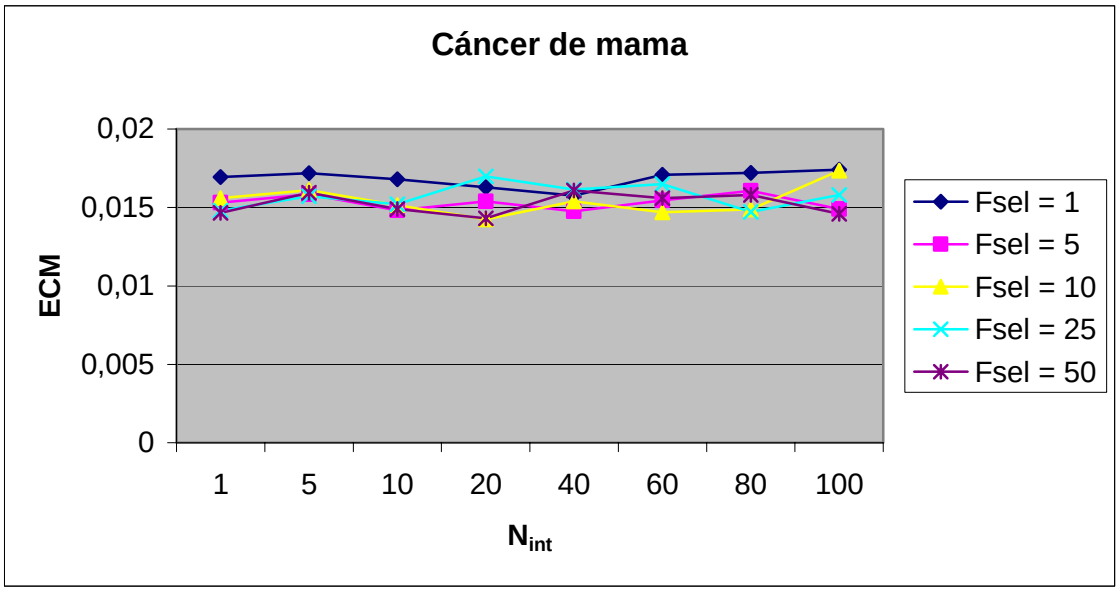


Figura 6.16. *Comparativa de los errores con diferentes valores de N_{int} y F_{sel} para el problema de cáncer de mama*

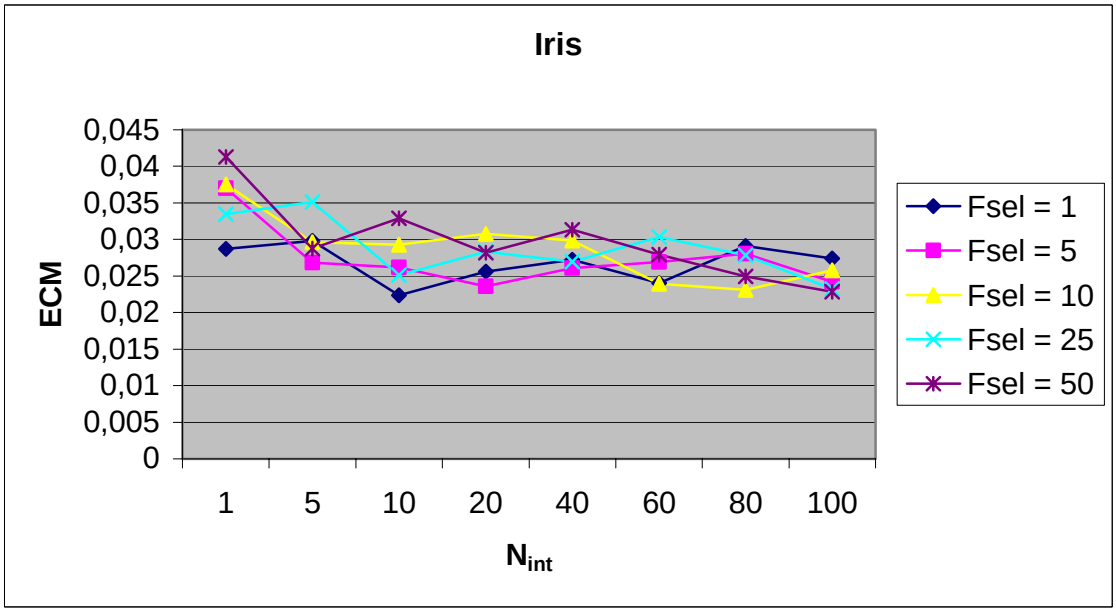


Figura 6.17. *Comparativa de los errores con diferentes valores de N_{int} y F_{sel} para el problema de flores iris*

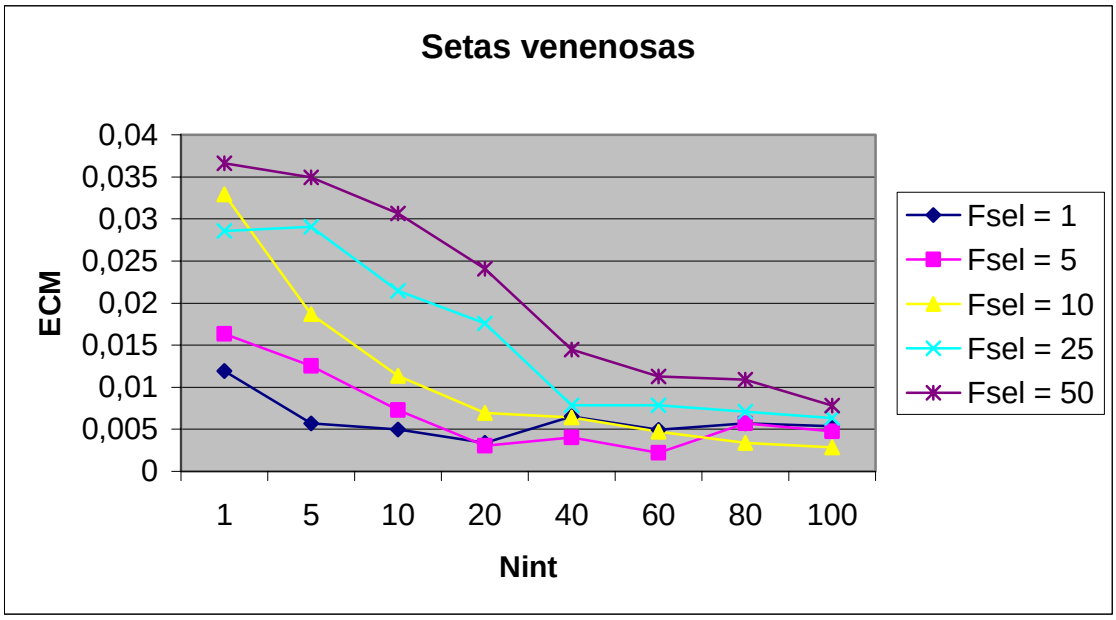


Figura 6.18. *Comparativa de los errores con diferentes valores de N_{int} y F_{sel} para el problema de setas venenosas*

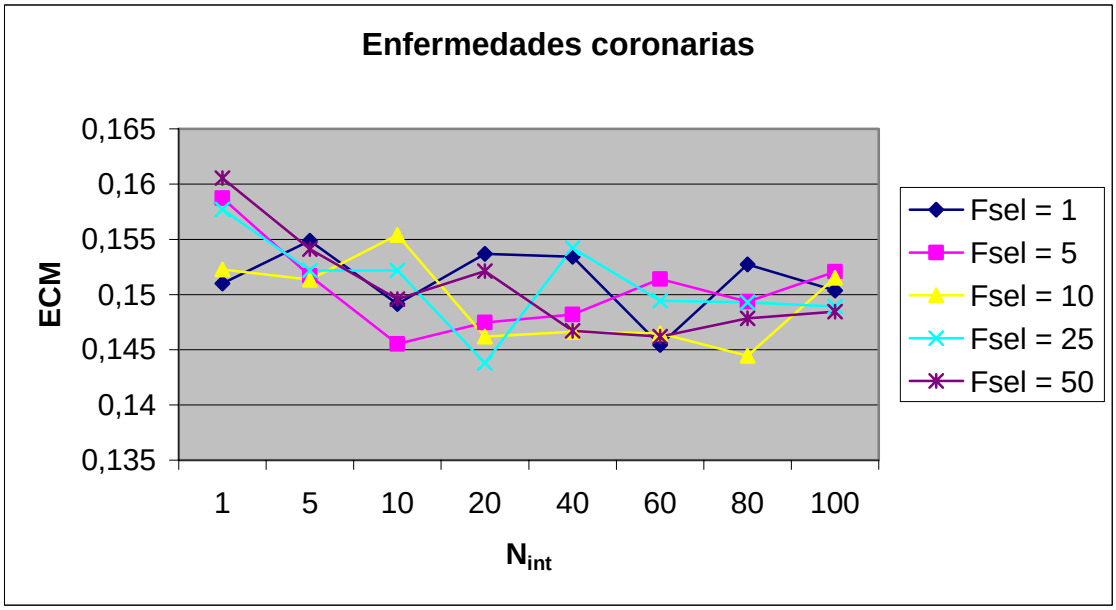


Figura 6.19. *Comparativa de los errores con diferentes valores de N_{int} y F_{sel} para el problema de enfermedades coronarias*

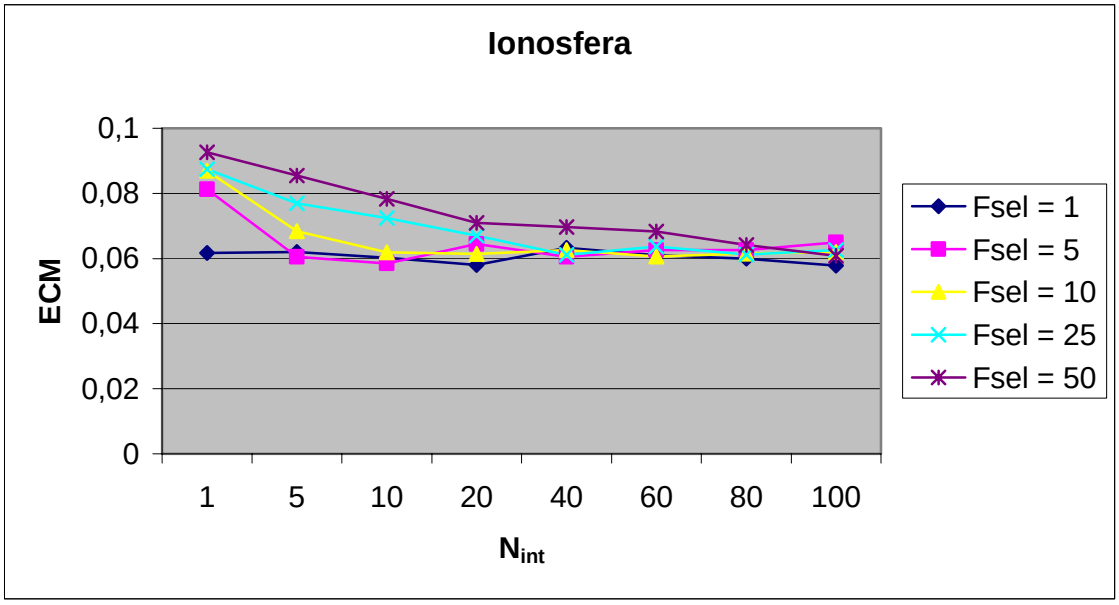


Figura 6.20. *Comparativa de los errores con diferentes valores de N_{int} y F_{sel} para el problema de ionosfera*

Nuevamente, en estas gráficas se puede apreciar la robustez del sistema; es decir, que dentro de un determinado rango para estos parámetros los resultados del sistema no ofrecen ninguna diferencia significativa. Según puede verse en estas gráficas, para valores de N_{int} superiores 20 y de F_{sel} inferiores o iguales a 10, se obtiene dicha estabilidad. Por tanto, se adoptan dichos rangos como los recomendados para estos parámetros, con valores de N_{int} en el rango [20-100] y de F_{sel} en el rango [1-10]. Para efectuar el resto de los experimentos, se toman unos valores de N_{int}=80 y F_{sel}=10, que ofrecen buenos resultados en la mayoría de los problemas.

Con estos valores se consigue que el sistema tenga un buen rendimiento. Sin embargo, cabe plantearse qué perdida de eficiencia se está teniendo cuando se varían estos parámetros. La figura 6.21 muestra el número de generaciones del AG ejecutadas al alcanzar las 500.000 evaluaciones del fichero de entrenamiento, para distintos valores del parámetros N_{int}, manteniendo los valores constantes de F_{sel}=10, N_{ga}=50 y N_{gagen}=50. Como se puede ver, a medida que se reduce el valor de N_{int}; osea, a medida que aumenta la frecuencia en la que se ejecuta el algoritmo de optimización, lógicamente aumenta el número de generaciones ejecutadas del AG. Sin embargo, este aumento no es proporcional, como cabría esperarse, sino que parece crecer de forma exponencial al decrecimiento de N_{int}. Este aumento sería proporcional si el número de generaciones del AG ejecutadas por el proceso de optimización fuese fijo. Sin embargo, no lo es, sino que varía cada vez que se ejecuta; es decir, la ejecución del AG se para cuando se han

realizado 5 ejecuciones consecutivas sin mejorar el ajuste. Por lo tanto, si el número de generaciones del AG ejecutadas es muy alto, esto quiere decir que, en general, cada proceso de optimización tarda más en alcanzar estas 5 generaciones sin cambio, es decir, el proceso de búsqueda de los valores de los pesos es más costoso.

Al utilizar valores de N_{int} bajos, se está ejecutando el proceso de optimización con una frecuencia muy alta, es decir, muchas veces, y, por lo que se puede ver en la figura 6.21, el número de generaciones ejecutadas es más alto de lo que debería, por lo que, según lo explicado anteriormente, en cada una de estas ejecuciones se está invirtiendo más esfuerzo en el proceso de búsqueda de los valores de los pesos. Por contra, si se toman valores de N_{int} altos, este proceso de optimización será menos costoso, es decir, se invertirá menos esfuerzo en encontrar estos valores de los pesos.

Todo esto lleva a que, a medida que crece el valor del parámetro N_{int} el sistema, al ejecutar el proceso de optimización menos veces, centra sus esfuerzos en encontrar redes utilizando los valores de los pesos disponibles, es decir, la búsqueda que realiza el sistema se basa más en el diseño de la arquitectura. Por la contra, a medida que disminuye el valor de este parámetro, al ejecutar el sistema de optimización más veces, la búsqueda se centra más en hallar buenos valores para los pesos de las redes existentes.

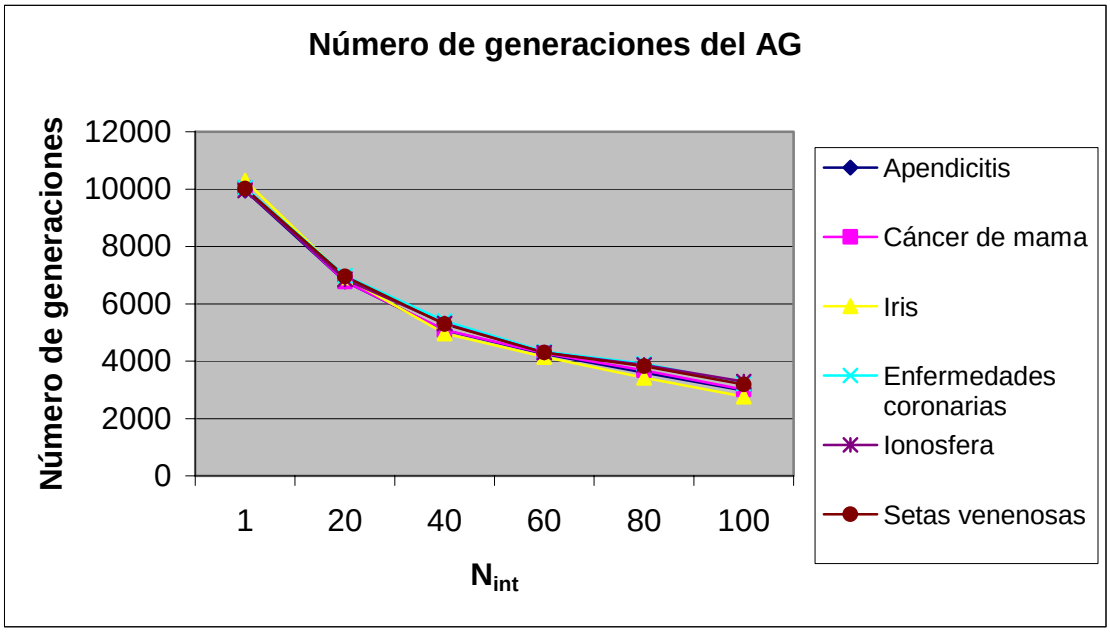


Figura 6.21. *Número de generaciones del AG ejecutadas para distintos valores de* N_{int}

Un efecto similar ocurre con el parámetro F_{sel}, que rige a cuántos individuos se aplica el proceso de optimización. En la figura 6.22 se estudia el efecto tomando diferentes valores de este parámetro, y manteniendo constantes el resto de parámetros como N_{int}=80, N_{ga}=50 y N_{gagen}=50. Como puede verse, el crecimiento del número de generaciones a medida que crece este parámetro no es proporcional, sino que sigue la curva de una exponencial inversa, es decir, el crecimiento de la curva se va atenuando a medida que aumenta el valor de F_{sel}, lo que quiere decir que no se ejecutan tantas generaciones del AG como cabría suponerse, osea, se alcanzan antes unos valores de los pesos que no se pueden seguir mejorando en este proceso de optimización. Esto es consecuencia de aplicar el proceso de optimización a un número mayor de individuos, con lo que la siguiente vez que se ejecute, habrá más posibilidades de que este proceso se aplique sobre individuos que ya están optimizados, al menos en parte. De igual manera a lo expuesto anteriormente, si este parámetro toma valores altos, la búsqueda se centra más en buscar valores para los pesos de las conexiones. Por contra, si toma valores bajos, la búsqueda se centra más en el diseño de la arquitectura de la red. Este efecto explica la robustez que tiene el sistema al cambio de valores en estos dos parámetros, lo cual no producirá grandes variaciones en el resultado, sino en la forma de búsqueda de soluciones.

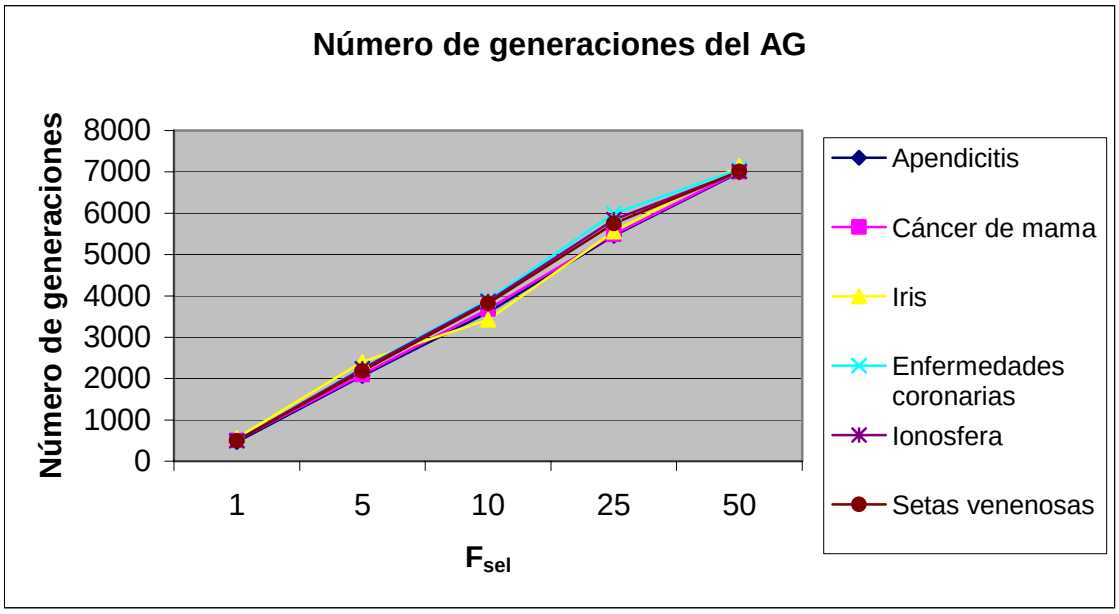


Figura 6.22. *Número de generaciones del AG ejecutadas para distintos valores de* F_{sel}

6.3.1.2.3 Tamaño de la población del algoritmo genético

El último parámetro que se va a estudiar respecto al proceso de optimización de constantes se corresponde al tamaño de la población del algoritmo genético, es decir, el parámetro N_{ga}. Este parámetro, junto con el parámetro N_{gagen}, que, como se ha explicado, se ha mostrado que no es influyente en los experimentos previos, dicta el esfuerzo que se realiza en la búsqueda de los valores de los pesos de las conexiones.

En la figura 6.23 puede verse una comparativa de los valores de error obtenidos para distintos valores de este parámetro, dejando el resto de parámetros con sus valores recomendados anteriormente N_{int}=80, F_{sel}=10 y N_{ga}=50.

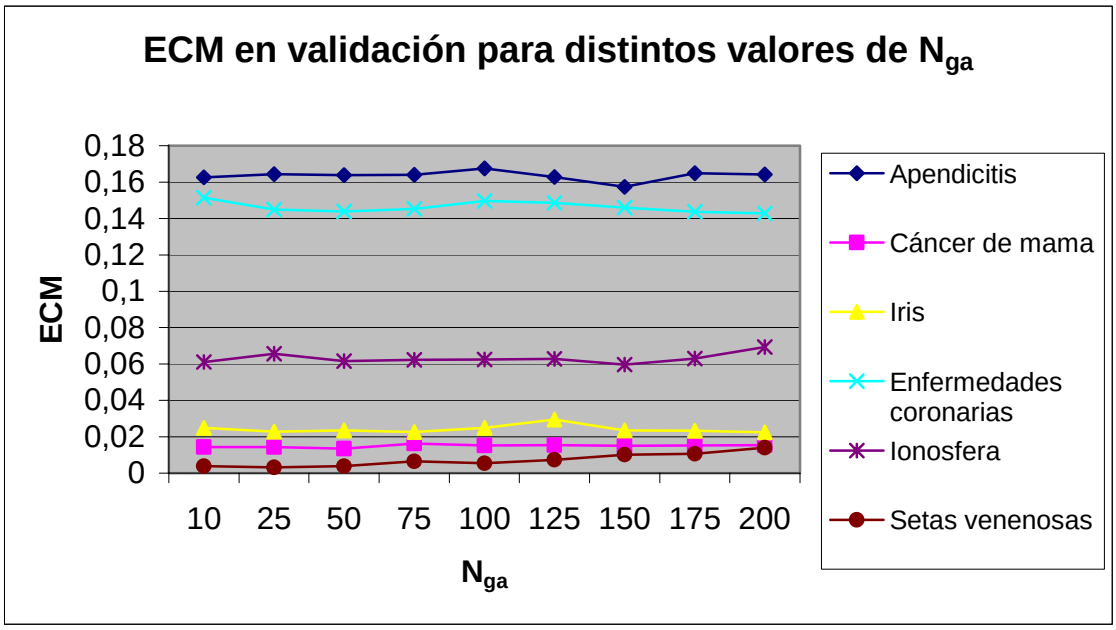


Figura 6.23. *Comparativa de los errores con diferentes valores de N_{ga}*

Como puede apreciarse, esta vez el rango en el cual el sistema se mantiene estable parece ser más amplio. Valores de N_{ga} superiores a 150 parecen ofrecer resultados ligeramente peores en algún problema (setas venenosas, ionosfera). En general, las diferencias en resultados son poco significativas, y el sistema se mantiene robusto en el rango de valores [10-150]. El valor adoptado es el de N_{ga}=50. Nuevamente, al igual que ocurría con los otros dos parámetros, se estudia aquí el número de generaciones ejecutadas por el algoritmo de optimización. La figura 6.24 muestra una gráfica en la que se ven, en relación a distintos valores de este parámetro.

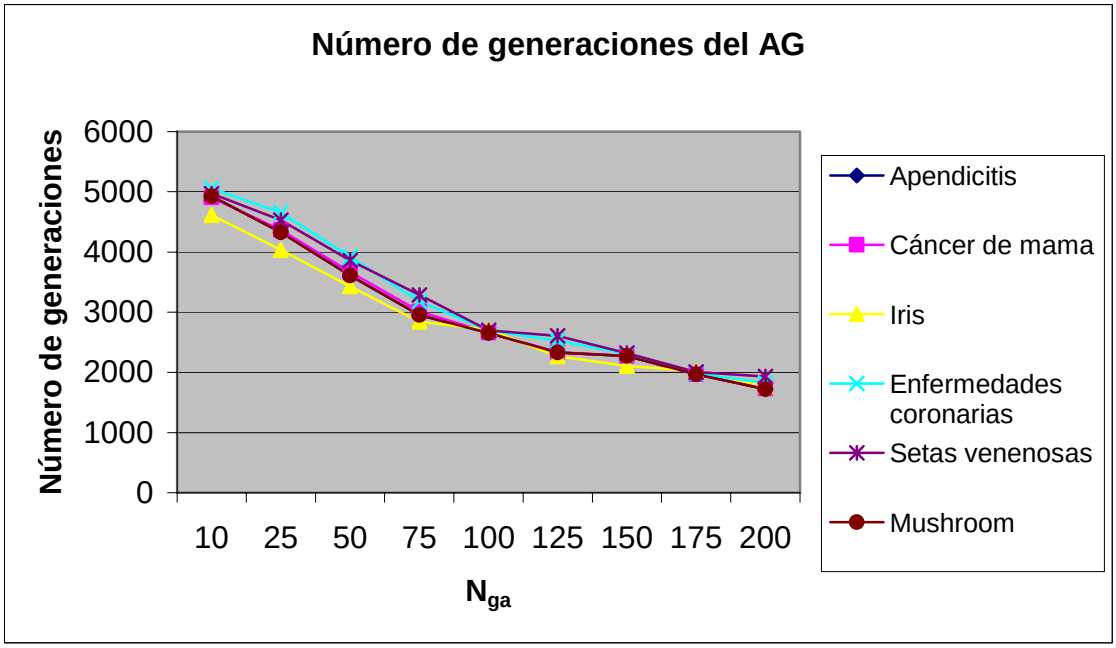


Figura 6.24. *Número de generaciones del AG con diferentes valores de* N_{ga}

Al igual que ocurría anteriormente, el decrecimiento del número de generaciones ejecutadas no es proporcional al crecimiento de este parámetro. Sin embargo, de esto no puede deducirse un comportamiento particular del sistema global, puesto que, si se aumenta el tamaño de la población del AG, se realiza una búsqueda más amplia en el espacio de estados, con lo que son necesarias menos generaciones para alcanzar un mínimo que produzca las 5 generaciones sin cambio.

6.3.2 Uso de grafos

En esta sección se hallan los valores de los parámetros cuando se emplea el sistema utilizando grafos como forma de codificación.

Al igual que se ha realizado anteriormente con árboles, en primer lugar, es necesario fijar los valores de los parámetros del sistema en sí, sin usar la optimización de constantes, para una vez fijados, poder utilizar estos valores para ejecutar el sistema usando dicha optimización.

6.3.2.1 Sin optimización de los pesos

6.3.2.1.1 Altura máxima y número máximo de entradas

Como se ha realizado anteriormente, los primeros experimentos llevados a cabo son aquellos en los que se intenta determinar cuál es la complejidad del sistema idónea para resolver problemas de dificultad arbitraria. Como ya se ha descrito, esta

complejidad se mide según el efecto de dos parámetros: altura máxima de los árboles y número máximo de entradas de las neuronas.

Los valores tomados por estos parámetros son los mismos que en el caso de usar árboles: una altura mínima de 4 y máxima de 6, y un número máximo de entradas a cada neurona mínimo de 3 y máximo de 12, para permitir que el sistema pueda generar soluciones a problemas de distinta complejidad.

La figura 6.25 muestra los resultados obtenidos en la validación para la resolución de los problemas anteriormente descritos. En todos estos resultados se ha mantenido un tamaño de población constante de 1000 individuos y una penalización al número de neuronas de 0, es decir, sin optimización.

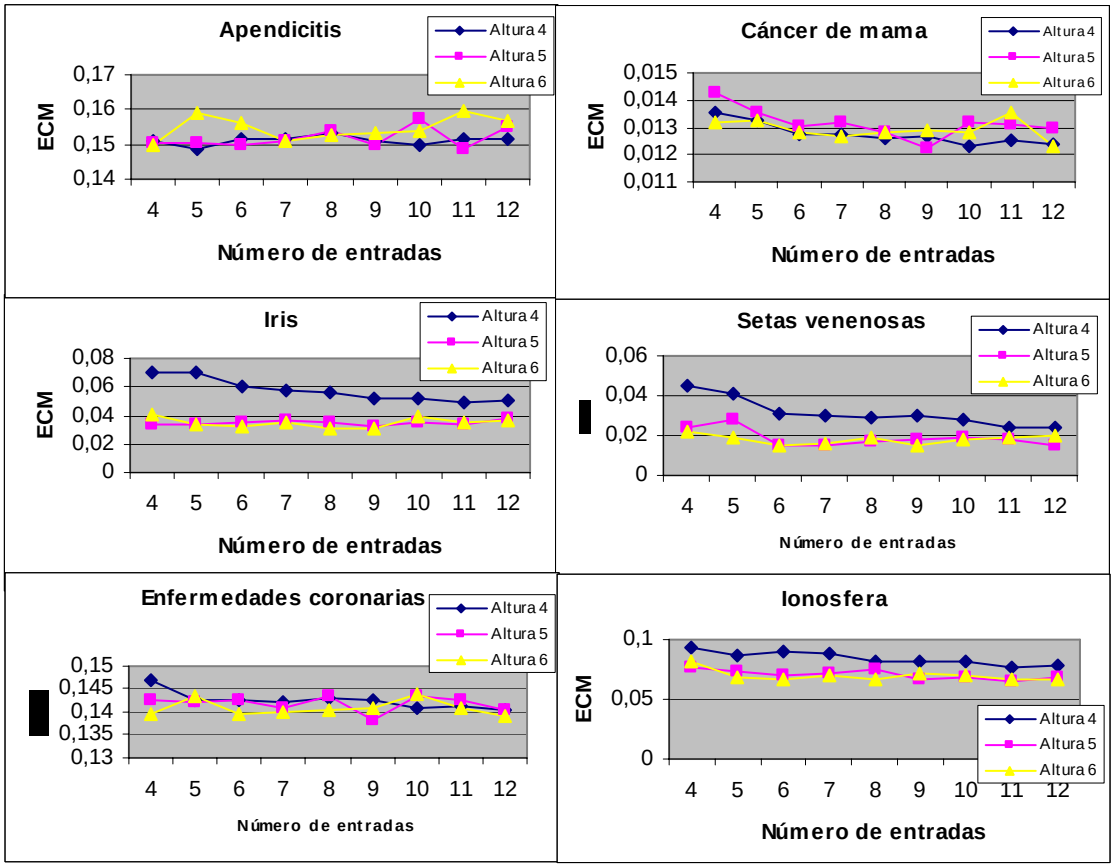


Figura 6.25. Resultados (ECM) comparando la altura máxima del árbol y el número máximo de entradas de cada neurona

Esta figura muestra, como también ocurría en el caso de usar árboles, la robustez del sistema, puesto que éste se mantiene estable, es decir, el error que devuelve no presenta grandes cambios, cuando estos parámetros adoptan valores en determinados rangos. Los resultados obtenidos con una altura de 4 han sido peores en determinados

problemas (iris, setas venenosas), mientras que con altura 5 ó 6 se han mantenido sin grandes variaciones. De igual forma, los resultados también se mantienen estables si se toma un número máximo de entradas en el rango [6-12]. Los valores de los parámetros adoptados para la realización del resto de experimentos están comprendidos en ambos rangos. Concretamente, se toma una altura de 5 y un número de entradas de 9 por minimizar el coste computacional de las simulaciones.

6.3.2.1.2 Penalización al número de neuronas

El siguiente parámetro que afecta a la complejidad de las redes es la penalización al número de neuronas. Como se ha dicho antes, el valor de este parámetro se suma al valor de la función de ajuste multiplicado por el número de neuronas que tiene la RNA. De esta forma, se intenta obtener redes sencillas, aunque no impone ninguna restricción propiamente dicha. Los valores que se han probado para este parámetro son 0.1, 0.001, 0.0001, 0.00001 y 0. El valor de 0 es el que se ha tomado para los experimentos realizados hasta el momento.

En las figuras 6.26 a 6.31 se puede ver la evolución seguida por el sistema en los distintos problemas con los distintos valores de penalización, con los parámetros de altura y número máximo de entradas a cada neurona los escogidos en la sección anterior. En estas gráficas se muestran los valores del ECM (sin sumar la penalización) de la validación correspondiente al mejor individuo encontrado por el algoritmo. En el eje X se muestra el esfuerzo computacional a que se refiere cada valor, medido como el número de ejecuciones de la función de ajuste a que se refiere, hasta un máximo de 500000 ejecuciones. En estas gráficas puede observarse claramente que, en general, el error obtenido es menor cuanto menor es la penalización, con la salvedad del valor de 0, que en muchas ocasiones aumenta el valor de error en la validación.

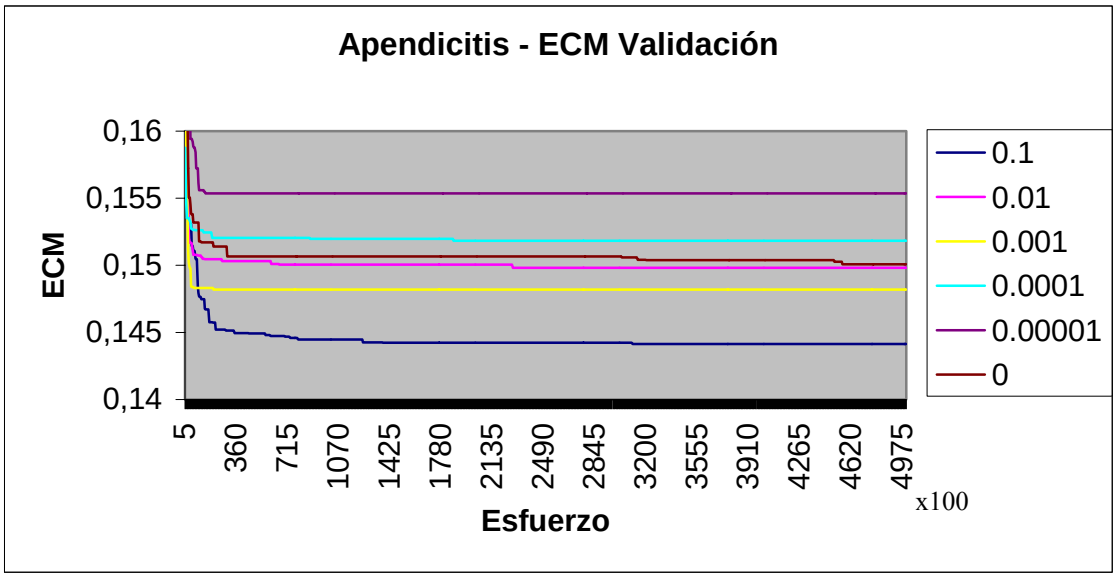


Figura 6.26. *ECM para distintos valores de penalización en el problema de apendicitis*

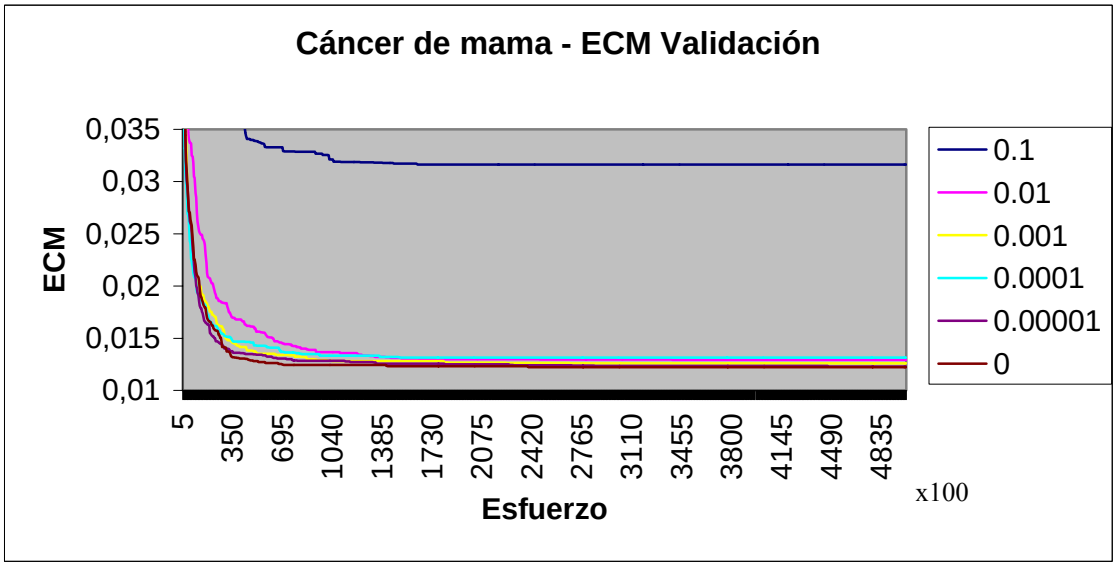


Figura 6.27. *ECM para distintos valores de penalización en el problema de cáncer de mama*

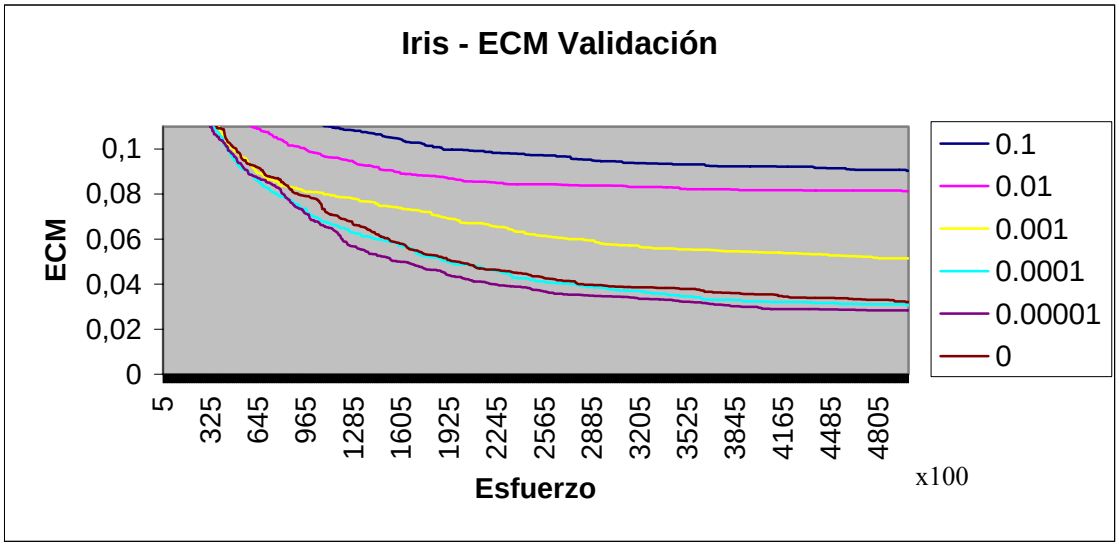


Figura 6.28. *ECM para distintos valores de penalización en el problema de flores iris*

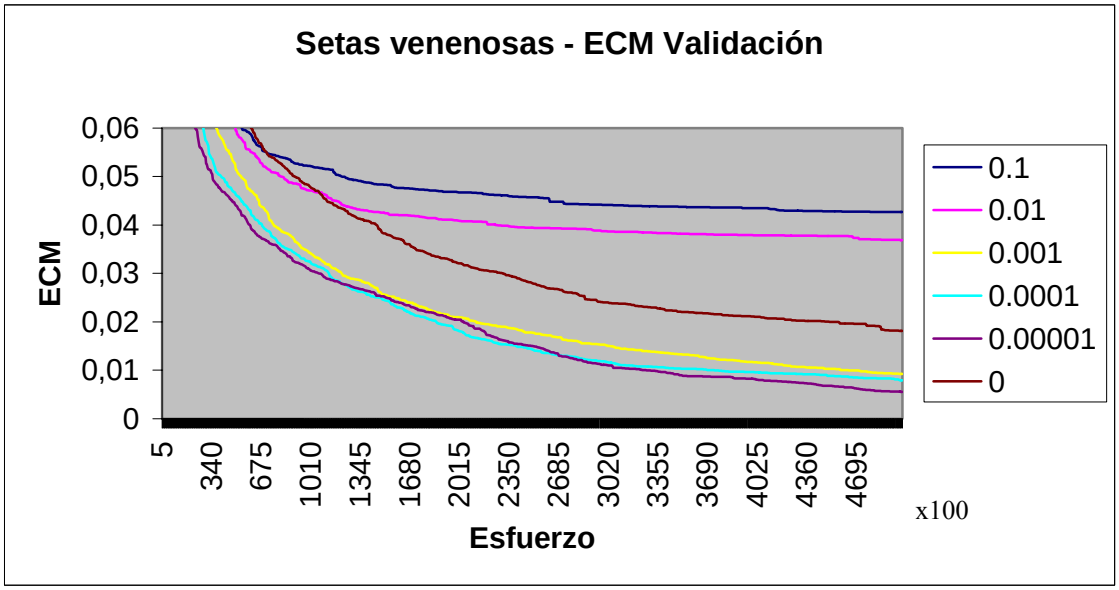


Figura 6.29. *ECM para distintos valores de penalización en el problema de setas venenosas*

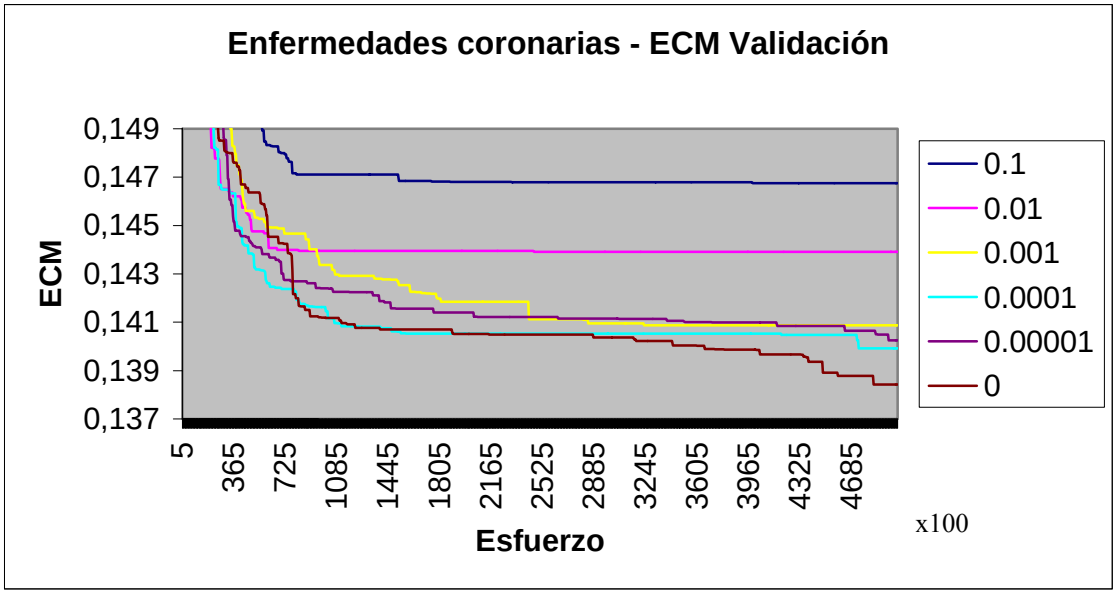


Figura 6.30. *ECM para distintos valores de penalización en el problema de enfermedades coronarias*

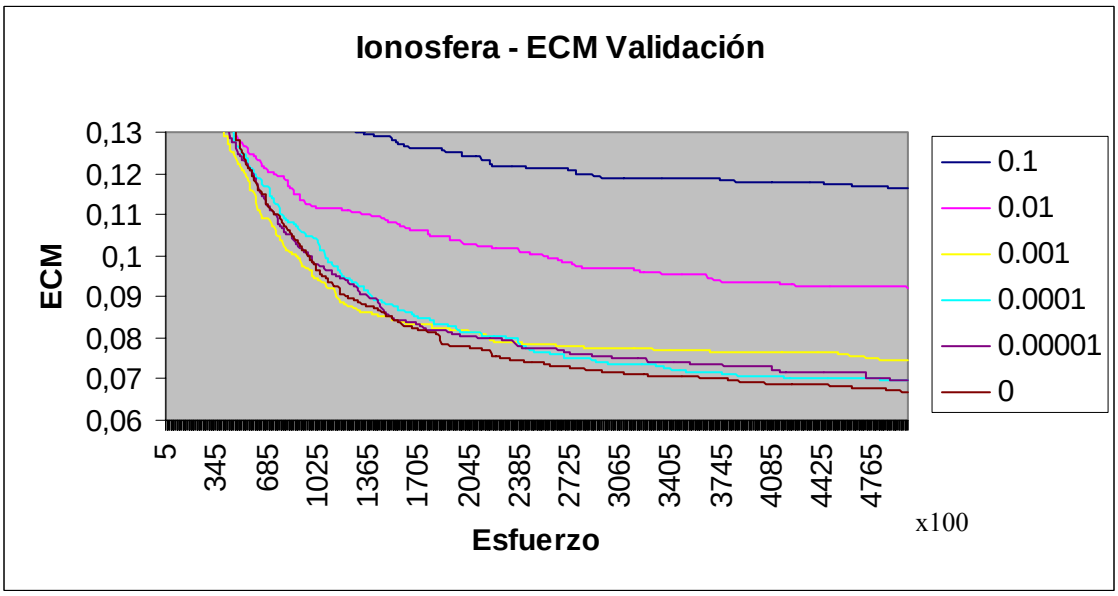


Figura 6.31. *ECM para distintos valores de penalización en el problema de la ionosfera*

Estas figuras muestran los errores cometidos por el sistema para cada problema, en base al esfuerzo computacional realizado. Estas gráficas son parecidas a las mostradas cuando se trabajaba con árboles, pero esta vez con grafos. Algunos problemas, como los de setas venenosas e ionosfera, son más complejos, por lo que, tras la ejecución de 500.000 veces la función de ajuste, el sistema seguía mejorando. En cambio, en otros problemas, como el de cáncer de mama o el de apendicitis, mucho antes de llegar ese punto ya se había encontrado la mejor solución al problema. En estos casos, el resto del proceso produciría un efecto de sobreentrenamiento, lo que se evita, como ya se ha

explicado, devolviendo como mejor individuo aquel que haya producido mejor resultado en el conjunto de validación.

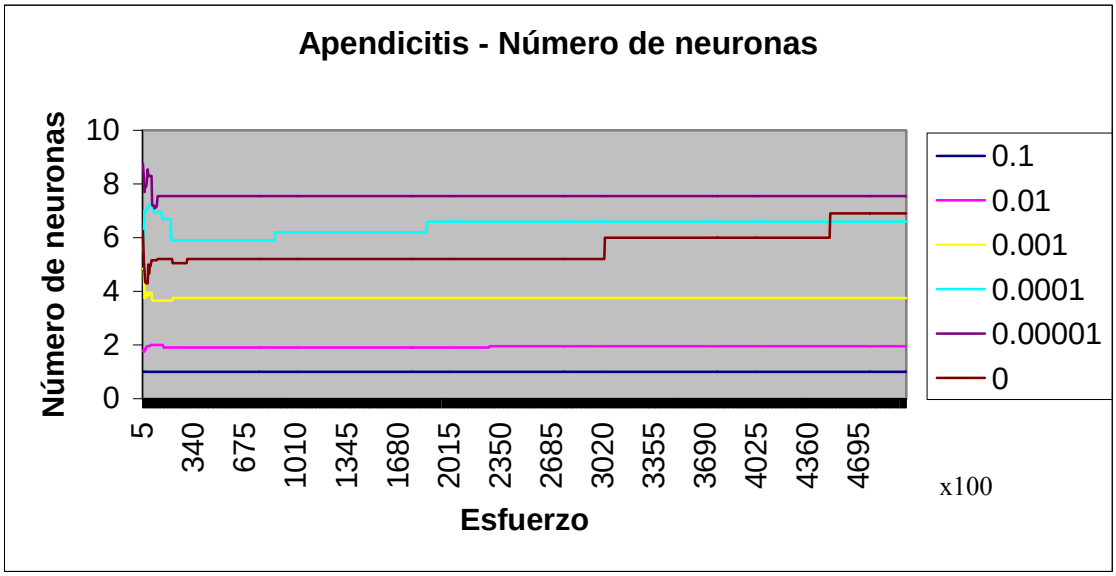


Figura 6.32. *Número medio de neuronas para distintos valores de penalización en el problema de apendicitis*

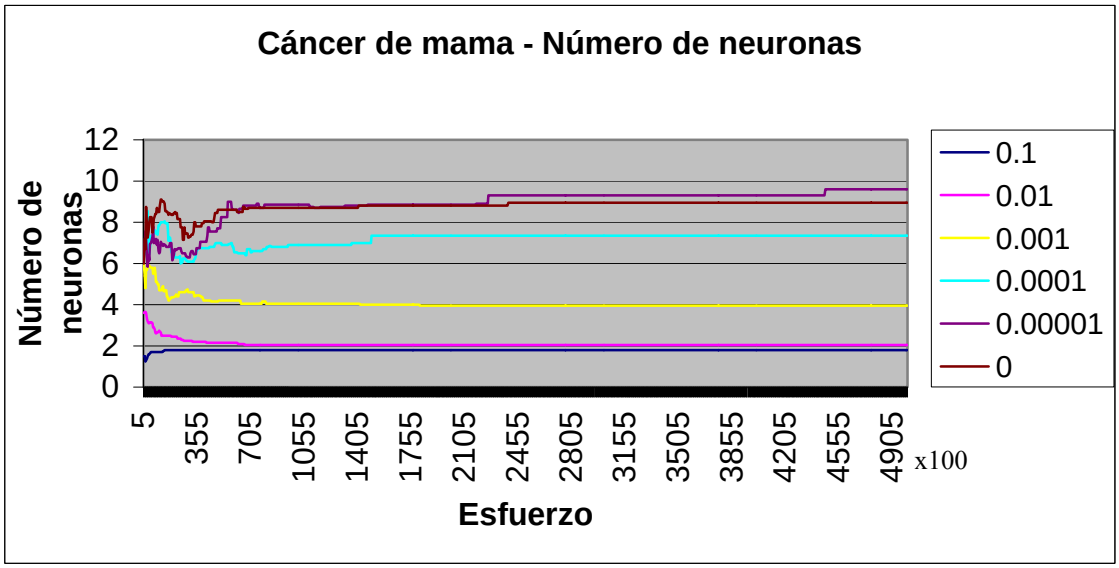


Figura 6.33. *Número medio de neuronas para distintos valores de penalización en el problema de cáncer de mama*

El número de neuronas que tenían estas redes que daban los mejores resultados en validación puede verse en las figuras 6.32 a 6.37, que se corresponden con los errores mostrados en las figuras 6.26 a 6.31. Como se puede ver, a medida que aumenta la penalización, el número de neuronas de las redes generadas disminuye. Con un valor muy alto de este parámetro (0.1), el sistema sólo genera redes con un número muy bajo

de neuronas, seguramente sólo con neuronas de salida. Por la contra, con un valor demasiado baja, como se puede ver, el número de neuronas es muy alto, pero el error obtenido es menor.

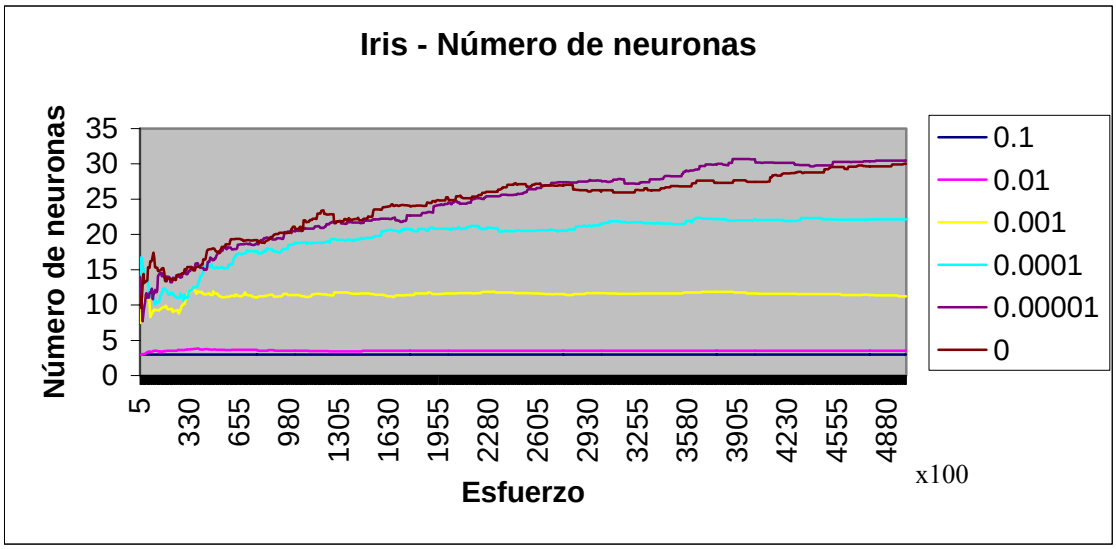


Figura 6.34. *Número medio de neuronas para distintos valores de penalización en el problema de iris*

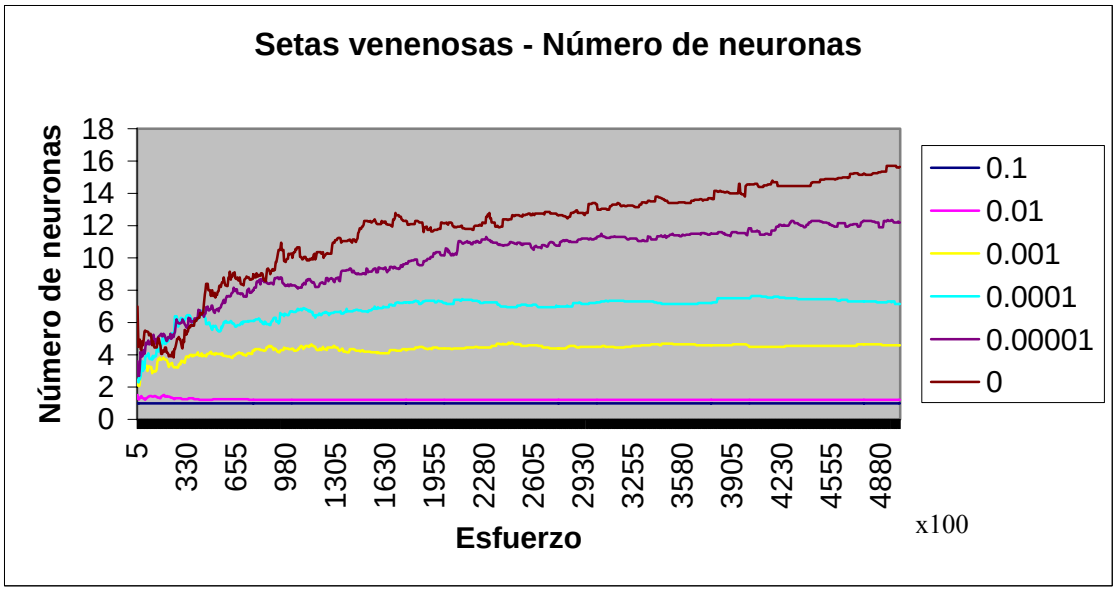


Figura 6.35. *Número medio de neuronas para distintos valores de penalización en el problema de setas venenosas*

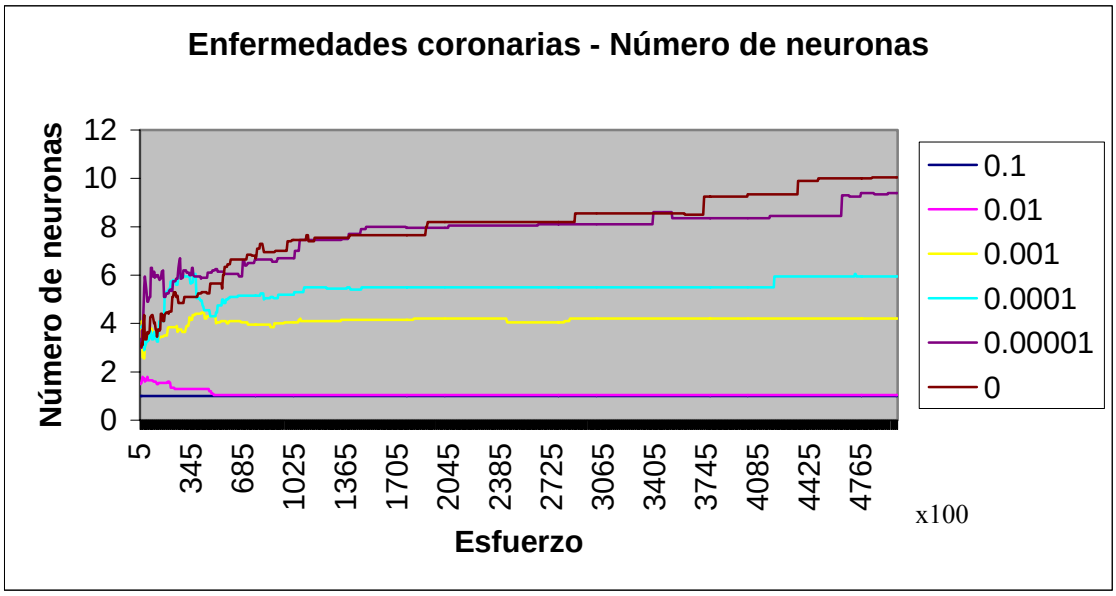


Figura 6.36. *Número medio de neuronas para distintos valores de penalización en el problema de enfermedades coronarias*

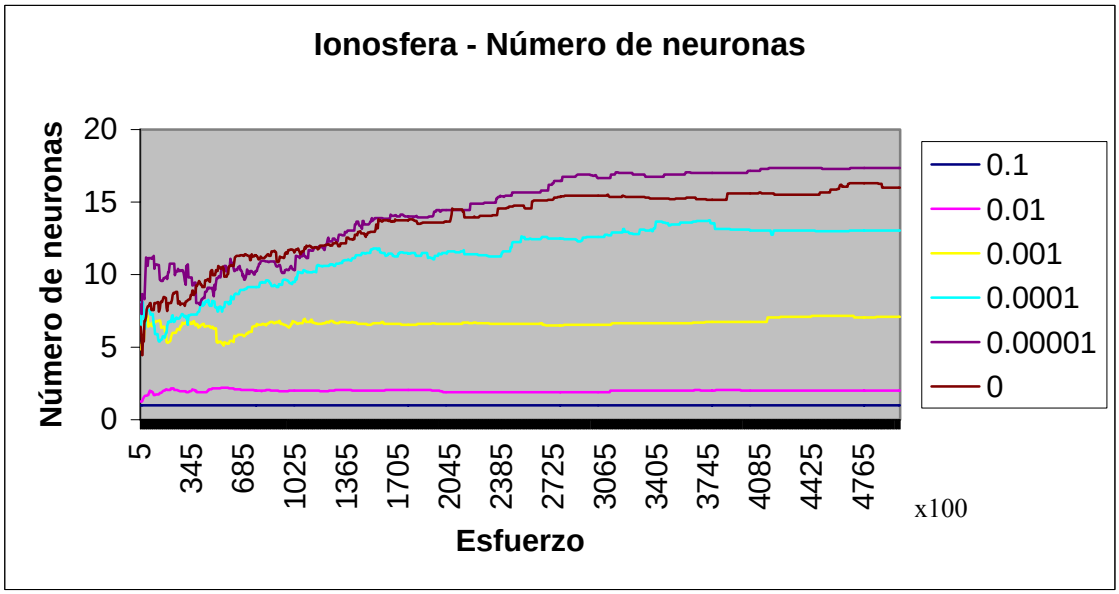


Figura 6.37. *Número medio de neuronas para distintos valores de penalización en el problema de ionosfera*

Un resumen de estas gráficas puede verse en la tabla 6.4, en la que se muestra el número medio de neuronas y conexiones obtenido para cada valor de penalización en cada problema, junto con las medias de los errores cuadráticos medios obtenidos en el entrenamiento y en la validación tras la ejecución de 500.000 veces la función de ajuste, en los 20 experimentos distintos realizados. Como ocurría en el caso de árboles, el número de neuronas que se muestra se refiere a neuronas ocultas y de salida, sin contar neuronas de entrada, puesto en ellas no se realiza ningún procesamiento.

		0.1	0.01	0.001	0.0001	0.00001	0
Apendicitis	Neuronas	1	1.9	3.75	6.6	7.55	6.9
	Conexiones	4.4	7.1	13.75	26.85	31.6	26.1
	Entrenamiento	0.077146	0.051612	0.057949	0.039271	0.057945	0.026931
	Validación	**0.144143**	0.150055	0.148183	0.151842	0.155353	0.150063
Cáncer de mama	Neuronas	1.8	2.05	3.95	7.35	9.6	8.95
	Conexiones	7.85	8.5	16.95	31.95	40.45	37.15
	Entrenamiento	0.070606	0.021491	0.020090	0.019177	0.016791	0.018198
	Validación	0.031632	0.012937	0.012615	0.013158	0.012284	**0.012229**
Iris	Neuronas	3	3.55	11.25	22.15	30.4	30
	Conexiones	8.05	8.9	32.65	56.95	83.56	81.35
	Entrenamiento	0.082532	0.074874	0.045774	0.028300	0.028010	0.030666
	Validación	0.090460	0.081330	0.051547	0.030554	**0.028395**	0.032143
Setas venenosas	Neuronas	1	1.2	4.6	7.15	12.2	15.65
	Conexiones	5.3	6.15	22.4	37.25	58.9	70
	Entrenamiento	0.045246	0.039326	0.009683	0.008543	0.005317	0.019229
	Validación	0.042679	0.036900	0.009265	0.007947	**0.005520**	0.018164
Enfermedades coronarias	Neuronas	1	1.05	4.2	5.95	9.4	10.05
	Conexiones	4.5	5.05	19.2	27.95	42.05	45.6
	Entrenamiento	0.112354	0.111496	0.092973	0.085861	0.083235	0.084436
	Validación	0.146748	0.143918	0.140872	0.139919	0.140257	**0.138428**
Ionosfera	Neuronas	1	2	7.1	13.05	17.35	16
	Conexiones	4.45	9.7	38.45	62.3	80.9	75.7
	Entrenamiento	0.119074	0.083277	0.053061	0.050352	0.045774	0.044971
	Validación	0.116165	0.092259	0.074564	0.069869	0.069926	**0.067003**

Tabla 6.4. *Número medio de neuronas, conexiones y errores de entrenamiento y validación para cada valor de penalización y cada problema*

Como ya ocurría con los árboles, sólo uno de los problemas ha ofrecido buenos resultados con un valor alto de penalización (apendicitis). En el resto de los problemas, los mejores resultados han sido obtenidos con valores bajos de este parámetro, llegando un punto a partir del cual las diferencias en errores no son muy significativas. En este caso, este punto es un valor de penalización de 0.0001. Valores menores de este parámetro no suponen una diferencia significativa en el error cometido. Por tanto, el

intervalo recomendado es [0-0.0001]. El valor adoptado para efectuar el resto de experimentos es de 0.00001.

6.3.2.2 Optimizando los pesos con AA.GG.

Al igual que se ha hecho con árboles, una vez establecidos los parámetros para el sistema en sí, se fijarán los valores a utilizar por el sistema de optimización de constantes, con el objetivo de determinar qué valores de los parámetros son los idóneos para resolver problemas de diferente complejidad cuando se utilizan los grafos.

6.3.2.2.1 Rangos de los pesos

Nuevamente el primer paso aquí es establecer qué rangos de valores pueden tomar los pesos de las conexiones. Por la misma razón que la expuesta en el caso del uso de árboles como forma de codificación, al utilizar el sistema de optimización de constantes los pesos sólo podrán tomar valores en el rango que se especifique, lo que no ocurre si no se utiliza este sistema.

	Apendicitis	Cáncer de mama	Iris	Setas venenosas	Enfermedades coronarias	Ionosfera
5	0,147505	**0,013499**	0,065054	0,039836	**0,138341**	0,091006
10	0,15508	0,013827	0,047513	0,028744	0,14754	**0,077371**
15	0,149876	0,013614	0,04274	0,026761	0,154589	0,082256
20	0,147772	0,015243	0,037559	0,025087	0,155793	0,084094
25	0,145471	0,015184	0,038607	0,024165	0,158296	0,080253
30	0,154369	0,016468	0,037707	0,022405	0,16057	0,083203
35	0,143354	0,017194	0,039472	0,023207	0,165263	0,078619
40	0,154306	0,016152	0,043308	0,022213	0,167452	0,082576
45	**0,142486**	0,017167	0,038679	0,022874	0,162944	0,080606
50	0,15057	0,017599	0,038765	0,0238	0,164327	0,081183
100	0,154717	0,018451	0,045951	0,024396	0,164888	0,082035
150	0,159536	0,015851	0,038944	0,019721	0,165644	0,090401
200	0,152579	0,017206	**0,033251**	0,021951	0,167002	0,081172
250	0,154834	0,019329	0,044891	0,020222	0,165778	0,096799
300	0,161485	0,016958	0,044026	0,019367	0,168008	0,090931
350	0,161369	0,019161	0,047771	0,020509	0,165751	0,080224
400	0,156389	0,017888	0,044429	0,020091	0,17437	0,083973
450	0,157188	0,018788	0,048328	**0,018975**	0,173423	0,085906
500	0,160091	0,017703	0,045592	0,021385	0,166227	0,083376

Tabla 6.5. *Errores en validación cometidos para diferentes valores de pesos*

Por lo tanto, los primeros experimentos están orientados a establecer el rango en el que estarán los pesos de las conexiones. En estos experimentos se mantiene el resto de los parámetros con un valor constante de N_{int}=5, F_{sel}=25, N_{ga}=50 y N_{gagen}=50, es decir, que se aplica el método cada 5 generaciones del algoritmo de PG, sobre el 25% de la población (osea, sobre 250 individuos), y con un tamaño de población del AG de 50

individuos y una ejecución de un máximo de 50 generaciones del mismo. Sin embargo, como ya se ha explicado, este último parámetro apenas tiene influencia en la ejecución del sistema.

En la tabla 6.5 puede verse diferentes valores de ECM cometidos tomando distintos valores para el rango de los pesos. Los valores límite para el rango tomados son los mismos que cuando se usaban árboles.

Otra vez, el sistema se mantiene robusto con respecto al valor tomado por este parámetro, puesto que las diferencias en resultados son muy pequeñas. Gráficamente, esto puede verse en la figura 6.38.

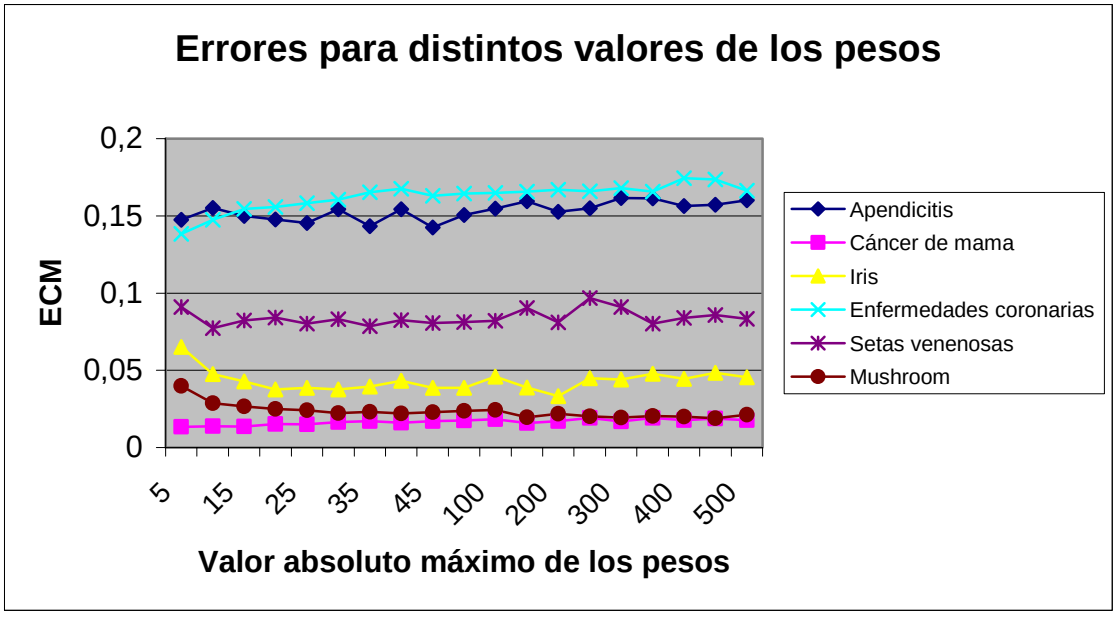


Figura 6.38. *Errores obtenidos para distintos valores de los rangos de los pesos*

Esta gráfica muestra unos valores de error muy similares para un rango muy amplio de valor absoluto máximo de los pesos. Un valor menor que 20 parece ofrecer resultados peores en ciertos problemas (ionosfera, iris y setas venenosas), mientras que uno superior a 200 también provoca lo mismo en otros problemas (iris y enfermedades coronarias). Por lo tanto, el intervalo que se recomienda es de [20-200]. Para efectuar el resto de los experimentos, el valor adoptado es de 25, es decir, los pesos se tomarán en un intervalo de [-25, 25].

6.3.2.2.2 Número de individuos a optimizar y frecuencia de la optimización

En esta sección se estudian los parámetros que dictan cuántas veces se aplicará el algoritmo de optimización de pesos. Como ya se ha explicado en el caso de usar

árboles, es este algoritmo el que realiza modificaciones en los valores de los pesos, dado que ahora ya no existen subárboles aritméticos y, evolutivamente, el único cambio en los pesos consiste en el intercambio con otros pesos de otro individuo. Por esta razón se estudia de manera conjunta el efecto de los dos parámetros que rigen este proceso, N_{int} y F_{sel}, que controlan cada cuánto se lanza este proceso, y sobre cuántos individuos se realiza. Para la realización de estos experimentos se mantuvieron unos valores constantes para el resto de los parámetros no estudiados hasta el momento, osea, N_{ga}=50 y N_{gagen}=50.

Las figuras 6.39 a 6.44 muestran una comparativa, para cada problema, de los resultados obtenidos para distintos valores de ambos parámetros.

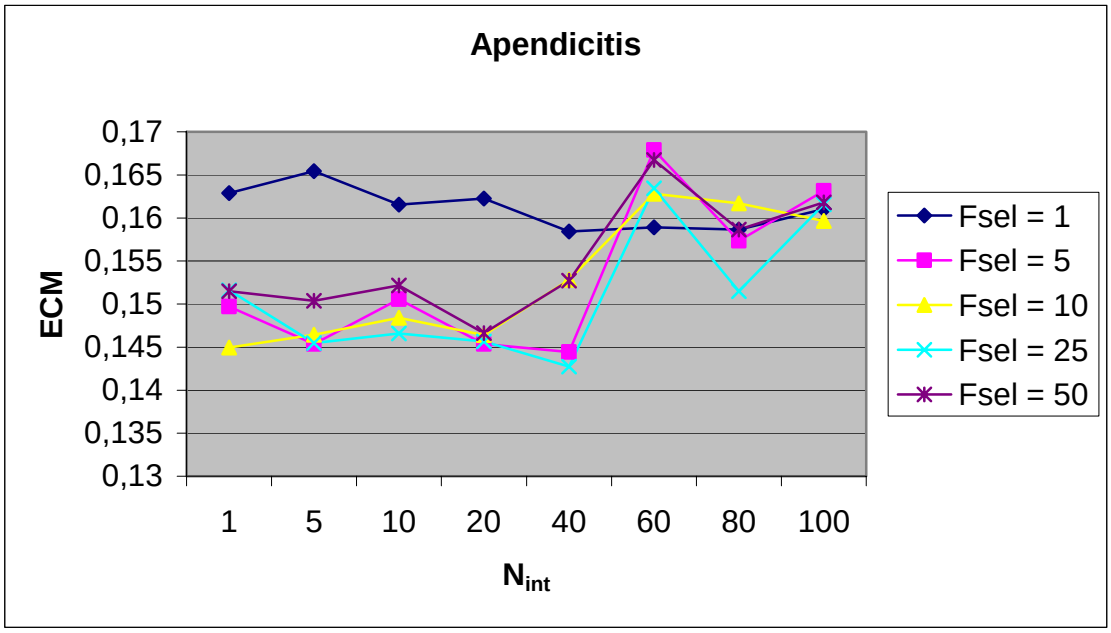


Figura 6.39. *Errores con diferentes valores de N_{int} y F_{sel} para el problema de apendicitis*

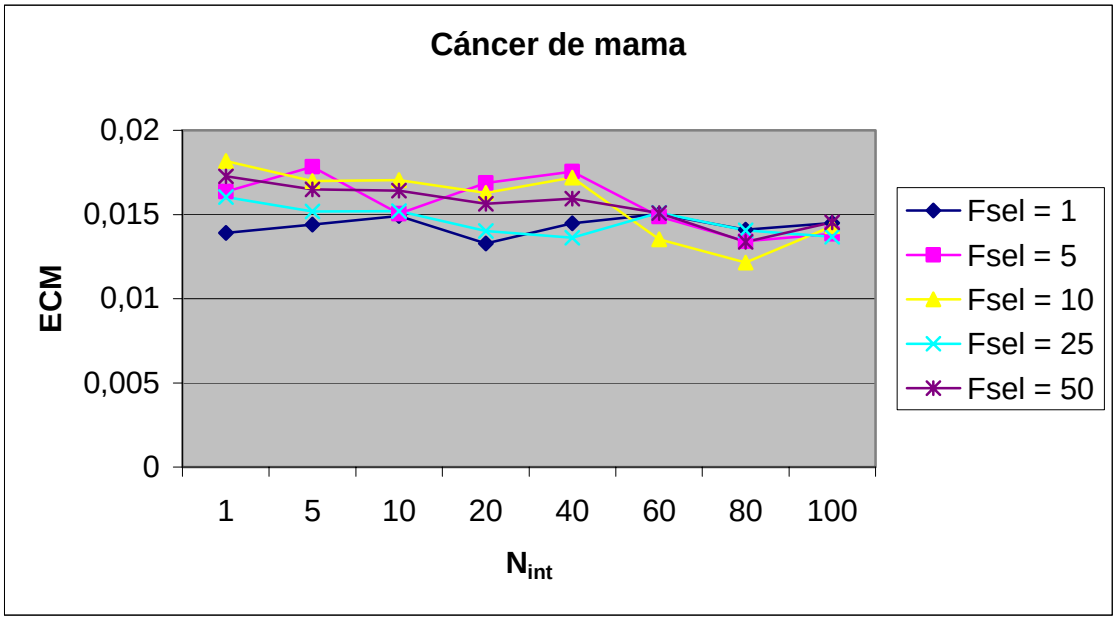


Figura 6.40. *Errores obtenidos con diferentes valores de N_{int} y F_{sel} para el problema de cáncer de mama*

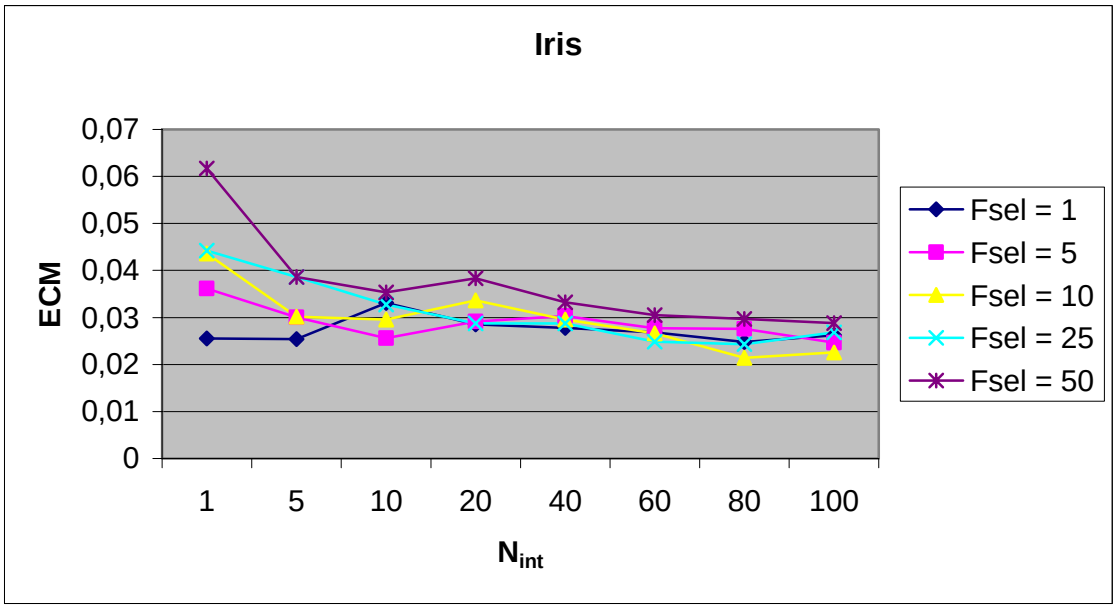


Figura 6.41. *Errores obtenidos con diferentes valores de N_{int} y F_{sel} para el problema de iris*

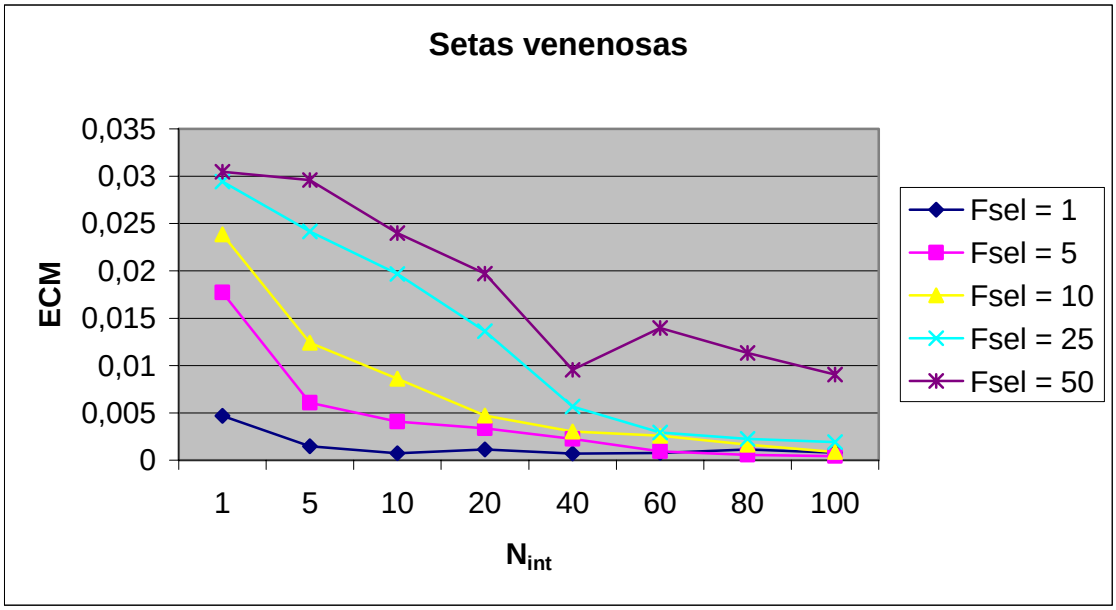


Figura 6.42. *Errores obtenidos con diferentes valores de N_{int} y F_{sel} para el problema de setas venenosas*

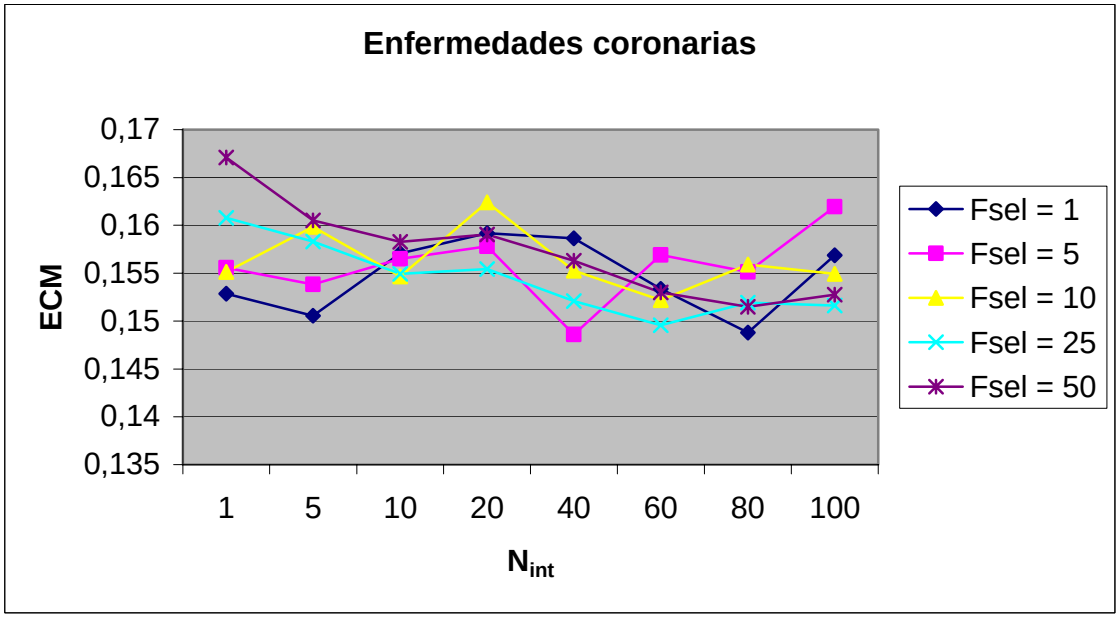


Figura 6.43. *Errores obtenidos con diferentes valores de N_{int} y F_{sel} para el problema de enfermedades coronarias*

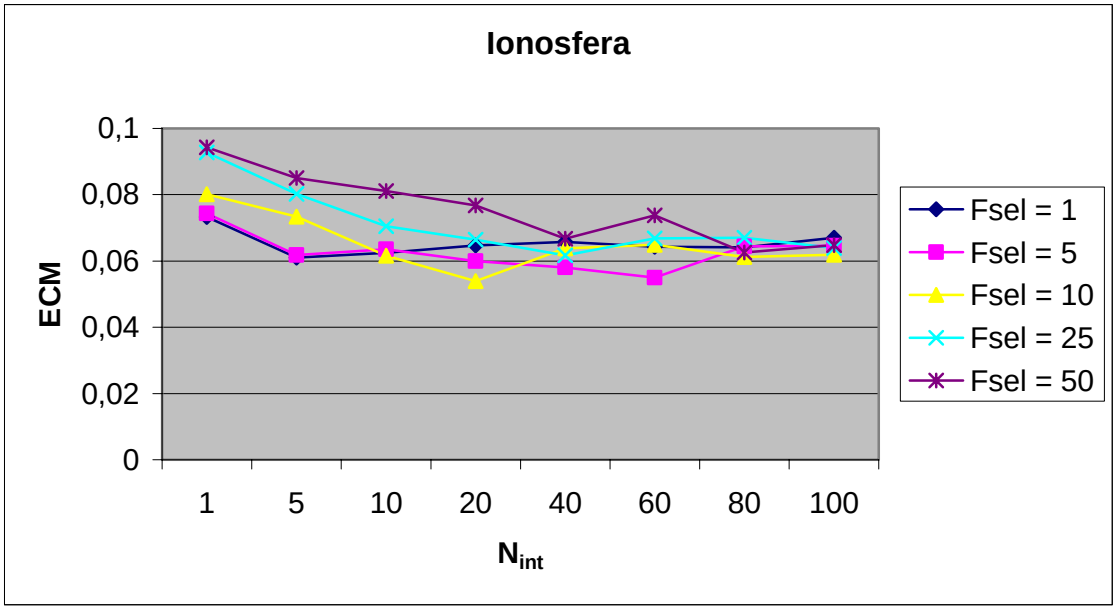


Figura 6.44. *Errores obtenidos con diferentes valores de N_{int} y F_{sel} para el problema de ionosfera*

Al igual que ocurría con árboles, las gráficas muestran claramente que valores muy altos del parámetro F_{sel} (25, 50) provocan la obtención de malos resultados en ciertos problemas (como el de setas venenosas). De igual forma, valores bajos del parámetro N_{int} (inferiores a 20) llevan a que el sistema se comporte peor en ciertos problemas (iris, setas venenosas e ionosfera). En general, si los parámetros se mantienen dentro de los rangos [1-10] para F_{sel} y [20-100] para N_{int}, el sistema se comporta de forma estable y el error cometido presenta pocas fluctutaciones. Para el resto de experimentos, se han tomado unos valores de N_{int}=80 y F_{sel}=1, que han ofrecido buenos resultados con un bajo coste computacional.

Como se ha realizado anteriormente, se plantea el problema de la pérdida de eficiencia del sistema cuando se varían estos parámetros. La figura 6.45 muestra el número de generaciones del AG ejecutadas al alcanzar las 500.000 evaluaciones del fichero de entrenamiento, para distintos valores del parámetros N_{int}, y manteniendo los valores constantes de F_{sel}=1, N_{ga}=50 y N_{gagen}=50. Como se puede ver, a medida que se reduce el valor de N_{int}, es decir, a medida que aumenta la frecuencia en la que se ejecuta el algoritmo de optimización, lógicamente aumenta el número de generaciones ejecutadas del AG. Sin embargo, tal como ocurría en el caso de usar árboles, este aumento no es proporcional y sucede el mismo efecto que sucedía entonces, que el proceso global de búsqueda se centra más en el desarrollo de la topología o en la búsqueda de buenos valores para los pesos de las conexiones existentes.

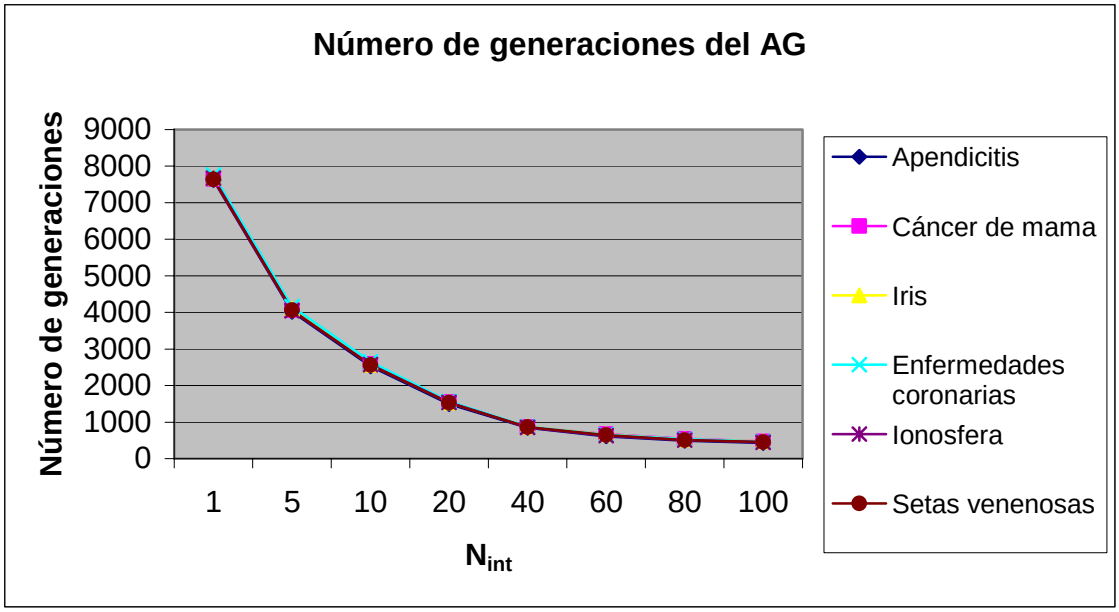


Figura 6.45. *Número de generaciones del AG ejecutadas para distintos valores de* N_{int}

El otro parámetro que afecta al número de veces que se ejecuta este proceso, estudiado en esta sección, es el parámetro F_{sel}, que rige a cuántos individuos se aplica el proceso de optimización. En la figura 6.46 se estudia el efecto de este parámetro, tomando diferentes valores, con valores constantes para el resto de parámetros de N_{int}=80, N_{ga}=50 y N_{gagen}=50. Como puede verse, el crecimiento del número de generaciones a medida que crece este parámetro no es proporcional, sino que sigue la curva de una exponencial inversa. El crecimiento de la curva se va atenuando a medida que aumenta el valor de F_{sel}, lo que quiere decir que no se ejecutan tantas generaciones del AG como cabría suponerse. En la ejecución del AG se alcanzan antes unos valores de los pesos que no se pueden seguir mejorando en este proceso de optimización, con lo que cesa este proceso de optimización. Esto es consecuencia de aplicar el proceso de optimización a un número mayor de individuos, con lo que la siguiente vez que se aplique, habrá más posibilidades de que este proceso se aplique sobre individuos que ya están optimizados, al menos en parte. De igual manera a lo expuesto anteriormente, si este parámetro toma valores altos, la búsqueda se centra más en buscar valores para los pesos de las conexiones. Por contra, si toma valores bajos, la búsqueda se centra más en el diseño de la arquitectura de la red.

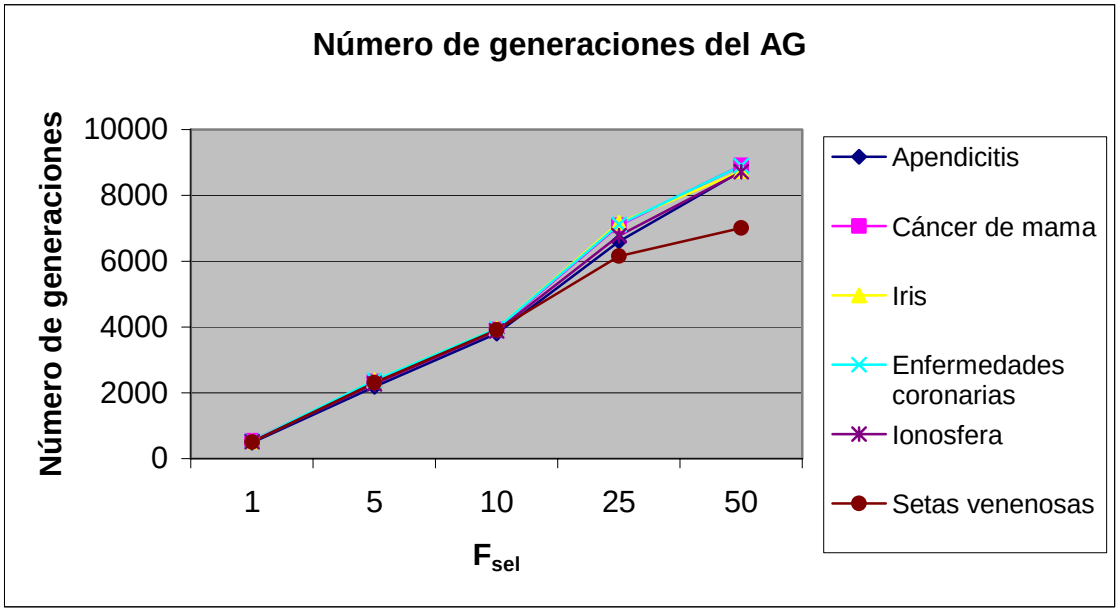


Figura 6.46. *Número de generaciones del AG ejecutadas para distintos valores de* F_{sel}

6.3.2.2.3 Tamaño de la población del algoritmo genético

El último parámetro que se va a estudiar respecto al proceso de optimización de constantes es el parámetro N_{ga}, que, como ya se ha visto, representa el tamaño de la población del algoritmo genético. Este parámetro, junto con el parámetro N_{gagen} dicta el esfuerzo que se realiza en la búsqueda de los valores de los pesos de las conexiones.

En la figura 6.47 puede verse una comparativa de los valores de error obtenidos para distintos valores de este parámetro, dejando el resto de parámetros con sus valores recomendados anteriormente N_{int}=80, F_{sel}=1 y N_{ga}=50.

Nuevamente, no existe una diferencia significativa entre distintos valores de este parámetro, las diferencias en el error tomadas con distintos valores de N_{ga} son muy bajas, con lo que el sistema se comporta de forma muy estable ante valores muy distintos de este parámetro. Para el problema de ionosfera, un valor muy alto (200) sí ofrece un resultado un poco peor, sin ser una diferencia muy significativa. En este caso, el intervalo recomendado para tomar los valores es [25-175]. Para efectuar el resto de experimentos y pruebas, se toma un valor de 150.

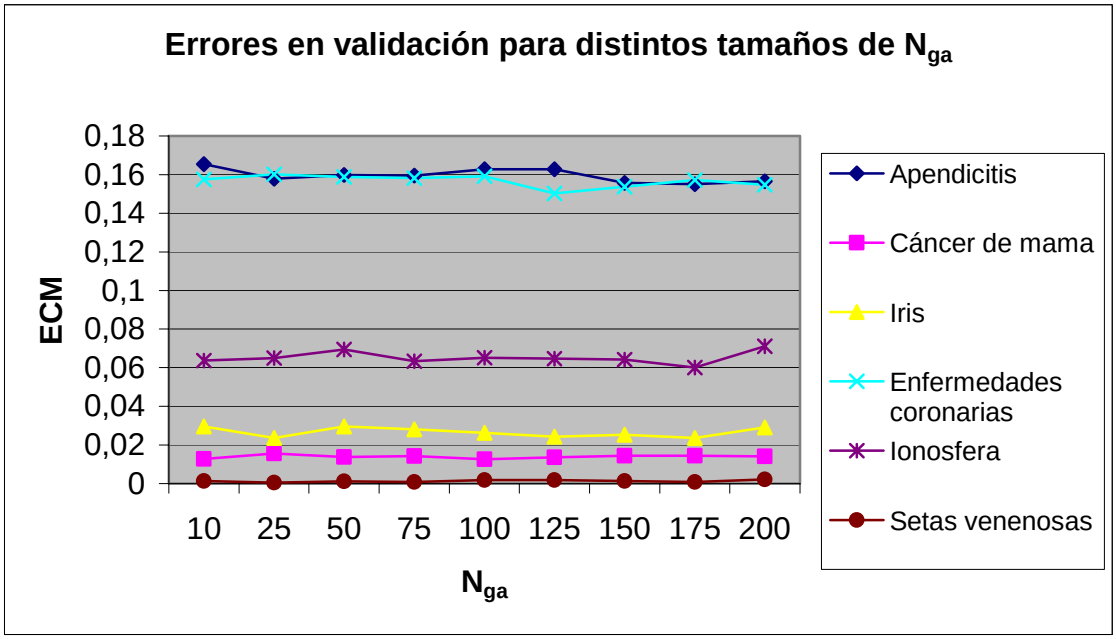


Figura 6.47. *Comparativa de los errores con diferentes valores de* N_{ga}

Cabe plantearse el por qué de tan poca variabilidad. Para ello, se muestra gráficamente en la figura 6.48 el número de generaciones del AG ejecutadas en total, al llegar a las 500.000 evaluaciones del fichero de entrenamiento, según distintos valores del parámetro N_{ga}.

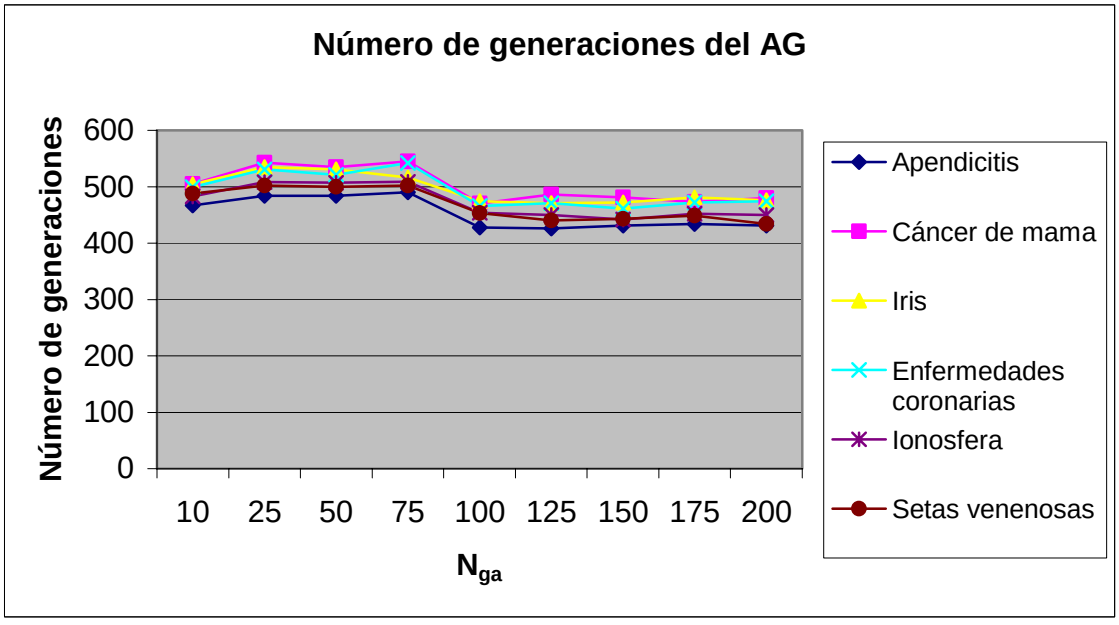


Figura 6.48. *Comparativa de los errores con diferentes valores de* N_{ga}

En esta figura puede apreciarse que, llegado a un punto, no existen diferencias importantes en el número de generaciones del AG ejecutadas tomando distintos tamaños de la población del mismo. Esto quiere decir que, a pesar de que se aumente el tamaño

de la población del AG, el número de generaciones que se ejecuta es el mismo, de donde se deduce que se invierte más esfuerzo en la búsqueda de los pesos de las conexiones. Sin embargo, al mirar las gráficas de error, estas no presentan ninguna diferencia significativa. Por lo tanto, este esfuerzo extra que se realiza en la búsqueda de los pesos se extrae del proceso global de diseño de la arquitectura. Nuevamente, el ajuste de este parámetro sirve para determinar si se desea que el sistema centre la búsqueda en el diseño de la arquitectura o en el entrenamiento de los pesos, y no para obtener diferencias significativas en los errores obtenidos.

6.4 Discusión

Según los resultados expuestos en la sección 6.3, el sistema ofrece un buen comportamiento y los resultados indican que los problemas han sido resueltos con un nivel de precisión muy alto. En esa sección se han analizado los distintos parámetros que afectan al comportamiento del sistema. En función de los resultados obtenidos en los problemas a resolver se han adoptado diferentes valores para estos parámetros con el objetivo de conseguir un conjunto de parámetros que sean válidos para problemas de diferente complejidad.

En primer lugar, se han analizado los parámetros que limitan la complejidad de las redes. Tras un estudio de los mismos, realizado sobre experimentos realizados tomando distintos valores, se concluye cuales son los mejores valores para generar redes con la suficiente complejidad como para resolver distintos problemas.

Respecto al valor de penalización, tras la ejecución de los experimentos se ha tomado un valor bajo (0.00001) como el adecuado para la resolución del problema. Este valor tan bajo se debe a que la complejidad de las redes ya ha sido limitada por los dos parámetros anteriores, altura máxima y número máximo de entradas, así que este parámetro no es tan determinante. En experimentos previos, tomando valores de estos dos parámetros más altos, el valor de la penalización sí resultó ser muy importante para hacer que el sistema genere redes sencillas, que eviten el problema de la penalización. Otra opción, por lo tanto, sería tomar valores de estos parámetros altos, para poder generar redes muy complejas y ajustar el parámetro de penalización para que el proceso de búsqueda genere redes sencillas. La tabla 6.6 muestra un resumen de los resultados obtenidos tomando una altura máxima de 6 y 12 entradas como máximo a cada neurona,

utilizando árboles como forma de codificación, y sin usar el sistema de optimización de constantes.

		0.1	0.01	0.001	0.0001	0.00001	0
Apendicitis	Neuronas	1	2	3.6	10.55	21.45	35.625
	Conexiones	4.45	6.75	15.1	47.4	94.25	155.416672
	Entrenamiento	0.061952	0.040163	0.027820	0.023821	0.019658	0.014382
	Validación	**0.189005**	0.217360	0.224560	0.214523	0.223411	0.215173
Cáncer Mama	Neuronas	1.75	2	2.65	9.3	22.65	27.833
	Conexiones	8.8	9.2	13.1	47.6	100.7	126.944443
	Entrenamiento	0.033715	0.019682	0.018014	0.014265	0.013660	0.012574
	Validación	0.033921	**0.020638**	0.020965	0.023816	0.025513	0.025146
Iris	Neuronas	3	4	8.45	24.2	38.9	42.85
	Conexiones	8.8	11.95	27.85	86.55	140.25	157.05
	Entrenamiento	0.060797	0.030212	0.017468	0.016583	0.016811	0.015728
	Validación	0.077240	0.052225	**0.037990**	0.040840	0.040717	0.040751
Setas venenosas	Neuronas	1	1.4	4.65	12.85	15.15	23.39
	Conexiones	5.45	7.75	26.15	74.65	85.6	140.78
	Entrenamiento	0.042936	0.033357	0.008244	0.002883	0.003407	0.002724
	Validación	0.040074	0.032868	0.007998	**0.003358**	0.003661	0.003360
Enfermedades coronarias	Neuronas	1	1.3	4.95	16.4	24.8	30.4
	Conexiones	7.25	8.7	28.45	88.1	118.95	167.5
	Entrenamiento	0.105899	0.101315	0.075176	0.068677	0.066005	0.062186
	Validación	**0.157284**	0.163359	0.176544	0.178164	0.174721	0.172926
Ionosfera	Neuronas	1	2.35	8.15	21.6	32.4	42.45
	Conexiones	6.1	11.95	48.3	128.15	197.4	261.8
	Entrenamiento	0.098364	0.062530	0.030976	0.021149	0.022647	0.017810
	Validación	0.109412	0.091278	**0.068541**	0.072695	0.073937	0.071236

Tabla 6.6. *Número medio de neuronas, conexiones y errores de entrenamiento y validación para cada valor de penalización y cada problema para valores altos de complejidad de las redes*

En esta tabla puede verse que los resultados son mejores en la mayoría de las ocasiones limitando primero la complejidad de las redes y ajustando luego este parámetro, como puede verse en comparación con las tablas 6.2 y 6.4, que utilizan árboles y grafos respectivamente. Solamente en uno de los problemas, en el de las setas venenosas, los resultados son mejores para esta prueba. Esto es debido a la alta complejidad de este problema. Sin embargo, la diferencia en el error es muy baja, del orden de 10^{-3}.

La razón fundamental por la que funciona mejor el sistema limitando primero la complejidad es porque, de esta forma, una vez limitada, se reduce el espacio de estados, con lo que la búsqueda se realiza de forma más eficaz y eficiente.

Como se ha comentado anteriormente, con un valor de penalización alta (0.1), el sistema genera redes que solamente tienen neuronas de salida. El número de neuronas va aumentando conforme disminuye el valor de penalización, con la consecuente

disminución del error en el entrenamiento. A medida que aumenta el número de neuronas, también aumenta el número de conexiones de la red. Estos dos valores (número de neuronas y de conexiones) son un indicativo de la complejidad de la red. Por lo tanto, a mayor complejidad de la red, se han conseguido errores menores en el entrenamiento.

En problemas más sencillos, como en el de predicción de cáncer de mama, estos valores que limitan la complejidad de la red (altura del árbol y número máximo de entradas de cada neurona) sí son suficientes para resolver satisfactoriamente el problema, con lo que el valor de penalización sí es útil para optimizar las redes resultantes y generar redes con pocas neuronas. En las figuras 6.3 y 6.27 se puede apreciar que, en el caso de predicción de cáncer de mama, el proceso ha terminado de hallar una solución mucho antes de la ejecución de las 500.000 veces la función de ajuste, con lo que la complejidad máxima de la red parece suficiente para resolver el problema y, a partir de ese punto, el sistema intenta optimizar la arquitectura de la red mediante el parámetro de penalización.

Los valores tomados para los parámetros han sido seleccionados tras un análisis de los resultados obtenidos en los 6 problemas descritos en este trabajo. Sin embargo, el problema de apendicitis parece acusar del efecto del sobreentrenamiento. Tras mirar las tablas 6.2 y 6.4 (y también la 6.6), puede apreciarse una gran diferencia entre los valores de error en el entrenamiento y en la validación. Esto es debido al bajo número de patrones de este problema, que hace que, una vez dividido este conjunto en entrenamiento y validación, los subconjuntos resultantes no son representativos del problema. Por esta razón el problema de apendicitis no ha sido tenido en cuenta tanto como el resto de los problemas para escoger los parámetros del sistema.

Comparando los resultados obtenidos en validación (tablas 6.2 y 6.4) utilizando árboles y grafos, puede verse que los mejores resultados se obtienen mediante el uso de grafos, excepto en los problemas de iris e ionosfera, a pesar de que los valores obtenidos en entrenamiento puedan mostrarse distintos. Esto es consecuencia de que el nivel de aprendizaje que alcanza una red realmente se mide con los valores de test y, en este caso, con el de validación, que arroja una estimación del valor del test. La sección 7 realiza distintos tests, comparando este método con otras técnicas, y hace un análisis más exhaustivo de la comparativa entre técnicas.

Respecto a los parámetros del sistema de optimización, después del análisis efectuado, puede concluirse que la elección de distintos valores de los parámetros N_{int}, F_{sel} y N_{ga} influye muy poco en la eficacia del sistema, es decir, en los resultados obtenidos. Influyen, no obstante, en el comportamiento del sistema. La elección de distintos valores, como ya se ha explicado, determinará que el sistema enfoque el proceso de búsqueda en el diseño de la red, o bien en la búsqueda de pesos adecuados con las redes que se generan en la población.

Sin embargo, sí es importante en este punto la elección del rango de los pesos puesto que, como se ha mostrado, los valores que se generen no podrán salir de ese rango, cosa que no pasaba cuando no se usaba el sistema de optimización.

Finalmente, la tabla 6.7 muestra un resumen de los intervalos en los cuales el sistema se mantiene robusto y no presenta grandes diferencias en los resultados que ofrece, para los distintos parámetros. Dicha tabla muestra, además, los valores escogidos en este trabajo.

		Árboles		Grafos	
		Intervalo recomendado	Valor escogido	Intervalo recomendado	Valor escogido
Parámetros de PG	Tamaño de la población	1000			
	Algoritmo de selección	Torneo de 2 individuos			
	Algoritmo de creación	*Ramped Half&Half*			
	Tasa de cruces	95%			
	Tasa de mutación	4%			
	Probabilidad de selección de función	95%			
	Rango de los pesos	[-x, x] x ϵ [15, 35]	[-20, 20]	[-x, x] x ϵ [20, 200]	[-25, 25]
Parámetros del sistema	Altura	[5, 6]	5	[5, 6]	5
	Número máximo de entradas	[6, 12]	6	[6, 12]	9
	Penalización	[0, 0.001]	0.00001	[0, 0.0001]	0.00001
Parámetros del sistema de optimización	N_{int}	[20, 100]	80	[20, 100]	80
	F_{sel}	[1, 10]	10	[1, 10]	1
	N_{ga}	[10, 150]	50	[25, 175]	150
	N_{gagen}	-	50	-	50

Tabla 6.7. *Resumen de los parámetros que se adoptan e intervalos recomendados*

CAPÍTULO

7 COMPARACIÓN CON OTROS MÉTODOS DE GENERACIÓN Y ENTRENAMIENTO DE RR.NN.AA. MEDIANTE CE

El sistema aquí presentado ha sido comparado con otros métodos de generación y entrenamiento de RR.NN.AA. para evaluar su funcionamiento. A partir de esta comparación de los resultados con los obtenidos con otras técnicas, se extraen una serie de ventajas que la herramienta descrita en este trabajo posee frente a las otras técnicas.

7.1 Método de comparación

A la hora de comparar algoritmos de clasificación, el método más utilizado es el de *cross validation* o validación cruzada [Stone 1978] para estimar la precisión de los algoritmos y utilizar *t-tests* para confirmar si los resultados son significativamente diferentes. En el método de la validación cruzada, el conjunto D de datos se divide en k conjuntos D_1, …, D_k que no se solapen (*k-fold cross validation*). En cada iteración i (que varía de 1 a k), el algoritmo se entrena con el conjunto $D \backslash D_i$ y se hace test en D_i. Sin embargo, algunos estudios han mostrado que la comparación de algoritmos utilizando estos *t-tests* en validación cruzada conlleva un ligero aumento de lo que se denomina error tipo I [Herrera 2004], es decir, detectar una diferencia cuando no existe.

Dietterich analizó el comportamiento del método *k-fold cross validation*, combinado con el empleo de un test t [Dietterich 1998]. En ese trabajo se propuso modificar el estadístico utilizado y se justifica que es más efectivo realizar k/2 ejecuciones de un test *2-fold cross validation*, con diferentes permutaciones de los datos, que realizar un test *k-fold cross validation*. Como solución de compromiso entre

la potencia del test y el tiempo de cálculo, propone realizar 5 ejecuciones de un test de validación cruzada con k=2, de ahí el nombre 5x2cv. En cada una de las 5 iteraciones, los datos se dividen aleatoriamente en dos mitades. Cada una de las mitades se toma como entrada del algoritmo y la otra se utiliza para hacer un test de la solución final, con lo que en total se tienen 10 tests distintos (5 iteraciones, 2 resultados por cada una) [Herrera 2004]. Este es el método que se utiliza en este trabajo para comparar los resultados con los obtenidos con otras técnicas. En todos los casos, la base de la comparación es la media de los 10 tests realizados por el método 5x2cv, que realiza una medida de la precisión obtenida en los tests.

Un problema adicional del método 5x2cv como base para comparar métodos es que este método exige particionar el conjunto de patrones en dos mitades. Esto puede no presentar ningún problema cuando se trabajan con conjuntos lo suficientemente amplios. Sin embargo, tanto aquí como en el trabajo en que se basa la comparativa se utilizan conjuntos de patrones bastante pequeños (ver tabla 6.1, problemas de cáncer de mama, iris, enfermedades coronarias e ionosfera). Para evitar el sobreentrenamiento, en este trabajo se propone reservar una parte del conjunto de entrenamiento para realizar una validación de las redes resultantes y tomar la que mejores resultados dé en ese subconjunto. En el caso del uso del método 5x2cv, esto conlleva particionar en dos el conjunto de entrenamiento, el cual a su vez es la mitad del conjunto inicial de patrones. Esta segunda partición provoca que, o bien el entrenamiento o bien la validación se produzca con un número muy reducido de patrones, que seguramente no será representativo del espacio de búsqueda que se está explorando. Como prueba de esto, se han realizado distintos experimentos para verificar este efecto. En estos experimentos se ha dividido el conjunto de entrenamiento en dos partes (entrenamiento y validación) extrayendo un total de 50%, 40%, 30%, 20%, 10% y 0% de los patrones de entrenamiento para realizar validación. Esta validación se produce al devolver el sistema la red que mejores resultados haya producido en el conjunto de validación (lo cual es una estimación del resultado que ofrecerá en el test). En el caso de extraer un 0%, quiere decir que no se ha realizado ninguna validación, con lo que los resultados presentarán efecto de sobreentrenamiento. Los resultados se muestran en las figuras 7.1 a 7.4, en las que se puede ver las precisiones obtenidas en los test hasta un máximo de 2.000.000 ejecuciones de la función de ajuste. Estos resultados se han obtenido tras la

ejecución del sistema con árboles como forma de codificación y sin usar el método de optimización de constantes.

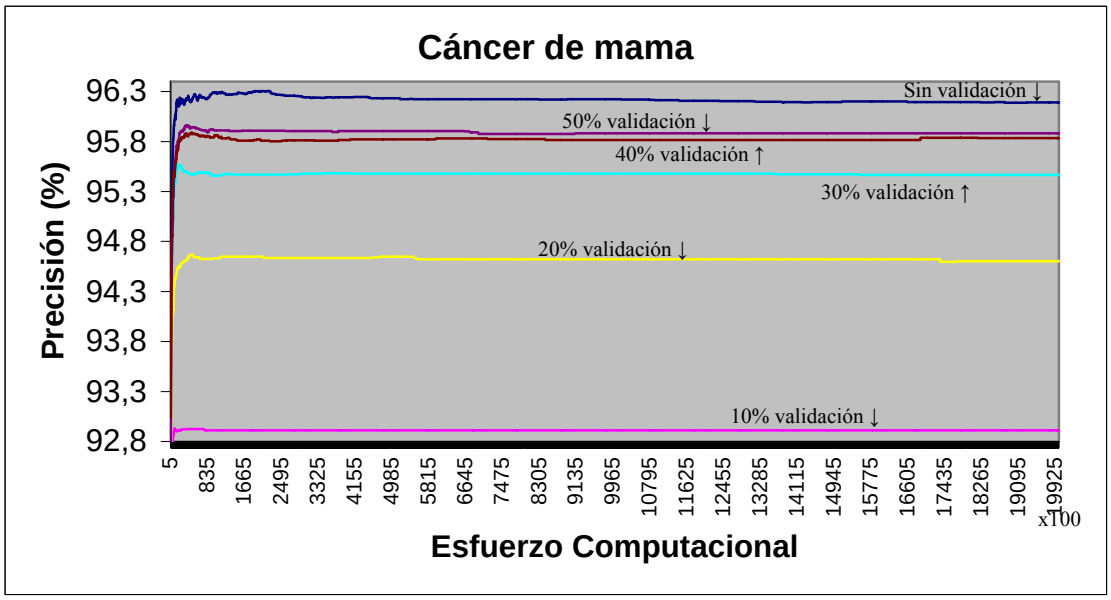


Figura 7.1. *Comparativa de precisiones obtenidas en el test 5x2cv reservando distintas cantidades de valores de entrenamiento para realizar validación en el caso de cáncer de mama*

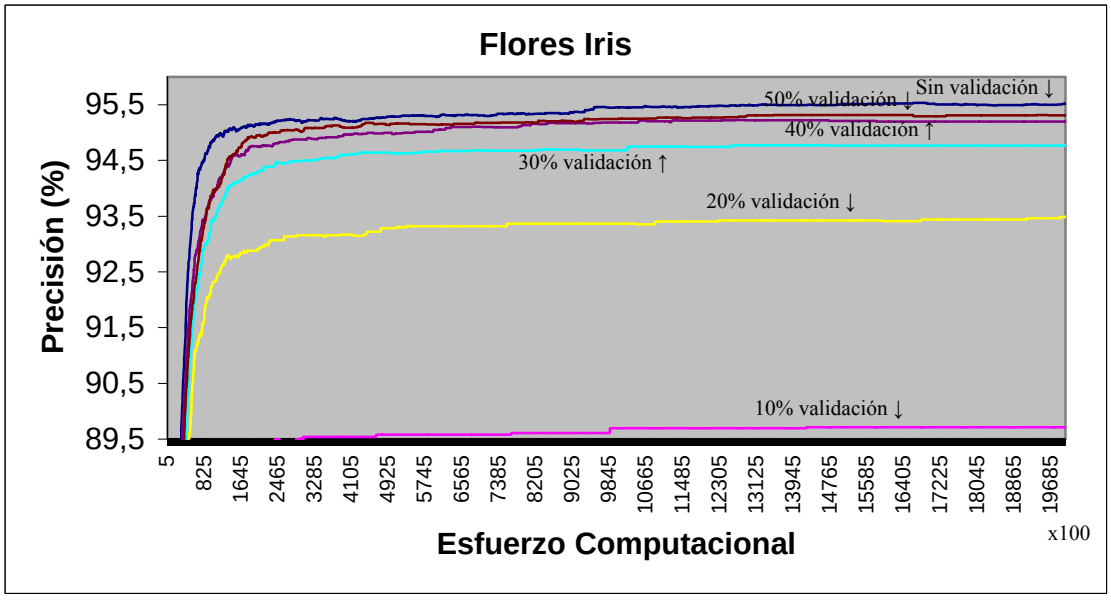


Figura 7.2. *Comparativa de precisiones obtenidas en el test 5x2cv reservando distintas cantidades de valores de entrenamiento para realizar validación en el caso de iris*

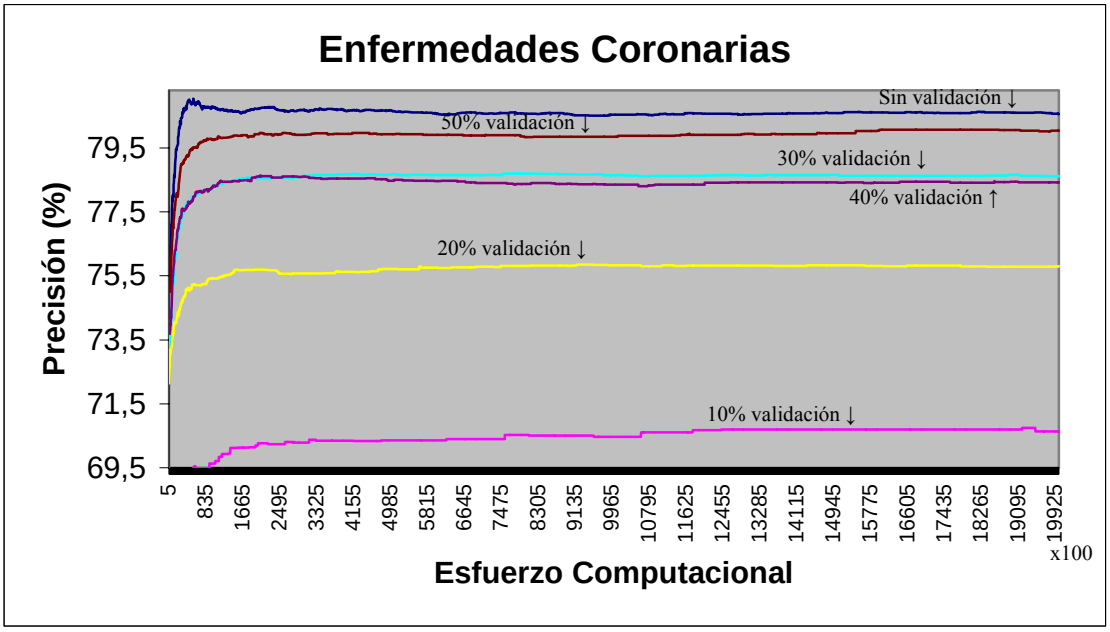


Figura 7.3. *Comparativa de precisiones obtenidas en el test 5x2cv reservando distintas cantidades de valores de entrenamiento para realizar validación en el caso de enfermedades coronarias*

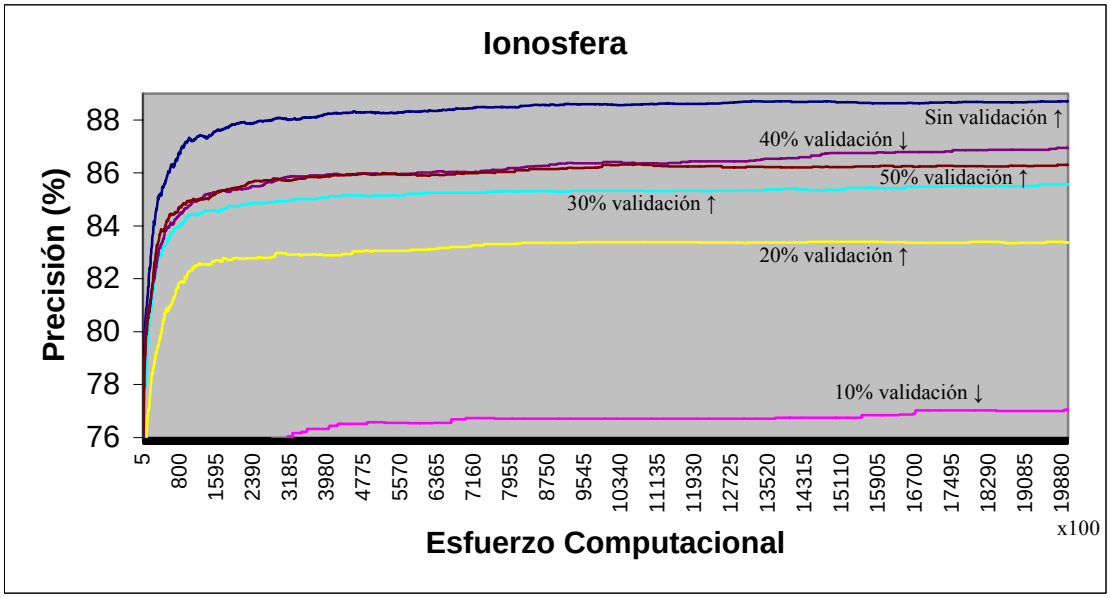


Figura 7.4. *Comparativa de precisiones obtenidas en el test 5x2cv reservando distintas cantidades de valores de entrenamiento para realizar validación en el caso de ionosfera*

A partir de estas figuras, pueden extraerse las dos siguientes observaciones:

- ✓ A medida que aumenta el tamaño del conjunto de validación, aumenta la precisión obtenida.
- ✓ La mejor precisión se obtiene cuando el conjunto de validación está vacío, es decir, no se realiza validación.

El primero de los efectos se produce porque si el conjunto de validación es muy pequeño, no será lo suficientemente representativo del conjunto total de patrones, con lo que la evaluación de las redes con el mismo no conducirá hacia la obtención de redes que generalicen bien y, por tanto, se comporten bien en el test. En consecuencia, cuanto mayor sea, más representativo del conjunto total de patrones será y ofrecerá mejores resultados en el test.

A pesar de lo que pudiera parecer por lo explicado anteriormente, al no realizar validación la precisión obtenida no es la peor, sino la mejor. Esto indica que se obtienen mejores resultados al no realizar validación que haciéndola, lo cual podría parecer una contradicción con lo explicado en las secciones 2.1.4.3 y 5.2, pero que tiene una explicación sencilla:

Para realizar el entrenamiento correctamente, el conjunto de entrenamiento debe de ser lo suficientemente representativo de la base de datos. Para realizar la validación correctamente, debe de ocurrir lo mismo. El conjunto de validación debe de ser representativo de la base de datos. Si alguno de lo dos conjuntos es demasiado pequeño, no representa de forma adecuada los datos, y, por tanto, el proceso de desarrollo de RR.NN.AA. no se realiza de forma adecuada. El sistema genera redes que han aprendido según un conjunto de patrones que representan una parte pequeña del conocimiento que se deseaba que adquiriesen y, por lo tanto, este conocimiento aprendido por las redes sólo es una parte del deseado, por lo que se comportarán mal en el test.

Este efecto se produce al usar bases de datos pequeñas y posteriormente particionarlas en entrenamiento, validación y test. En este caso, el método 5x2cv requiere reservar el 50% de los patrones para realizar el test. Como puede verse en las gráficas 7.1 a 7.4, el mejor de los resultados ha sido obtenido, en la mayoría de las ocasiones, reservando un 50% de los patrones restantes para realizar la validación. Por lo tanto, de la base de datos original, se toman un 25% para entrenamiento, un 25% para validación y un 50% para test. Si la base de datos es muy pequeña, es muy probable que, después de particionarla, el conjunto de entrenamiento o el de validación (o, lo más probable, ambos) no sean representativos de los datos originales, con lo que se producirá el efecto antes mencionado, de crear redes que no representen el conocimiento deseado, sino sólo una parte del mismo. Para que no ocurra esto, hay que evitar trabajar con conjuntos de patrones demasiado pequeños. En este caso, esto se

hace al no dividir el conjunto de entrenamiento en entrenamiento y validación, es decir, no realizar validación. Al no hacerla, se está realizando un sobreentrenamiento de las redes generadas. Sin embargo, y como se puede ver en las gráficas, la precisión obtenida es mayor cuando esas redes se sobreentrenan que cuando son generadas con conjuntos de entrenamiento y validación pequeños.

En caso de contar con un conjunto de patrones lo suficientemente amplio, lo más recomendable sería particionar éste en tres partes: entrenamiento, test y validación. En este caso, la precisión ofrecida por el sistema sin validación (sobreentrenando) sería menor que al usar validación. Sin embargo, al no ser esto posible, se divide sólo en entrenamiento y test.

7.2 Resultados obtenidos

En estudios previos se utiliza el método 5x2cv para comparar diferentes técnicas basadas en métodos evolutivos para generar y entrenar RR.NN.AA. [Cantú-Paz 2005]. En ese trabajo, se presentan como resultados las medias de las precisiones obtenidas en cada uno de los 10 resultados que genera este método. Estos valores se toman como base para comparar la técnica descrita en este trabajo con algunas de las existentes más conocidas, que pueden ser clasificadas de la siguiente manera:

- Técnicas basadas en entrenamiento de RR.NN.AA.
- Técnica de selección de variables.
- Técnicas de diseño de RR.NN.AA. mediante CE.

A la hora de comparar los resultados obtenidos por este método con los obtenidos por otras herramientas, extraídos del trabajo tomado como referencia, un concepto importante es el de esfuerzo computacional. Este puede definirse como el número de veces que se evalúa el archivo de patrones de entrenamiento. Dado que los tiempos de cómputo mostrados en el trabajo de referencia no pueden ser aplicados en este estudio, puesto que las máquinas utilizadas son distintas, se utiliza como base el esfuerzo computacional, el cual supone una medida más universal del esfuerzo real de computación, puesto que no tiene la limitación de estar basada en su aplicación en una determinada máquina.

Para cada resultado mostrado en el trabajo de referencia, se ejecuta el sistema un determinado número de veces para realizar la comparativa. El método utilizado para la comparación es la siguiente:

1. Se calcula el esfuerzo computacional que ha sido necesario para lograr ese determinado valor. Esto se puede calcular a partir de los datos conocidos del algoritmo que se ha aplicado: tamaño de población, número de generaciones, número de ciclos del algoritmo BP, etc., y la forma en la que se aplican dichas técnicas. A partir de todos estos datos, puede calcularse cuántas veces se ha evaluado el conjunto de entrenamiento para alcanzar ese determinado resultado. A pesar de que este valor de esfuerzo puede representar una cantidad aproximada, es una buena estimación del valor real del esfuerzo computacional que ha sido realizado, y por lo tanto se utiliza como el real.
2. Se ejecuta el sistema 10 veces, cada una de ellas vez entrenando con cada conjunto de entrenamiento del método 5x2cv y haciendo test con el correspondiente conjunto de test. Sin embargo, y para minimizar la componente aleatoria, cada uno de estos 10 entrenamientos con cada conjunto de entrenamiento se realiza 20 veces, con lo que cada valor de cada test en realidad es una media de 20 resultados de test. Como resultado de este proceso, se obtienen 10 valores de test.
3. Se realiza la media de estos 10 valores de test, lo cual ofrece un único valor de precisión obtenido en test, resultado de ejecutar el sistema hasta un determinado número de evaluaciones del conjunto de entrenamiento. Este valor es el que se utiliza en la comparación.

Con respecto a los parámetros utilizados para generar RR.NN.AA. con la técnica descrita en este trabajo, se han utilizado los mismos que los recomendados en la sección 6, es decir, los expuestos en la tabla 6.7.

Para cada tabla de comparativas que se muestra, cada casilla se corresponde a un determinado problema con una determinada técnica. En ellas se muestran 6 valores distintos:

- ✓ El superior corresponde con el valor de precisión obtenido en el trabajo de referencia [Cantú-Paz 2005].

- ✓ Debajo de ese valor, figura el del esfuerzo computacional que ha sido necesario para obtener ese determinado valor de precisión con esa determinada técnica.
- ✓ Debajo del valor del esfuerzo figuran 4 valores distintos, ordenados en una matriz de dos filas y dos columnas, que representan los resultados obtenidos por la técnica aquí descrita. La fila superior se corresponde con los resultados obtenidos con codificación en forma de árbol, sin usar el sistema de optimización de pesos (izquierda) y usándolo (derecha). La fila inferior, a su vez, tiene los resultados usando grafos como forma de codificación, igualmente sin y con el sistema de optimización de pesos. Respecto a estos 4 valores, fueron hallados, como ya ha sido descrito, después de ejecutar el sistema repetidas veces hasta alcanzar un determinado nivel de esfuerzo computacional, el calculado para esa técnica en ese problema. Si el esfuerzo computacional necesario por cada técnica es inferior a 2.000.000 de ejecuciones de la función de ajuste, los valores de precisión mostrados por la técnica descrita en este trabajo será la correspondiente a ese esfuerzo. Sin embargo, si es mayor, los valores de precisión mostrados serán los correspondientes a los obtenidos tras 2.000.000 de ejecuciones de la función de ajuste.

Finalmente, la última fila de cada tabla contiene 5 valores que se corresponden a los valores medios de precisión obtenidos con una determinada técnica en todos los problemas, y los 4 obtenidos en comparación con la técnica aquí descrita, dispuestos de forma ya explicada.

7.2.1 Técnicas basadas en entrenamiento de RR.NN.AA.

El primer conjunto de técnicas con las que se compara son aquellas en las que simplemente se utilizan algoritmos genéticos para entrenar RR.NN.AA. con topología ya fijada, lo cual ha requerido un estudio previo por parte de los autores. El sistema presentado en este trabajo desarrolla la arquitectura, por lo que hay que tenerlo en cuenta a la hora de comparar resultados. La tabla 7.1 muestra un resumen de las topologías de red utilizadas en estudios anteriores [Cantú-Paz 2005] en comparación con una media de las topologías de las redes resultantes en el método aquí propuesto, usando árboles como forma de codificación, pero sin usar el sistema de optimización de constantes (extraídas de la tabla 6.2). En esa tabla puede verse, además, el número de ciclos del algoritmo BP aplicados en aquellos algoritmos que lo requieren (se describen

más adelante). Es importante tener en cuenta que, si bien en el citado trabajo se presentan resultados obtenidos utilizando RR.NN.AA. con una capa oculta totalmente conectada y un determinado número de neuronas en la capa oculta, las topologías de red que se presentan según el método propuesto en este estudio no se corresponden a una topología clásica dividida en capas, puesto que las neuronas ocultas pueden tener cualquier tipo de interconexión entre ellas y las de entrada y salida, pudiendo haber también conexiones de neuronas de entrada a las salidas.

	Aquí propuesto			[Cantú-Paz 2005]			
	Número de entradas	Número de neuronas ocultas	Número de neuronas de salida	Número de entradas	Número de neuronas ocultas	Número de neuronas de salida	*Epochs*
Cáncer de mama	9	3.3	1	9	5	1	20
Iris	4	8.85	3	4	5	3	80
Enfermedades coronarias	13	2.9	1	13	5	1	40
Ionosfera	34	9.25	1	34	10	1	40

Tabla 7.1. *Arquitecturas de las redes*

7.2.1.1 Entrenamiento de las redes mediante técnicas evolutivas

En la tabla 7.2 se comparan los resultados obtenidos por el método aquí propuesto en comparación con los obtenidos con el algoritmo de *backpropagation* (BP) tradicional y entrenados mediante AA.GG., bien sea con codificación binaria o real (algoritmo G3PCX [Deb 2002]).

Los resultados tomados utilizando codificación binaria corresponden a la siguiente configuración del algoritmo genético:

- Cromosoma de 16 bits por peso en el rango [-1, 1].
- Tamaño de población de n = $\lfloor 3\sqrt{l} \rfloor$, donde l es el número de bits del cromosoma.
- Tasa de mutación de 1/l.
- Torneo de dos individuos sin reemplazo.
- En cada generación se reemplazaba toda la población, sin hacer elitismo.

- Después de 100 generaciones se paró el algoritmo y el resultado fue el individuo con la precisión más alta.

Por su parte, el algoritmo genético con codificación real, denominado G3PCX [Deb 2002], presenta una configuración similar, con las siguientes variantes:

- Cromosoma de tamaño l, donde l es el número de pesos.
- Tamaño de población: n = $\lfloor 30\sqrt{l} \rfloor$, donde l es el número de pesos.
- El algoritmo fue detenido después de n iteraciones sin mejora en la mejor solución, o después de 50n ejecuciones de la función de ajuste.
- Utiliza un algoritmo de tipo "steady-state", en el que una única población evoluciona y no es reemplazada por otra resultado de aplicar operadores genéticos a la misma.

	BP	AG	
		G3PCX	Binario
Cáncer de mama	96.39	98.94 2247200 96.19 95.90 96.25 96.15	98.88 8400 95.98 95.74 96.17 95.96
Iris	94.53	89.73 1566450 95.51 95.11 95.64 94.83	88.67 7000 84.48 85.23 84.37 83.72
Enfermedades coronarias	78.17	90.42 6055200 80.56 79.85 80.53 79.25	87.72 13900 79.02 77.44 79.63 78.60
Ionosfera	84.77	64.10 15736050 88.72 90.05 88.62 91.03	74.10 22400 83.75 83.25 84.19 84.36
Media	88.46	85.79 90.24 90.22 90.26 90.31	87.34 85.80 85.41 86.09 85.66

Tabla 7.2. *Comparativas de resultados con métodos de entrenamiento de RR.NN.AA.*

7.2.1.2 Refinando los pesos mediante BP

Además, también se han realizado comparaciones con los resultados obtenidos utilizando estrategias "lamarckianas" y "baldwinianas". En estos casos, también se han tomado los resultados de realizar otro entrenamiento mediante un AG (con codificación binaria), pero esta vez refinando los pesos dentro del algoritmo genético mediante el uso del algoritmo BP.

	Estrategias baldwinianas			Estrategias lamarckianas		
	1BP	2BP	5BP	1BP	2BP	5BP
Cáncer de mama	98.48 8820 95.99 95.74 96.20 95.94	98.91 9240 96.02 95.82 96.26 95.92	99.03 10500 96.06 95.82 96.25 95.94	98.88 8820 95.99 95.74 96.20 95.94	98.74 9240 96.02 95.82 96.26 95.92	99.08 10500 96.06 95.82 96.25 95.94
Iris	89.47 7350 84.63 85.50 84.60 83.88	91.07 7700 84.89 85.65 84.76 84.21	87.20 8750 85.34 85.91 85.33 84.95	89.20 7350 84.63 85.50 84.60 83.88	88.00 7700 84.89 85.65 84.76 84.21	88.13 8750 85.34 85.91 85.33 84.95
Enfermedades coronarias	88.68 14595 79.15 77.49 79.69 78.67	89.25 15290 79.30 77.51 79.70 78.76	89.21 17375 79.71 77.53 80.01 78.81	86.45 14595 79.15 77.49 79.69 78.67	88.82 15290 79.30 77.51 79.70 78.76	87.98 17375 79.71 77.53 80.01 78.81
Ionosfera	68.43 23520 84.01 83.35 84.23 84.52	69.43 24640 84.12 83.42 84.63 84.77	67.86 28000 84.50 83.90 84.73 85.24	65.12 23520 84.01 83.35 84.23 84.52	65.88 24640 84.12 83.42 84.63 84.77	64.65 28000 84.50 83.90 84.73 85.24
Media	86.26 85.94 85.52 86.18 85.75	87.16 84.58 85.60 86.33 85.91	84.89 86.40 86.04 86.58 86.23	84.91 85.94 85.52 86.18 85.75	85.36 86.08 85.60 86.33 85.91	84.96 86.40 86.04 86.58 86.23

Tabla 7.3. *Resultados de las comparativas con estrategias evolutivas de entrenamiento de RR.NN.AA., utilizando estrategias lamarckianas y baldwinianas en un 5% de la población*

En el caso de estrategia "lamarckiana", se aplica el algoritmo BP a los cromosomas de los individuos durante la evaluación de la función de ajuste en cierto porcentaje de individuos de la población. Los cambios que sufren permanecen en los mismos. El valor de ajuste resultante será el resultante de aplicar el algoritmo de BP. Por su parte, en el caso de la "baldwiniana", la estrategia es similar, con la diferencia de que los cambios que sufren los individuos tras la ejecución del algoritmo de BP no

permanecen en los cromosomas, sino que el algoritmo BP sólo se ejecuta para calcular el valor del ajuste. En estos experimentos, además, el mejor conjunto de pesos hallado en la estrategia "baldwiniana" fue utilizado para entrenar una red mediante el algoritmo BP. Las tablas 7.3 y 7.4 muestran la comparativa entre el método propuesto en este trabajo y las estrategias "lamarckianas" y "baldwinianas", aplicando estas estrategias al 5% y al 100% de la población, y realizando 1, 2 y 5 iteraciones del algoritmo BP en los procesos de refinamiento de los pesos. Los AA.GG. utilizados en estos experimentos tienen los mismos parámetros que el AG binario utilizado en el caso anterior.

	Estrategias baldwinianas						Estrategias lamarckianas					
	1BP		2BP		5BP		1BP		2BP		5BP	
Cáncer de mama	98.83		98.86		98.60		98.88		98.94		98.86	
	16800		25200		50400		16800		25200		50400	
	96.18	95.92	96.21	95.97	96.27	96.12	96.18	95.92	96.21	95.97	96.27	96.12
	96.33	96.11	96.35	96.23	96.27	96.34	96.33	96.11	96.35	96.23	96.27	96.34
Iris	91.33		89.87		91.07		92.40		93.20		92.00	
	14000		21000		42000		14000		21000		42000	
	86.56	86.83	87.66	87.41	91.78	92.08	86.56	86.83	87.66	87.41	91.78	92.08
	86.47	85.99	88.09	86.77	93.26	88.53	86.47	85.99	88.09	86.77	93.26	88.53
Enfermedades coronarias	88.58		88.45		88.32		86.87		88.98		87.66	
	27800		41700		83400		27800		41700		83400	
	80.43	78.25	80.89	79.40	80.76	80.59	80.43	78.25	80.89	79.40	80.76	80.59
	80.45	79.41	80.84	79.60	81.24	79.56	80.45	79.41	80.84	79.60	81.24	79.56
Ionosfera	73.88		73.15		72.89		74.25		74.03		73.77	
	44800		67200		134400		44800		67200		134400	
	85.47	86.78	86.27	87.93	87.35	88.93	85.47	86.78	86.27	87.93	87.35	88.93
	85.74	88.45	86.71	90.12	87.59	90.69	85.74	88.45	86.71	90.12	87.59	90.69
Media	88.15		87.58		87.72		88.1		88.78		88.07	
	87.16	86.94	87.75	85.54	89.04	89.43	87.16	86.94	87.75	85.54	89.04	89.43
	87.24	87.49	87.99	88.18	89.59	88.78	87.24	87.49	87.99	88.18	89.59	88.78

Tabla 7.4. *Resultados de las comparativas con estrategias evolutivas de entrenamiento de RR.NN.AA., utilizando estrategias lamarckianas y baldwinianas en un 100% de la población*

7.2.2 Técnica de selección de variables

La siguiente técnica que es utilizada en el trabajo usado para la comparación [Cantú-Paz 2005] se basa en la selección de variables del problema [Yang 1998] [Ozdemir 2001]. Esta técnica se basa en tener un algoritmo genético con codificación

binaria en el que, dentro del cromosoma, cada bit indicará si una determinada variable será utilizada o no para el entrenamiento. La evaluación de los individuos, por tanto, consistirá en entrenar la RNA (que tendrá una estructura fija) con las variables fijadas por el cromosoma del individuo en cuestión. La población fue inicializada de forma aleatoria con $\lfloor 3\sqrt{l} \rfloor$ individuos, pero con un tamaño mínimo de 20. Se utilizó una tasa de cruces con probabilidad 1.0 y una tasa de mutación de $1/l$. Las redes fueron diseñadas y entrenadas de acuerdo con la tabla 7.1. El algoritmo se paró al no encontrar cambio en la mejor solución en 5 generaciones, o bien al alcanzar un límite de 50 generaciones. Sin embargo, este límite nunca se alcanzó, puesto que el algoritmo encontró mucho antes una buena solución. La tabla 7.5 muestra una comparativa de esta técnica con la presentada en este trabajo.

	Selección de variables
Cáncer de mama	96.48 20000 96.22 95.93 96.32 96.14
Iris	93.60 80000 94.48 95.10 95.25 91.69
Enfermedades coronarias	84.72 40000 80.78 79.29 80.79 79.56
Ionosfera	87.00 40000 85.14 85.96 88.37 87.85
Media	90.45 89.15 89.07 90.18 88.81

Tabla 7.5. *Resultados de las comparativas con el método de selección de variables*

7.2.3 Técnicas de diseño de RR.NN.AA. mediante CE

Finalmente, esta categoría abarca un conjunto de técnicas que permiten el desarrollo automatizado de RR.NN.AA. Estas técnicas sí son un conjunto representativo con el que realizar la comparación de la herramienta descrita en este trabajo, puesto que las anteriormente mencionadas no realizaban un diseño de la topología de la red, sino únicamente un entrenamiento de la misma, lo cual se aleja del propósito de esta Tesis. Concretamente, se compara con cuatro técnicas distintas:

- Matriz de conectividad.
- Poda.
- Búsqueda de parámetros de redes.
- Gramática de reescritura de grafos.

La técnica de matriz de conectividad se basa en representar la topología de una red como una matriz binaria: el elemento (i,j) de la matriz tendrá un valor de 1 si existe una conexión entre los nodos i y j, y cero en caso contrario. Un algoritmo genético con codificación binaria puede ser fácilmente utilizado, puesto que el cromosoma se obtiene fácilmente concatenando las filas de la matriz [Belew 1991]. En este caso, se utilizaron el número de neuronas ocultas indicado en la tabla 7.1. Se han permitido conexiones entre las entradas y las salidas, con lo que la longitud de los cromosomas es l = (ocultas + salidas)*entradas + ocultas*salidas. Se utilizó un cruce de multipunto con una probabilidad de 1.0 con l/10 puntos de cruce y una tasa de mutación de 1/l. La población tiene un tamaño de $\lfloor 3\sqrt{l} \rfloor$ individuos con un mínimo de 20. El algoritmo se paró después de 5 generaciones sin mejora en la mejor solución o si se alcanzó un máximo de 50 generaciones.

La técnica de poda se basa en una representación similar a la anterior. Sin embargo, el método es distinto. Se parte de una red totalmente conectada [Reed 1993], que se entrena mediante el algoritmo BP según los parámetros de la tabla 7.1. Una vez se tiene esta red, se ejecuta el algoritmo evolutivo. La función de evaluación de cada individuo consistirá en tomar la red anterior entrenada y eliminar aquellos pesos cuyo valor en la matriz de conectividad sea igual a 0, para posteriormente evaluarla con el conjunto de entrenamiento, sin entrenarla más. Para resolver estos problemas, se partió

de la topología mostrada en la tabla 7.1, con la misma configuración de parámetros que en el caso anterior.

La búsqueda de los parámetros supone una óptica distinta, puesto que en este caso se utiliza un algoritmo evolutivo para encontrar los parámetros generales del diseño y entrenamiento de redes [Belew 1991] [Marshall 1991]. En este caso, estos parámetros son número de neuronas ocultas y parámetros del algoritmo de BP y rango inicial de pesos. La longitud del cromosoma fue de 36 bits, divididos de la siguiente manera:

- 5 bits para la tasa de aprendizaje y el coeficiente β de la función de activación, en el rango [0,1].
- 5 bits para el número de neuronas ocultas, en el rango [0, 31].
- 6 bits para el número de epochs del algoritmo BP.
- 20 bits para los rangos superior e inferior de los pesos iniciales (10 bits para cada valor), y sus rangos fueron [-10, 0] y [0, 10] respectivamente.

La evaluación de un individuo consiste en la construcción de la red, su inicialización y el entrenamiento de la misma según los parámetros. La población tuvo 25 individuos y fue inicializada aleatoriamente. El algoritmo utilizó cruce de dos puntos con una probabilidad de 1.0 y una tasa de mutación de 0.04. Como en el resto de los experimentos, se utilizó selección de torneo de dos individuos sin reemplazamiento y la ejecución se paró después de 5 generaciones sin cambio en la mejor solución o al llegar a un límite de 50 generaciones.

Finalmente, la gramática de reescritura de grafos, basada en [Kitano, 1990] consiste en una matriz de conectividad que representa la red, pero, al contrario que en los casos anteriores, la matriz no se codifica directamente en el cromosoma, sino que se utiliza una gramática para generar la matriz. El cromosoma solamente contiene reglas que convierten cada elemento de la matriz en submatrices de tamaño 2x2. En esta gramática hay 16 símbolos terminales que son matrices de tamaño 2x2 y 16 símbolos no terminales. Las reglas tienen la forma n→m, donde n es uno de los escalares no terminales, y m es una matriz 2x2 de no terminales. Existe además un símbolo de comienzo y el número de pasos está fijado por el usuario.

El cromosoma contiene las 16 reglas de la siguiente forma: contiene los 16 lados derechos de las reglas, puesto que el lado izquierdo está implícito en la posición de la

regla. Para evaluar el ajuste de un individuo, se decodifican las reglas y se construye la matriz de conectividad por medio de las mismas. A partir de esta matriz se genera la red, que se entrena mediante BP.

Para la aplicación en estos problemas, se limitó el número de pasos a 8, con lo que el resultado son redes con 256 elementos como máximo. El tamaño del cromosoma, por lo tanto, es de 256 bits (4 matrices binarias de 2x2 para cada una de las 16 reglas). El tamaño de población tomado es de 64 individuos, con cruce multipunto con una probabilidad de 1.0 y l/10 puntos de cruce, y una tasa de mutación de 0.004. El algoritmo se paró después de 5 generaciones sin mejora en el mejor individuo o al cabo de 50 generaciones.

Los resultados obtenidos con estos 4 métodos, en comparación con el propuesto en este trabajo, pueden verse en la tabla 7.6.

	Matriz de conectividad		Poda		Búsqueda de parámetros		Gramática de reescritura de grafos	
Cáncer de mama	96.77		96.31		96.69		96.71	
	92000		4620		100000		300000	
	96.27	96.20	95.66	95.60	96.28	96.16	96.24	96.07
	96.27	96.31	95.79	95.97	96.27	96.29	96.31	96.22
Iris	92.40		92.40		91.73		92.93	
	320000		4080		400000		1200000	
	95.22	95.25	81.43	83.65	95.20	95.25	95.47	95.10
	95.49	94.54	81.58	81.35	95.52	94.68	95.66	94.78
Enfermedades coronarias	76.78		89.50		65.89		72.8	
	304000		7640		200000		600000	
	80.68	80.02	77.75	76.90	80.71	80.01	80.57	80.01
	81.11	79.50	78.28	78.26	81.05	79.60	80.97	79.28
Ionosfera	87.06		83.66		85.58		88.03	
	464000		11640		200000		600000	
	88.29	89.62	81.68	82.57	87.83	89.24	88.31	89.73
	88.34	90.95	82.37	83.25	87.81	90.90	88.36	90.98
Media	88.25		90.46		84.97		87.61	
	90.11	90.27	84.13	84.68	90.00	90.16	90.14	90.22
	90.30	90.32	84.50	84.70	90.16	90.36	90.32	90.31

Tabla 7.6. *Resultados de las comparativas con diversos métodos de diseño de redes*

7.3 Discusión

La sección anterior ha mostrado los buenos resultados obtenidos por el método aquí propuesto. Sin embargo, estos buenos resultados son solo una de las ventajas que ofrece el sistema. Esta sección analiza esta y otras ventajas que presenta el método descrito en este trabajo.

7.3.1 Resultados

A la vista de las tablas anteriores, los resultados obtenidos por el método aquí propuesto son del mismo orden que las presentadas en [Cantú-Paz 2005].

Las secciones 7.2.1 y 7.2.2 muestran una comparativa de los resultados obtenidos con este método y otras técnicas existentes de entrenamiento de RR.NN.AA. mediante técnicas CE y técnicas híbridas. Los valores de precisión obtenidos por la técnica aquí descrita en los test 5x2cv muestran que los resultados que ofrece son similares a los obtenidos con otras herramientas, llegando a mejorarlos en la mayoría de las ocasiones, especialmente en aquellos casos en los que se requiere mucha capacidad computacional (caso de técnicas híbridas como estrategias "lamarckianas"). Sin embargo, las técnicas con las que se compara parten de una topología de red ya fijada, con lo que sigue siendo necesaria la intervención del experto humano en esa parte. La herramienta aquí descrita, en cambio, y como se ha mostrado, es capaz de ofrecer resultados incluso mejores sin necesidad de tal intervención del experto.

Por otra parte, la sección 7.2.3 realiza una comparativa con un conjunto de técnicas que no precisan de una arquitectura prefijada de antemano, es decir, que ahora ya no precisan la intervención del experto para fijar la topología y conectividad de la RNA. Estas técnicas, por aunar evolución de arquitectura con entrenamiento de los pesos, requieren de una gran carga computacional. La tabla 7.6 muestra que solamente en una de las comparativas la técnica aquí descrita ofrece resultados peores. En el resto, los valores de precisión obtenidos muestran unos resultados ampliamente mejores que los ofrecidos por otras técnicas. Sin embargo, la técnica que ofrece resultados mejores que la aquí descrita, la técnica de selección de variables, sí necesita de cierto trabajo por parte del experto, con lo que no ofrece la ventaja que se busca de independencia.

Para realizar cada comparación, se ha calculado el esfuerzo computacional necesitado por esas técnicas para alcanzar ese nivel de precisión y posteriormente se ha

ejecutado la técnica aquí descrita hasta llegar a ese nivel de esfuerzo, presentándose en cada comparativa la precisión obtenida cuando se llegó a ese nivel. Sin embargo, en muchas ocasiones este nivel de precisión se había obtenido mucho antes de llegar a ese punto. Las figuras 7.1 a 7.4 muestran el coste computacional necesario para obtener los resultados presentados en las tablas anteriores. Como se puede ver, en muchos casos la convergencia de este algoritmo es muy temprana y los resultados se obtienen con un esfuerzo computacional mucho menor que el mostrado en las tablas, especialmente en la tabla 7.6, que realiza la comparativa con otros métodos de generación automática de RR.NN.AA.

En algunos problemas, como en el de cáncer de mama o enfermedades coronarias, existe un descenso en la precisión obtenida tras un rápido ascenso inicial. Esto es provocado porque las redes obtenidas se sobreentrenan; es decir, tras alcanzar un máximo en el test siguen aprendiendo de los patrones de entrenamiento, pero poco a poco pierden la capacidad de generalización. Este problema, como ya se ha explicado en las secciones 2.1.4.3 y 5.2, se produce por la inexistencia de un conjunto de validación. Si se trabajase con bases de datos más grandes, lo apropiado sería contar con un conjunto de validación para eliminar este efecto indeseado, con lo que no existirían caídas en los valores de precisión.

En otros problemas, en cambio, los valores de precisión siguen aumentando, con lo que cuando se llega al máximo de 2.000.000 de evaluaciones del conjunto de patrones de entrenamiento todavía no se había alcanzado el valor máximo de precisión que puede ofrecer esta técnica.

En resumen, puede decirse que los valores de precisión mostrados en las tablas no son, ni mucho menos, los mejores que pueden ofrecer esta técnica, puesto que para unos casos estos valores mejores han sido alcanzados previamente y se mantendrían de usar un conjunto de validación. En otros casos la parada del algoritmo es muy prematura, con lo que los resultados mejoran si se deja ejecutando más tiempo.

Las figuras 7.5 a 7.8 muestran, además, para cada problema, una comparativa de las precisiones obtenidas utilizando las dos formas de codificación explicadas, y haciendo uso y sin hacerlo del sistema de optimización de pesos.

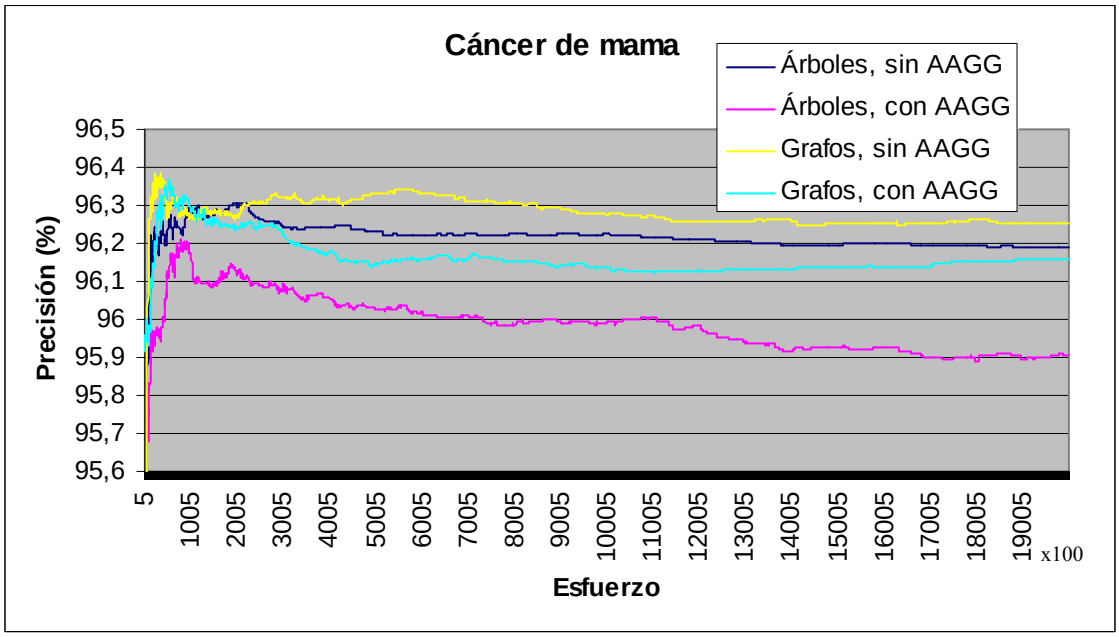


Figura 7.5. *Precisiones obtenidas para el problema de cáncer de mama*

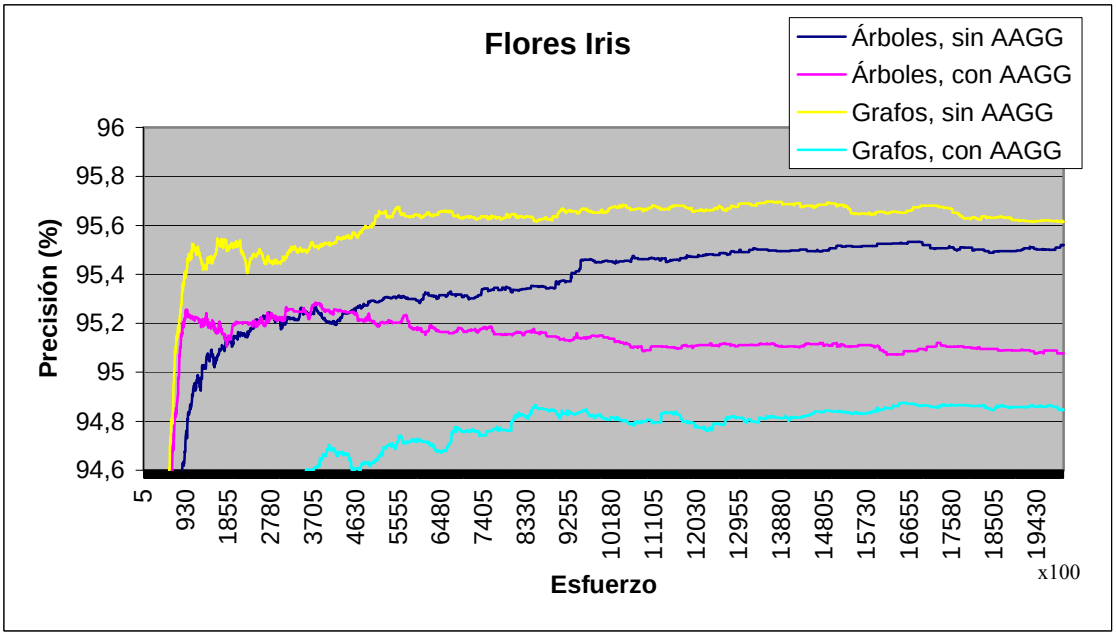


Figura 7.6. *Precisiones obtenidas para el problema de iris*

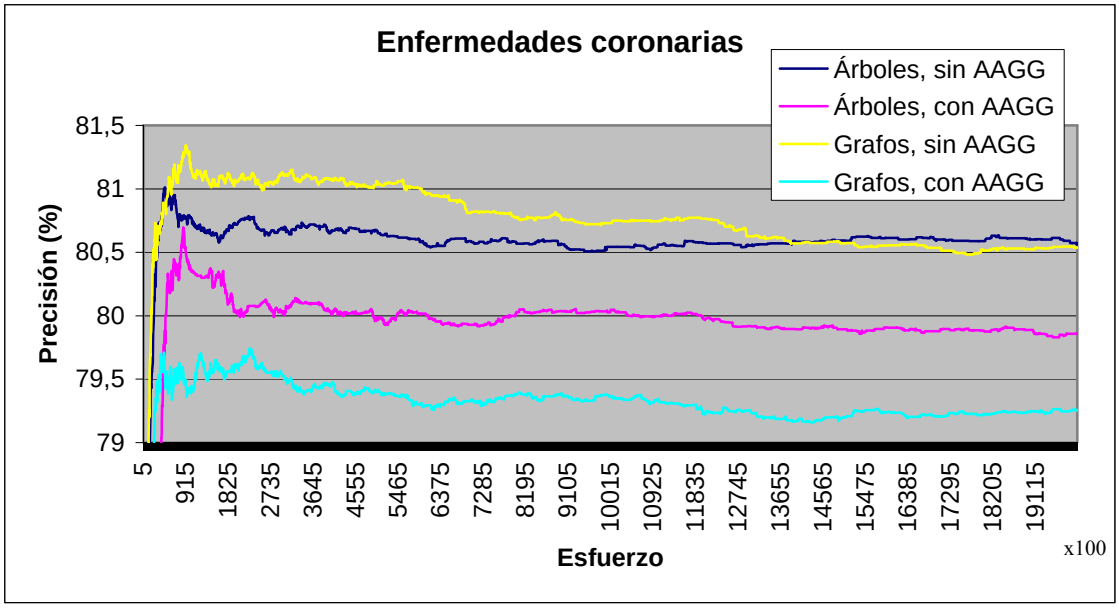


Figura 7.7. *Precisiones obtenidas para el problema de enfermedades coronarias*

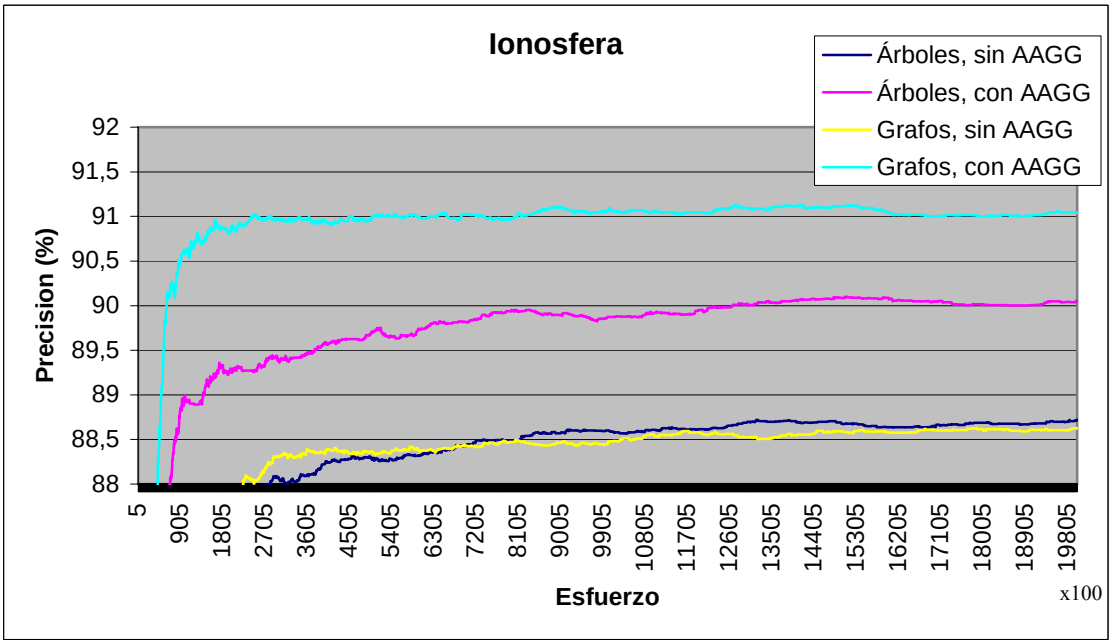


Figura 7.8. *Precisiones obtenidas para el problema de ionosfera*

A la vista de estas gráficas, no es fácil determinar si es recomendable el uso de árboles o grafos como forma de codificación, o si se obtienen mejores resultados mediante el uso del sistema de optimización de constantes. Las gráficas correspondientes a los problemas de cáncer de mama, flores iris y enfermedades coronarias parecen indicar que el sistema tiene un mejor comportamiento cuando no se emplea el sistema de optimización de pesos y, en general, con el uso de grafos. Sin

embargo, los resultados obtenidos con el problema de la ionosfera indica lo contrario: en este problema los mejores resultados se han obtenido con el uso del sistema de optimización de constantes.

Sin embargo, es necesario tener en cuenta también que estos problemas han sido resueltos sin reservar patrones para el conjunto de validación, con lo que se puede apreciar en dichas gráficas grandes fluctuaciones en el nivel de precisión obtenido en el test. De utilizar un conjunto de validación, se espera que el nivel de ajuste en el test no tenga grandes caídas. Son estas caídas las que provocan que no sea fácil determinar si es apropiado o no el uso de grafos o el sistema de optimización de constantes.

Si se toman los mejores resultados de test obtenidos para cada problema, se puede crear una comparativa como la mostrada en la tabla 7.7. Estos datos de precisión son los que, con pequeñas diferencias, se espera encontrar si se utiliza un conjunto de validación.

		Cáncer de mama			Iris			Enfermedades coronarias			Ionosfera		
		Precisión	Uso de grafos	Uso de AAGG	Precisión	Uso de grafos	Uso de AAGG	Precisión	Uso de grafos	Uso de AAGG	Precisión	Uso de grafos	Uso de AAGG
Mejor	1	96.38 %	Sí	No	95.69 %	Sí	No	81.34 %	Sí	No	91.13 %	Sí	Sí
	2	96.36 %	Sí	Sí	95.53 %	No	No	81.04 %	No	No	90.09 %	No	Sí
	3	96.30 %	No	No	95.28 %	No	Sí	80.69 %	No	Sí	88.72 %	No	No
Peor	4	96.21 %	No	Sí	94.87 %	Sí	Sí	79.74 %	Sí	Sí	88.62 %	Sí	No

Tabla 7.7. *Comparativa de los mejores resultados obtenidos*

Por lo tanto, de utilizar un conjunto de validación los resultados obtenidos en los 10 tests del método 5x2cv habrían sido mucho mejores y los datos mostrados en esta tabla parecen indicar que la mejor configuración se obtiene mediante el uso de grafos y no utilizando el sistema de optimización de constantes. Si se toman estos mejores resultados de test, presentados en la tabla 7.7, como los resultados finales y se realiza la comparación con las técnicas utilizadas en la sección 7.2.3 para el diseño y entrenamiento automatizado de RR.NN.AA. utilizando herramientas de CE, puede elaborarse una tabla comparativa, en la que se muestra, para cada problema las técnicas ordenadas según el nivel de precisión. Esto puede verse en la tabla 7.8.

		Cáncer de mama		Iris		Enfermedades Coronarias		Ionosfera	
Mejor	1	Matriz de conectividad	96.77 %	**Aquí propuesto**	**95.69 %**	Poda	89.50 %	**Aquí propuesto**	**91.13 %**
	2	Gramática de reescritura	96.71 %	Gramática de reescritura	92.93 %	**Aquí propuesto**	**81.34 %**	Gramática de reescritura	88.03 %
	3	Búsqueda de parámetros	96.69 %	Matriz de conectividad	92.40 %	Matriz de conectividad	76.78 %	Matriz de conectividad	87.06 %
	4	**Aquí propuesto**	**96.38 %**	Poda	92.40 %	Gramática de reescritura	72.80 %	Búsqueda de parámetros	85.58 %
Peor	5	Poda	96.31 %	Búsqueda de parámetros	91.73 %	Búsqueda de parámetros	65.89 %	Poda	83.66 %

Tabla 7.8. *Comparativa de los mejores resultados obtenidos en el test con otros métodos de generación y entrenamiento de RR.NN.AA.*

Según lo que se puede ver en esta tabla, los resultados obtenidos por el método aquí descrito son mejores que los ofrecidos por el resto de las técnicas en la mayoría de los problemas, excepto en el de cáncer de mama. Sin embargo, en este problema la diferencia en las precisiones obtenidas es muy pequeña, tanto que, sólo hay una variabilidad del 0.40% entre los resultados obtenidos por la técnica aquí descrita y la que mejor resultado ha ofrecido.

En otro de los problemas, el de enfermedades coronarias, la técnica aquí propuesta presentó los segundos mejores resultados, por detrás de la técnica de poda. Sin embargo, esta última técnica sólo ha ofrecido buenos resultados en ese problema particular, con unas precisiones muy bajas en el resto de los problemas, con lo que se ha revelado como una técnica bastante inestable.

Haciendo una media con la posición en esta tabla de cada técnica se podría hacer una cuantificación de lo buenas que son estas técnicas en la resolución de estos problemas. De esta forma, se pueden ordenar las técnicas en función de su eficacia a la hora de resolver estos problemas. Esto puede verse reflejado en la tabla 7.9.

		Técnica	Media posición tabla 7.8
Mejor	1	**Aquí propuesta**	2
	2	Gramática de reescritura	2.5
	3	Matriz de conectividad	2.5
	4	Poda	3.75
Peor	5	Búsqueda de parámetros	4.25

Tabla 7.9. *Técnicas ordenadas según el grado de eficacia*

En esta tabla puede verse que la técnica aquí propuesta ocupa el primer lugar en eficacia, con lo que puede concluirse que los resultados que ofrece en media son los mejores con independencia del problema a resolver.

7.3.2 Independencia del experto

Como ya se ha explicado, uno de los objetivos de este trabajo era desarrollar un sistema en el que el experto tuviera una mínima participación, para eliminar el excesivo esfuerzo que tiene que realizar en el proceso clásico de desarrollo de RR.NN.AA. El desarrollo de este sistema ha llevado a la aparición de una serie de parámetros (descritos en la sección 5.3).

En un principio, podría pensarse que el hecho de tener que ajustar toda esa serie de parámetros elimina la independencia con el experto que se deseaba conseguir. Sin embargo, la sección 6 desarrolla una serie de experimentos encaminada a conseguir un conjunto de parámetros que ofrezcan buenos resultados con problemas de distinta complejidad. Los resultados mostrados en esa sección muestran una serie de intervalos para cada parámetro en los cuales el sistema se mantiene robusto, sin mostrar grandes cambios en el error que comete, por lo que el valor exacto de que se asigne a cada parámetro no influye demasiado. Durante esa sección se adoptan un conjunto de valores para dichos parámetros (dentro de esos intervalos), que son los utilizados en la sección 7.2 para realizar la comparación con otras técnicas, en la resolución de problemas con niveles de complejidad muy diverso. De esta forma, al no tener que ajustar ningún parámetro del sistema, el conjunto de acciones que tiene que realizar el experto para poder utilizar este sistema es mínimo. Se refiere a las acciones clásicas en las técnicas de aprendizaje máquina, y son referidas fundamentalmente al análisis y preprocesado de los datos.

Además, las técnicas con las que se ha comparado la herramienta descrita en este trabajo sí que requieren cierta participación por parte del experto. Tanto las usadas en la sección 7.2.1 como en la sección 7.2.2 requieren un proceso previo de diseño de la red. En cambio las descritas en la sección 7.2.3 ya realizan esta tarea de una forma automatizada, pero aún así todas ellas requieren de cierta participación por parte del experto, que debe de realizar tareas tales como el diseño de una red inicial (en el caso de la técnica de poda) o ajustar cómo pueden ser las redes que se generen (en el caso de técnicas como búsqueda de parámetros o gramática de reescritura de grafos), por ejemplo. Esto hace que estas técnicas no sean totalmente independientes del experto y

que este siga teniendo que invertir cierto esfuerzo para que se apliquen de forma adecuada, lo que no ocurre con el sistema descrito en este trabajo.

7.3.3 Discriminación de variables de entrada

Otra importante ventaja de la técnica presentada en este trabajo es que permite discriminar aquellas variables que no son importantes para la resolución del problema. Esto es realizado por el proceso evolutivo puesto que, en las RR.NN.AA. resultantes, las variables que no son significativas para resolver el problema no aparecen en la red. Por ejemplo, una red que obtuvo una precisión del 90.85% en la primera iteración del método 5x2cv, en el problema de ionosfera, es la mostrada en la figura 7.9. El problema de la ionosfera se caracteriza, entre otras cosas, por el alto número de entradas que posee (34). Sin embargo, tras echar un rápido vistazo a la figura 7.9, se puede ver que, para resolver el problema con una tasa de precisión alta, no son necesarias tantas variables, sino que el sistema ya se encarga de discriminar aquellas que no son útiles, en este caso hasta quedar un reducido grupo de 7 variables. En esta figura se puede ver cómo la red encontrada por este método, aparte de discriminar el número de variables, es mucho más sencilla que las redes que se utilizarían de forma clásica para resolver este problema. Además, la red encontrada no tiene la limitación arquitectónica de tener la conectividad clásica en la separación en capas, sino que puede llegar a tener cualquier tipo de conectividad.

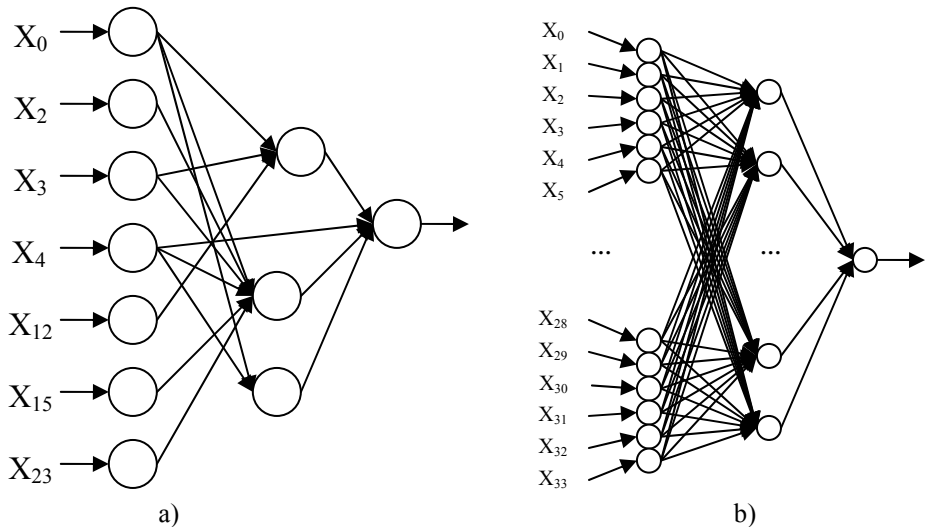


Figura 7.9. *Redes que resuelven el problema de la ionosfera: a) encontrada por este método, y b) red que se utilizaría por el método tradicional*

La reducción del número de variables utilizadas puede verse en la tabla 7.10, en la que se muestra el número medio de variables utilizado por distintas redes al alcanzar las 500.000 evaluaciones del archivo de entrenamiento y el número original de entradas de cada problema. Como puede verse, el número de variables utilizadas es muy bajo en comparación con el número original, lo cual supone una gran ventaja, sobre todo en problemas con un número muy alto de variables, como el de la ionosfera.

Esta tabla también muestra un análisis de componentes principales (*Principal Component Analysis*, PCA) [Jolliffe 1986] de dichos problemas. Este análisis se basa en el estudio de la base de datos desde el punto de vista de las variables de entrada, puesto que generalmente los componentes de dichos vectores de entrada suelen tener bastante correlación, con lo que existe mucha información de entrada redundante. Esta técnica intenta reducir la dimensión de este conjunto de variables de entrada, realizando diversas transformaciones en las mismas, de tal forma que las resultantes (un número menor que el original) sean ortogonales y, por tanto, estén "incorreladas". La importancia de cada variable en PCA está indicada por su varianza. Para realizar el análisis debe de indicarse qué porcentaje de la varianza total se desea que el nuevo modelo explique, es decir, el nuevo conjunto de variables y la transformación utilizada para llegar al mismo.

La tabla 7.10 muestra distintos valores de número de variables para distintos valores de porcentajes de varianza y para cada problema. Estos valores representan el número de variables del modelo resultante después de eliminar del mismo aquellas variables que contribuyen en menos de ese porcentaje a la varianza total del conjunto de datos. Como se puede ver en la tabla, el número de variables utilizadas para cada problema se corresponde con el número resultante de aplicar el análisis de componentes principales tomando un porcentaje de varianza bajo, inferior al 10%, y en ocasiones inferior al 5% (en el caso de flores iris) o incluso del 2% (en el problema de la ionosfera). El sistema discrimina una serie de variables que, por sí mismas, contienen una gran cantidad de información de la base de datos original, con la ventaja frente al análisis de componentes principales de que no se ha realizado ninguna transformación en las variables que se ha tomado.

	Num. Entradas	Entradas usadas	PCA				
			1%	2%	5%	10%	15%
Cáncer de mama	9	5.775	8	8	7	3	3
Iris	4	3.15	3	3	2	2	1
Enfermedades coronarias	13	6.175	11	9	7	4	2
Ionosfera	34	7.025	15	7	3	2	1

Tabla 7.10. *Comparativa de variables usadas en cada problema*

Esta reducción del número de variables de entrada utilizadas también puede verse gráficamente en las figuras 7.10 a 7.13. Dichas figuras muestran una media de las variables utilizadas en distintas redes durante el proceso de ejecución hasta un máximo de 500.000 evaluaciones del archivo de patrones, en el caso de no utilizar grafos ni el sistema de optimización de variables.

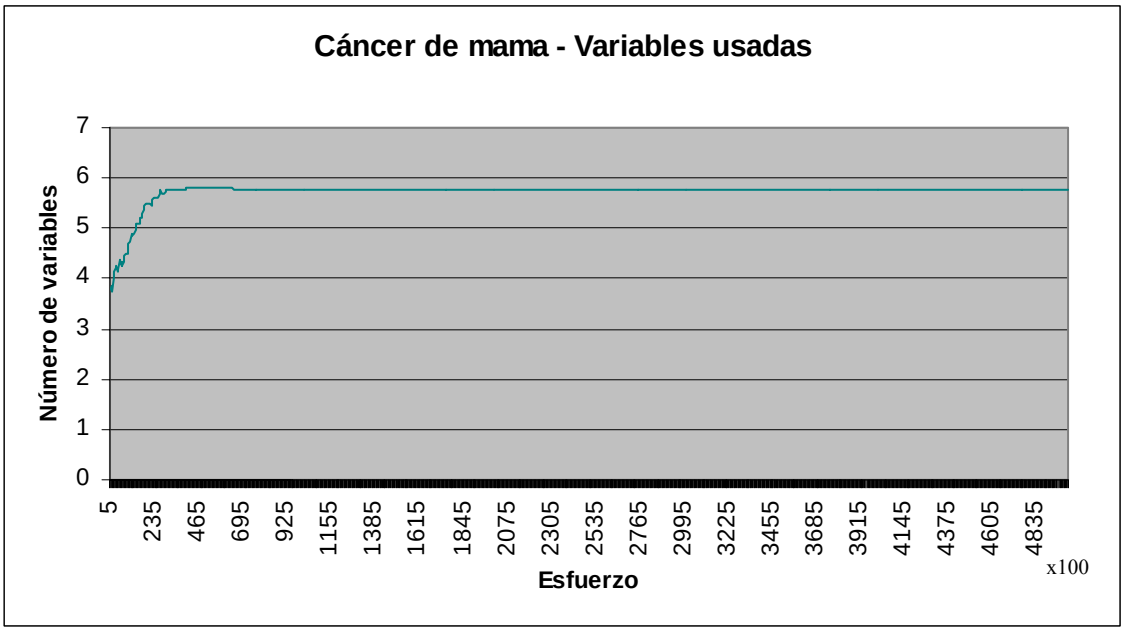


Figura 7.10. *Número de variables usadas en el problema de cáncer de mama a partir de las 9 iniciales*

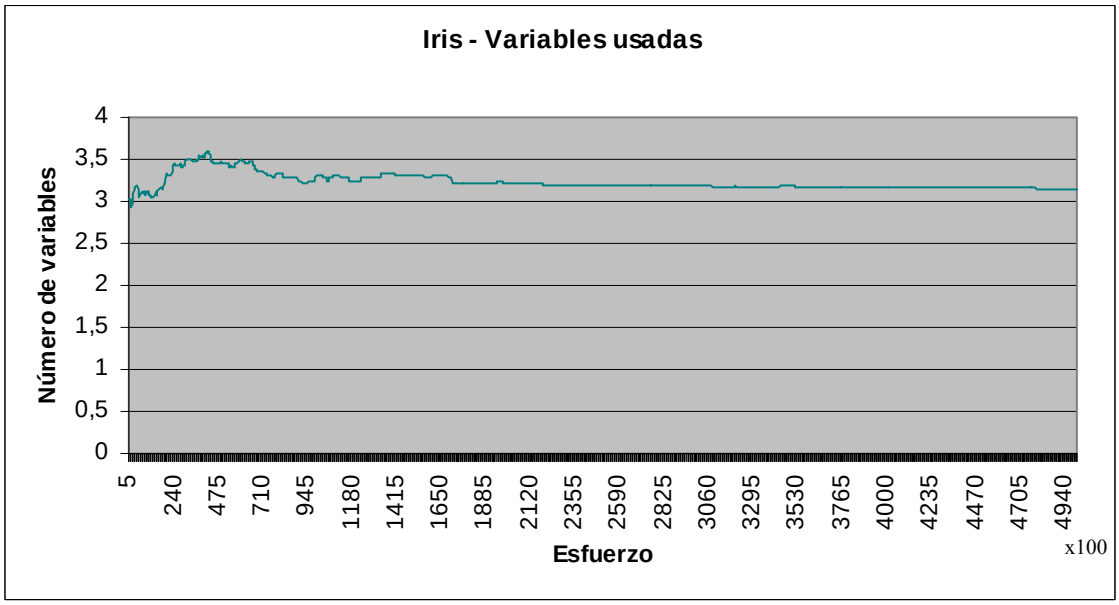


Figura 7.11. *Número de variables usadas en el problema de iris a partir de las 4 iniciales*

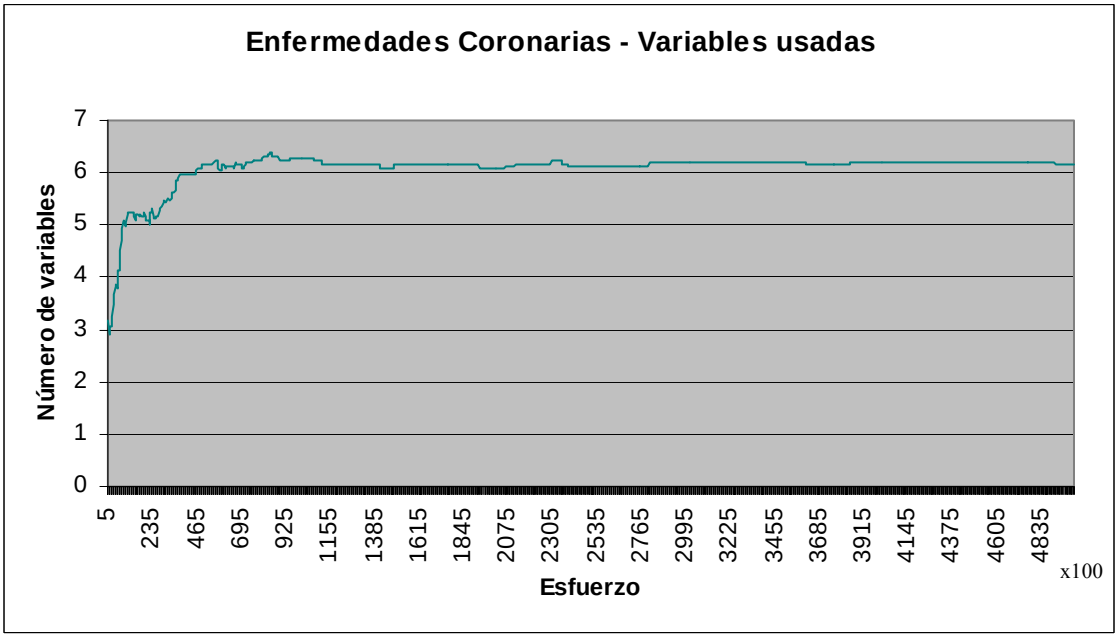


Figura 7.12. *Número de variables en el problema de enfermedades coronarias a partir de las 13 iniciales*

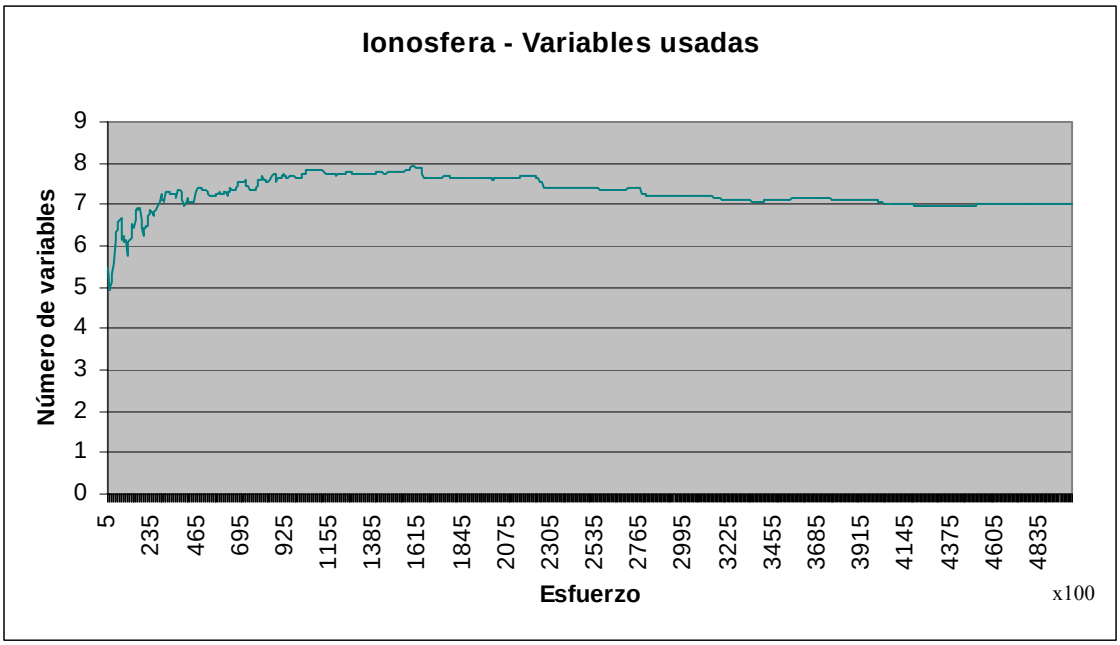


Figura 7.13. *Número de variables usadas en el problema de ionosfera a partir de las 34 iniciales*

Esta discriminación de las variables de entrada constituye una ventaja importante, puesto que puede ser utilizada para extraer conocimiento acerca del problema. Un ejemplo claro de cómo utilizar este sistema para extraer conocimiento es el caso del problema de las flores iris. Una de las razones para aplicar este problema es la situación física de las clases en el espacio de 4 dimensiones formado por las 4 variables de entrada. La figura 7.14 muestra la distribución de las clases en el espacio formado por las variables X_3 y X_4 (longitud y anchura de pétalo). Como ya ha sido estudiado, solamente con esas dos variables se puede lograr una correcta clasificación del 98% de las mismas (es decir, una correcta clasificación de 147 muestras del total de 150) [Duch 2000]. Por lo tanto, son una importante referencia para comparar los resultados gráficamente. Sin embargo, las otras dos variables todavía se necesitan para obtener una precisión mejor.

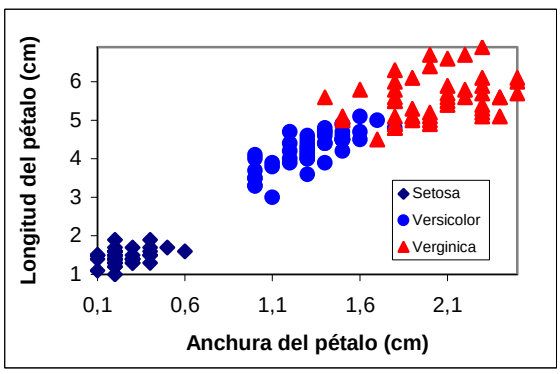


Figura 7.14. *Distribución de las 3 clases*

Este problema ha sido resuelto con diferentes herramientas, entre ellas RR.NN.AA. Entre los resultados obtenidos, se han hallado redes que obtuvieron una precisión del 98,67% (dos errores) con una capa oculta con 6 neuronas en la misma [Martínez 2001]. Mejorando dicha red, se halló otra que alcanzó el mismo valor de precisión, pero con un número menor de neuronas ocultas, solamente 5, y 35 conexiones, como se puede ver en la figura 7.15 [Rabuñal 1999].

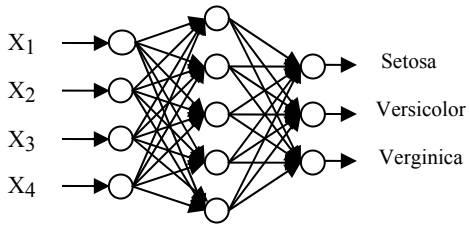


Figura 7.15. *Red que soluciona el problema de iris*

En la aplicación del sistema aquí descrito para la resolución de este problema, no se ha dividido el archivo de patrones en entrenamiento, validación y test, puesto que, para este estudio, sólo interesan los resultados de entrenamiento, para saber qué porcentaje de los patrones es posible clasificar correctamente. Una vez ejecutado el sistema, se han obtenido diversas redes [Rivero 2007]. Un resumen de las mismas puede verse en la tabla 7.11.

Método	Aciertos	Precisión	Neuronas ocultas	Conexiones	Figura
[Rabuñal 1999]	148	98.66 %	5	35	Fig. 7.15
Aquí propuesta	149	99.33 %	3	15	Fig. 7.16 b)
	148	98.66 %	1	11	Fig. 7.17 a), b)
	147	98 %	1	9	Fig. 7.18 a)
	146	97.33 %	1	10	Fig. 7.19 b)
	145	96.66 %	1	10	Fig. 7.20 a)

Tabla 7.11. *Comparativa de arquitecturas encontradas*

Como se puede ver en esta tabla, las arquitecturas de las redes que se han encontrado son muy sencillas, con un número muy bajo de neuronas y conexiones, mejorando de esta forma los resultados encontrados en trabajos previos. La precisión obtenida también ha sido mejorada, llegando a un éxito del 99.33% (149 clasificaciones correctas).

La figura 7.16 muestra dos redes con esta precisión (99.33%). La red b) es la que se indica en la tabla 7.11, con 3 neuronas ocultas y 15 conexiones. La red a) es un poco más complicada, con más conexiones. Sin embargo, esta red tiene una característica

importante: la entrada X_1 (longitud del sépalo) no se utiliza. Consecuentemente, puede concluirse que esta entrada no es útil para clasificar correctamente 149 de los 150 patrones.

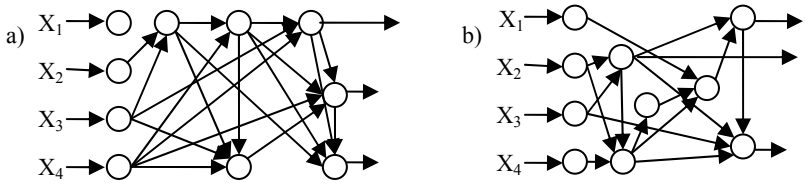


Figura 7.16. *Redes que obtienen una precisión del 99.33% (1 fallo)*

La figura 7.17 muestra otras dos redes que resuelven este problema con una precisión del 98.66%; es decir, producen dos fallos. Ambas redes muestras que la variable X1 (longitud del sépalo) no es necesaria para realizar esta clasificación. Estas redes son mucho más sencillas que las obtenidas por otros métodos [Martinez 2001] [Rabuñal 1999], presentando la misma precisión.

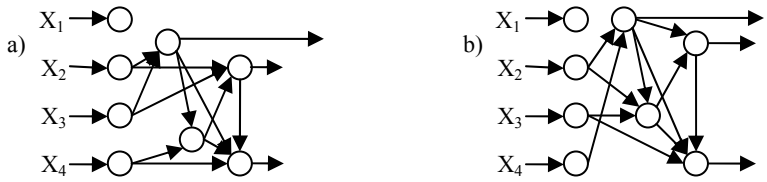


Figura 7.17. *Redes que obtienen una precisión del 98.66% (2 fallos)*

La figura 7.18 muestra tres redes distintas que tienen una precisión del 98%. Como ya ha sido estudiado, esta precisión puede encontrarse con solamente dos entradas [Duch 2000]. Esto también se muestra en esta figura, puesto que la red c) no necesita ni X_1 ni X_2 (longitudes y anchuras de sépalos) para conseguir una correcta clasificación del 98%. Las redes a) y b) son ejemplos de redes sencillas que también tienen esta precisión.

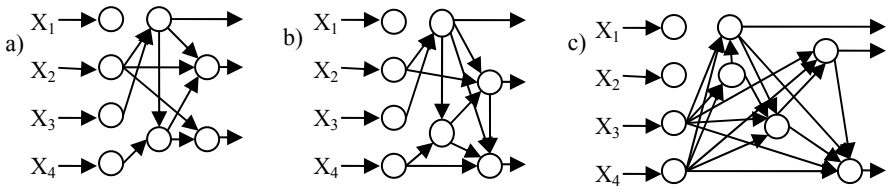


Figura 7.18. *Redes que obtienen una precisión del 98% (3 fallos)*

Algunas redes con una precisión del 97.33% se muestran en la figura 7.19. En dos de ellas, ni X_1 ni X_2 han sido utilizadas para realizar la clasificación.

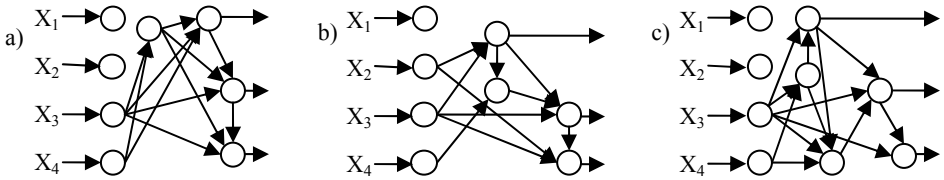


Figura 7.19. *Redes que obtienen una precisión del 97.33% (4 fallos)*

Finalmente, la figura 7.20 muestra tres redes diferentes que realizan la clasificación con una precisión del 96.66%, es decir, fallan en 5 patrones. Lo interesante de las redes b) y c) es que no usan las variables X_1 (longitud de sépalo) ni X_3 (longitud de pétalo) para realizar la clasificación. Como ya ha sido explicado anteriormente, las variables X_3 y X_4 (longitud y anchura de pétalo) se necesitan para clasificar correctamente el 98% de los patrones. Sin embargo, como se puede ver en estas redes, la variable X_3 no es necesaria para clasificar el 96.66% de los datos.

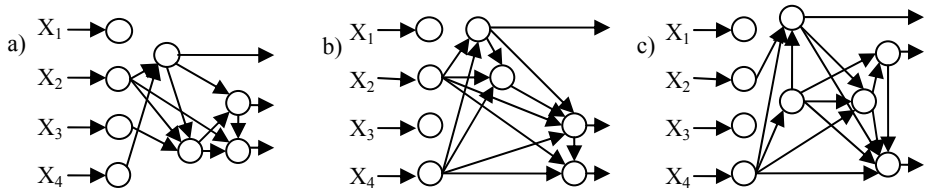


Figura 7.20. *Redes que obtienen una precisión del 96.66% (5 fallos)*

A partir de estos resultados, se pueden extraer las siguientes conclusiones:

- Para clasificar correctamente 149 de los 150 patrones, la variable longitud de sépalo no es necesaria.
- La variable longitud de pétalo, que es necesaria para conseguir una correcta clasificación superior a los 145, no se necesita junto con la variable longitud de sépalo, que tampoco es necesaria) para clasificar correctamente 145 de los 150 patrones.
- Como ya se sabía, y se ha vuelto a encontrar aquí, las variables longitud y anchura de sépalo no son necesarias para clasificar correctamente 147 patrones.

Por lo tanto, puede concluirse que la variable más importante es la anchura del pétalo, que es la única que se necesita en todos los casos. Por otra parte, la variable longitud de sépalo tiene muy poca influencia, puesto que se halló que no era útil en muchas de las redes obtenidas.

7.3.4 Optimización de las redes

Otra ventaja importante de esta técnica es que realiza una optimización de las redes resultantes. Como ya se ha explicado, esta optimización depende del parámetro que penaliza a las redes según el número de neuronas que tienen las mismas. Esta optimización provoca que se generen redes con un número muy bajo de neuronas. Algunos ejemplos de estas redes son los mostrados para el problema de iris en la sección 7.3.3 en las figuras 7.16 a 7.20. En ellas puede verse como las redes obtenidas son muy sencillas, con un número muy bajo de neuronas ocultas (en ocasiones sólo una o dos). Otro ejemplo claro puede verse en la figura 7.9, en la que se muestra una red que resuelve el problema de la ionosfera, en comparación con una red totalmente conectada, que es la que se utilizaría con el método clásico. Un experto que quiera resolver este problema con una red clásica generalmente utilizaría un número alto de neuronas ocultas, dada la gran cantidad de entradas que tiene el mismo. La tabla 7.12 muestra el número de conexiones resultantes de utilizar cierto número de neuronas ocultas, en comparación con las utilizadas por la red encontrada por el método descrito en este trabajo.

Método	Número de neuronas ocultas	Número de conexiones
Método clásico	25	875
	20	700
	15	525
	3	105
Aquí propuesto	3	15

Tabla 7.12. *Número de neuronas y conexiones de una red clásica en comparación con las obtenidas aquí*

Como puede verse en esta tabla, diseñando las redes mediante el método clásico el número de conexiones resultante es muy alto incluso para un número de neuronas ocultas muy bajo. Sin embargo, la figura 7.16 muestra que el problema se ha resuelto con el método aquí descrito con solamente 3 neuronas y 15 conexiones, como también puede verse en la citada tabla.

7.3.5 Arquitecturas obtenidas

Una ventaja adicional de este sistema es que las capas de las redes obtenidas no tienen las limitaciones de las redes construidas por el método tradicional, sino que internamente la red puede tener cualquier tipo de conectividad. En el caso particular de

este trabajo, esta conectividad se ha limitado sólo a obtener redes sin conexiones recurrentes, como se ha explicado en la sección 5.1.1.

Esta capacidad de tener cualquier tipo de conectividad supone una gran ventaja frente a las RR.NN.AA. que poseen una arquitectura tradicional, separada en capas, puesto que éstas presentan grandes dificultades en su análisis. Las redes generadas por este sistema, en cambio, al haber pasado por un proceso de optimización, tienen un conjunto bajo de neuronas, lo cual hace que sea más sencillo analizarlas para descubrir, por ejemplo, qué variables participan en la generación de una salida determinada.

Por ejemplo, en la figura 7.15 se muestra una red tradicional que resuelve el problema de clasificación de flores iris, con una capa oculta con 5 neuronas. Esta red es muy difícil de analizar, dado que existen conexiones de todas las neuronas de una capa a la capa siguiente. Sin embargo, las redes obtenidas que resuelven este problema son mucho más sencillas, y en ellas puede verse fácilmente cómo se tratan los valores de cada entrada para poder devolver una salida determinada. Por ejemplo, en la red 7.17 a) puede verse que para obtener la primera salida sólo se utilizan las entradas X_2 y X_3, y puede analizarse fácilmente su relación. De igual forma, para obtener la segunda salida se utilizan X_2, X_3 y X_4. De esta manera pueden analizarse las redes que se obtienen de una forma más sencilla que las generadas por el método tradicional, y consecuentemente aumentar el conocimiento del sistema creado.

Esta ventaja, unida a la explicada en la sección anterior acerca de la discriminación de las variables de entrada, permite aumentar el conocimiento que se posee del problema.

CAPÍTULO

8 CONCLUSIONES

En este trabajo se presenta una técnica con la que se pueden generar RR.NN.AA. mediante PG. Este sistema ha sido comparado con una gran cantidad de técnicas distintas en las que se usan algoritmos evolutivos para generar y entrenar redes. Entre todas estas técnicas, destacan aquellas en las que se realiza un diseño y entrenamiento de RR.NN.AA. automatizado. En estos casos, y a la vista de los resultados obtenidos en los test 5x2cv, se puede concluir que, desde el punto de vista de la precisión obtenida, el sistema ofrece mejores resultados que los otros métodos excepto en una ocasión, que se produce en la comparación con la técnica de poda. Sin embargo, como se puede ver en la tabla 7.8, esta técnica ofrece los mejores resultados sólo en uno de los problemas, en el de enfermedades coronarias, mientras que en el resto los resultados que ofrece son muy malos en comparación con los obtenidos con el resto de las técnicas (incluida la descrita en este trabajo). Además, esa técnica de poda no es totalmente automatizada, sino que requiere de un experto humano que diseñe la arquitectura inicial de la red y la entrene. Una de las razones por las que esta técnica ofrece mejores resultados es porque las redes que devuelve están dentro de un conjunto de redes con arquitecturas muy limitadas, con lo que, en la práctica, se realiza una búsqueda en un área restringida del espacio de estados, por lo cual resulta más eficiente, aunque más limitada.

La técnica aquí propuesta ofrece varias ventajas importantes sobre el resto:

- No requiere ninguna participación del experto humano en todo el proceso de desarrollo de redes. Tradicionalmente se requería que el experto tomase decisiones en las tareas de diseño, entrenamiento, validación o test de las redes. Sin embargo, en el sistema propuesto aquí todo este proceso ha sido

automatizado. Es más, se ha conseguido un conjunto de parámetros que son aplicables en problemas de muy diversa complejidad, con lo que el experto tampoco es necesario para fijar los parámetros del sistema, si bien es recomendable que se adapte al problema particular que está intentando resolver. El resto de las técnicas descritas en las secciones previas, y las utilizadas en las comparativas, requieren a menudo un trabajo previo del experto, para diseñar una arquitectura inicial, un número de ciclos de ejecución de un algoritmo, o cualquier otro parámetro. Como se ha dicho, utilizando los parámetros recomendados, esta técnica no requiere ningún trabajo por parte del experto.

- El sistema es capaz de generar redes sencillas, con un bajo número de neuronas y conexiones, lo cual evita el problema del sobreentrenamiento de las redes resultantes. Como se ha visto en el ejemplo descrito e ilustrado en la figura 7.16, se ha conseguido una red con solamente 3 neuronas ocultas y 15 conexiones, cuando lo normal habría sido utilizar bastantes más neuronas ocultas, lo que ocasionaría tener cientos de conexiones debido a que el desarrollo clásico de RR.NN.AA. suele implicar capas totalmente interconectadas, y el alto número de entradas provocaría una gran cantidad de conexiones de la capa de entrada a la capa siguiente. En el caso particular del problema de la ionosfera, como se puede ver en la tabla 7.12, por muy bajo que sea el número de neuronas de la capa oculta, el número de conexiones es de varios cientos (ya es de 105 con sólo 3 neuronas ocultas), en contraposición con las solamente 15 conexiones encontradas con el método aquí propuesto.
- El sistema es capaz de discriminar las variables de entrada necesarias para resolver el problema con un determinado nivel de éxito. Esto aporta nuevo conocimiento acerca del problema que se está resolviendo y de cómo están relacionadas las variables del mismo, como se ha mostrado en los ejemplos de los problemas de la ionosfera y las flores iris. En el ejemplo de la ionosfera, se ha reducido el número de entradas necesarias para resolver el problema a 7 de las 34 iniciales. Por su parte, el problema de clasificación de flores iris ha sido sometido a un estudio más extenso y, como consecuencia, se ha generado nuevo conocimiento acerca de las variables de entrada y de su impacto a la hora de clasificar correctamente la base de datos con un determinado nivel de precisión.

- Los resultados obtenidos indican que los problemas que se han planteado se han resuelto de forma satisfactoria. Es más, los valores de precisión expuestos en el capítulo 7 indican que la herramienta descrita en este trabajo ofrece de media mejores resultados que el resto de técnicas. La tabla 7.9 ofrece una visión de una comparativa de esta técnica con el resto de herramientas de desarrollo automatizado de RR.NN.AA., en la que se puede ver que, de forma genérica, los resultados ofrecidos por esta son mejores que los ofrecidos por otras técnicas.
- Por último, esta herramienta realiza la búsqueda en el espacio de estados sin poner limitaciones a las arquitecturas que pueden tener las redes. Esto supone una ventaja importante frente al resto de técnicas, puesto que, a menudo, estas desarrollan redes dentro de un conjunto muy limitado de arquitecturas (como por ejemplo, en el ya descrito caso de la técnica de búsqueda de parámetros). En cambio, la herramienta aquí presentada, al no tener esta limitación, puede encontrar redes que pueden llegar a ofrecer mejores resultados, o con una arquitectura más optimizada (menor número de neuronas y conexiones) que el resto de herramientas.

CAPÍTULO

9 TRABAJO FUTURO

El sistema desarrollado en este trabajo ha demostrado ofrecer buenos resultados. A partir de los resultados ofrecidos, el trabajo de investigación podría continuarse en diferentes direcciones.

Un trabajo interesante es probar este sistema pero sin utilizar cruces, es decir, solamente con mutación. Recientes trabajos [Davoian 2006] indican que los algoritmos evolutivos basados en mutaciones, lo que se denomina Programación Evolutiva (PE), muestran un mejor comportamiento en el mundo del entrenamiento de RR.NN.AA. que las técnicas clásicas que utilizan el operador de cruce. De esta forma, se elimina el problema de la permutación, que produce efectos nocivos al realizar el cruce, y que también parece estar presente en el sistema descrito en ese trabajo. Además, dada la diferencia entre las formas de codificación, en forma de árbol y grafo, adoptadas en este trabajo, es de esperar una gran variabilidad en los resultados que se obtienen al utilizar solamente mutación para la evolución de los individuos. La comparación de los resultados que se obtengan con los que ya se tienen constituye una parte fundamental para cuantificar los efectos negativos del cruce.

Otra línea de investigación interesante es utilizar este sistema para generar redes recurrentes, con lo que podría usarse para el procesado de series temporales, con la finalidad de realizar tareas como modelización, predicción, clasificación, etc. Las redes recurrentes tienen una topología más compleja que las redes alimentadas hacia delante, permitiendo conexiones recurrentes desde una neurona hacia otras, o hacia sí misma, y su naturaleza hace que tengan un mayor nivel de representatividad que las redes alimentadas hacia delante.

Como se ha explicado en la sección 5, la modificación del sistema para el desarrollo de redes recurrentes es relativamente sencilla para el caso de utilización de árboles como forma de codificación. En el caso de usar grafos, esto se vuelve una tarea más complicada, puesto que la forma de codificación es más compleja, y es necesario modificar los algoritmos de creación, cruce y mutación de individuos.

CAPÍTULO

10 BIBLIOGRAFÍA

[Aguiar 1998] Aguiar, H.C.; Maciel R. Modeling and optimization of pulp and paper process using neural networks. Comp.Chem.Eng, 1998, Vol. 22, 981-984.

[Aguirre 1999] Aguirre, H. E.; Tanaka, K.; Sugimura, T. Cooperative crossover and mutation operators in genetic algorithms. Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-99, Morgan Kaufmann. Orlando, Florida, 1999

[Alba 1993] Alba E.; Aldana J.F.; Troya J.M. Fully automatic ANN design: A genetic approach. En Proc. Int. Workshop Artificial Neural Networks (IWANN'93), Lecture Notes in Computer Science, Germany: Springer-Verlag, 1993, Vol. 686, 399-404

[Aldrich 1995] Aldrich C.; Deventer J.S. Comparison of different artificial neural nets for the detection and location of gross errors in process systems, Ind.Eng.Chem.Res., 1995, Vol. 34, 216-224

[Altissimi 1998] Altissimi R. Optimal operation of a separation plant using artificial neural networks. Comp.Chem.Eng., 1998, Vol. 22, 939-942

[Andersen 1993] Andersen H.C.; Tsoi A.C. A constructive algorithm for the training of a multilayer perceptron based on the genetic algorithm. Complex syst., 1993, Vol. 7, No. 4, 249-268

[Angeline 1994] Angeline P.J.; Suders G.M.; Pollack J.B., 1994, An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans. Neural Networks, 1994, Vol. 5, 54-65

[Angeline 1996] Angeline, P. J. An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. Genetic Programming 1996: Proceedings of the First Annual Conference GP-96. MIT Press. Stanford University, California, USA, 1996, 21-29

[Babovic 2001a] Babovic, V.; Keijzer, M.; Aguilera, D.R.; Harrington, J. An Evolutionary Approach to Knowledge Induction: Genetic Programming in Hydraulic Engineering.

Proceedings of the World Water & Environmental Resources Congress. Orlando, Florida, USA, 2001

[Babovic 2001b] Babovic, V.; Keijzer, M.; Aguilera, D. R,; Harrington, J. Automatic Discovery of Settling Velocity Ecuations. Technical Report, D2K-0201-1, Danish Hydraulic Institute, 2001

[Baker 1987] Baker, J. E. Reducing Bias and Inefficiency in the Selection Algorithm. Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1987, 14-22

[Baldwin 1896] Baldwin J. A new factor in evolution. Amerian Naturalist 30, 1896, 441-451

[Bartlett 1990] Bartlett P.; Downs T. Training a neural network with a genetic algorithm. Dep. Elect. Eng., Univ Queensland, Australia, Tech. Rep., 1990

[Baxter 1992] Baxter J. The evolution of learning algorithms for artificial neural networks. En D. Green, T. Bossomaier, eds., Complex Systems, IOS Press, Amsterdam, 1992, 313-326

[Belew 1991] Belew R.; McInerney J.; Schraudolph N. Evolving networks: using genetic algorithm with connectioninst learning. Tech. Rep. CS90-174 (Revised, Computer Science and Engr. Dept. (C014), Univ. of California at San Diego, La Jolla, CA 92093, USA, 1991

[Bengio 1990] Bengio Y.; Bengio S. Learning a synaptic learning rule, Tech. Rep. 751, Départment d'Informatique et de Recherche Opérationelle, Université de Montréal, Canada, 1990

[Bengio 1992] Bengio S.; Bengio Y.; Cloutier J.; Gecsei J. On the optimization of a synaptic learning rule. En Preprints of the Conference on Optimality in Artificial and Biological Neural Networks, Univ. of Texas, Dallas, 1992

[Bishop 1995] Bishop, C.M. Neural networks for pattern recognition. New York: Oxford University Press, 1995

[Bloch 1997] Bloch G. Neural intelligent control for a steel plant. IEEE Trans. Neural Networks, 1997, Vol. 8(4), 910-917

[Boers 1995] Boers E.; Borst M.; Sprinkhuizen-Kuyper I. Evolving Artificial Neural Networks using the Baldwin effect. En D.W. Pearson, N.C. Steele, R.F. Albrecht, eds. Artificial Neural Nets and Genetic Algorithms. Proceedings of the International Conference in Alès, France, Springer Verlag Wien New York, 1995, 333-336

[Booker 1982] Booker, L. B. Intelligent Behavior as an Adaptation to the Task of Environment. PhD thesis, University of Michigan, 1982

[Brindle 1981] Brindle, A. Genetic Algorithms for Function Optimization. PhD thesis, University of Alberta, 1981

[Butler 1995] Butler, J. M.; Tsang, E. P. K. EDDIE beats the bookies. Technical Report CSM-259. Computer Science. University of Essex, Colchester CO4 3SQ, UK, 1995

[Cagnoni 2005] Cagnoni S.; Rivero D.; Vanneschi L. A purely evolutionary memetic algorithm as a first step towards symbiotic coevolution. In Proc. of the IEEE Congress on Evolutionary Computation (CEC 2005), Vol. 2, 2005, 1156-1163

[Cangelosi 1994] Cangelosi A.; Nolfi S.; Parisi D. Cell division and migration in a 'genotype' for neural networks. Network-Computation in Neural Systems, 1994, Vol. 5, 497-515

[Cantú-Paz 2005] Cantú-Paz E.; Kamath C. An Empirical Comparison of Combinatios of Evolutionary Algorithms and Neural Networks for Classification Problems, IEEE Transactions on systems, Man and Cybernetics – Part B: Cybernetics, 2005, 915-927

[Castillo 1999] Castillo P.; González J.; Merelo J.; Rivas V.; Romero G.; Prieto A. G-Prop-II: Global Optimization of Multilayer Perceptrons using GAs. En Congress on Evolutionary Computation, ISBN: 0-7803-5536-9, 1999, Volume III, 2022-2027

[Castillo 2000] Castillo P.; Merelo J.; Rivas V.; Romero G.; Prieto A. G-Prop: Global Optimization of Multilayer Perceptrons using GAs, Neurocomputing, 2000, Vol. 35, 1-4

[Castillo 2001] Castillo P.; Castellano J.; Merelo J.; Romero G. Lamarckian Evolution and Balwin Effect in Artificial Neura Netowrks Evolution. 5th Internacional Conference on Artificial Evolution. Bourgogne, October 29-31, 2001

[Castillo 2002] Castillo P.A.; Arenas M.G.; Castillo-Valdivieso J.J.; Merelo J.J.; Prieto A.; Romero G. Artificial Neural Networks Design using Evolutionary Algorithms. En Proceedings of the Seventh World Conference on Soft Computing, 2002

[Chalmers 1990] Chalmers D., The evolution of learning: an experiment in genetic connectionism. En D.S. Touretzky, J.L. Elman, G.E. Hinton, eds., Proceedings of the 1990 Connectionist Models Summer School, Morgan Kaufmann, San Mateo, CA, 1990, 81-90

[Chellapilla 1997] Chellapilla, K. Evolutionary programming with tree mutations: Evolving computer programs without crossover. Genetic Programming 1997: Proceedings of the Second Annual Conference, Morgan Kaufmann, Stanford University, CA, USA, 1997, 431-438

[Cramer 1985] Cramer, M. L. A Representation for the Adaptive Generation of Simple Sequential Programs. Proceedings of an International Conference on Genetic Algorithms and their Applications, Erlbaum, 1985

[Crosher 1993] Crosher D. The artificial evolution of a generalized class of adaptive processes. En X. Yao, ed., Preprints of AI'93 Workshop on Evolutionary Computation, 1993, 18-36

[Darwin 1859] Darwin, C. On the Origin of Species by Means of Natural Selection, 1859

[DasGupta 1992] DasGupta B.; Schnitger G. Efficient approximation with neural networks: A comparison of gate functions. Dep. Comput. Sci., Pennsylvania State Univ., University Park, Tech. Rep., 1992

[Davoian 2006] Davoian, K.; Loppe W. A New Self-Adaptive EP Approach for ANN Weights Training. Enformatika, Transactions on Engineering, Computing and Technology, Vol. 15, 109-114

[Deb 2002] Deb K.; Anand A.; Joshi D. A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary Computation, 2002, Vol. 10, No. 4, 371-395

[DeJong 1975] DeJong, A. K. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis. University of Michigan, 1975.

[Dellaert, 1994] Dellaert F.; Beer R.D. Towards an evolvable model of development for autonomous agent synthesis, En R. Brooks, P. Maes, eds., Proceedings of the Forth Cenference on Artificial Life, Cambridge, MA: MIT Press, 1994

[Demuth 1994] Demuth, H.; Beale M. Neural network toolbox. User's guide, The MathWorks Inc., 1994

[Denoeux 1993] Denoeux T.; Lengellé R. Initializing back propagation networks with prototypes. Neural Networks, Pergamon Press Ltd., 1993, Vol. 6, 351-363

[Dietterich 1998] Dietterich T.G. Approximate statistical tests for comparing supervised classification learning algorithms. Neural Computation, 1998, Vol. 10, No. 7, 1895-1924

[Dodd 1991] Dodd N.; Macfarlane D.; Marland C. Optimization of artificial neural network structure using genetic techniques implemented on multiple transputers. En P. Welch, D. Stiles, T.L. Kunii, A. Bakkers, eds., Proc. Transputing'91, Amsterdam, The Netherlands: IOS, 1991, 687-700

[Dorado 1999] Dorado, J. Modelo de un sistema para la selección automática en dominios complejos, con una estrategia cooperativa, de conjuntos de entrenamiento y arquitecturas ideales de redes de neuronas artificiales utilizando algoritmos genéticos. PhD Thesis, University of A Coruña, 1999

[Duch 2000] Duch, W.; Adamczak, R.; Grabczewski, K. A new methodology of extraction, optimisation and application of crisp and fuzzy logical rules. IEEE Transactions on Neural Networks, 2000, Vol. 11, No. 2

[Fahlman 1988] Fahlman S. Faster-learning variantions of back-propagation: An empirical study. En D.S. Touretzky, G. Hinton, T. Sejnowski, eds., Proceedings of the 1988 Connectionist Models Summer School, Morgan Kaufmann, San Mateo, 1988, 38-51

[Fan 2001] Fan Z.; Hu J.; Seo K.; Goodman E.D.; Rosenberg R.C.; Zhang B. Bond Graph Representation and GP for Automated Analog Filter Design. 2001 Genetic and Evolutionary Computation Conference Late Breaking Papers, 2001, 81-86

[Fan 2002] Fan Z.; Seo K.; Rosenberg R.C.; Hu J.; Goodman E.D. Exploring Multiple Design Topologies Using Genetic Programming And Bond Graphs. GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Springer-Verlag, 2002, 1073-1080

[Fan 2003] Fan Z.; Seo K.; Hu J.; Rosenberg R.C.; Goodman E. D. System-Level Synthesis of MEMS via Genetic Programming and Bond Graphs. Genetic and Evolutionary Computation -- GECCO-2003, 2003, Vol. 2724. 2058-2071

[Fogel 1995] Fogel D.B.; Wasson E.C.; Boughton E.M., Evolving neural networks for detecting breast cancer. Cancer Lett., 1995, Vol. 96, No. 1, 49-53

[Fogel 1997] Fogel D.B.; Wasson E.C.; Porto V.W. A step toward computer-assisted mammography using evolutionary programming and neural networks, Cancer Lett., 1995, Vol. 119, No. 1

[Frean 1990] Frean M., The upstart algorithm: A method for constructing and training feedforward neural networks. Neural Computation, 1990, Vol. 2, No. 2, 198-209

[Freeman 1993] Freeman J. A.; Skapura D. M. Neural Networks, 1993, Ed. Addison-Wesley

[Freitas 1997] Freitas, A. A. A Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction. Genetic Programming 1997: Proceedings of the Second Annual Conference. Morgan-Kauffman. Stanford University, San Francisco, CA, 1997

[Fuchs 1999] Fuchs, M. Large Populations Are Not Always The Best Choice In Genetic Programming. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), Morgan-Kauffman. Orlando, Florida, USA, 1999, 1033-1038

[Fujiki 1987] Fujiki, C.; Dickinson, J. Using the Genetic Algorithm to Generate LISP Source Code to Solve the Prisoner's Dilemma. Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Erlbaum, 1987

[Galván 2004] Galván, E.; Poli R.; Coello C.A. Reusing Code in Genetic Programming. Genetic Programming Proc. 7th European Conference (EuroGP 2004), Coimbra, Portugal, 2004, 359-368

[Garis 1991] Garis H., GenNets: Genetically Programmed neural nets – Using the genetic algorithm to traing neural nets whose inputs and/or outputs vary in time. En Proc. 1991 IEEE Int. Joint Conf. Neural Networks (IJCNN'91 Singapore), 1991, Vol. 2, 1391-1396.

[Gathercole 1997] Gathercole, C.; Ross, P. Small Populations over Many Generations can beat Large Populations over Few Generations in Genetic Programming. En John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, Rick L. Riolo, eds., Genetic Programming 1997: Proceedings of the Second Annual Conference. Stanford University, CA, Morgan Kaufmann , 1997, pp. 111-118

[Goldberg 1989] Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley, 1989

[González 1998] González-García R.; Rico-Martínez R.; Kevrekidis G. Identification of distribuid parameter systems: A neural net based approach, Comp.Chem.Eng, 1998, Vol. 22, 965-968

[Greenwood 1997] Greenwood G.W. Training partially recurrent neural networks using evolutionary strategies. IEEE Trans. Speech Audio Processing, 1997, Vol. 5, 192-194

[Gruau 1992] Gruau, F. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. En D. Whitley, J.D. Schafer, eds., Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN92). Los Alamitos, CA: IEEE Computer Soc., 1992, 55-74

[Gruau 1994] Gruau, F. Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm. PhD thesis. Laboratoire de l'Informatique du Parallilisme. Ecole Normale Supirieure de Lyon, France, 1994

[Guillen 2007] Guillén A.; Rojas I.; González J.; Pomares H.; Herrera L.J.; Paechter B. Boosting the Performance of a Multiobjective Algorithms to design RBFNNs Through Parallelization. Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Proceedings, Part I, Springer, 2007, 85-92

[Hancock 1990] Hancock P.J.B. GANNET: Design of a neural net for face recognition by genetic algorithm. Center for Cognitive and Computational Neuroscience, Dep. Comput. Sci. Psychology, Stirling Univ., Stirling, U.K., Tech. Rep. CCCN-6, 1990

[Harp 1989] Harp S.A.; Samad T.; Guha A. Toward the genetic synthesis of neural networks. En J.D. Schafer, ed., Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications, San Mateo, CA: Morgan Kaufmann, 1989, 360-369

[Harp 1990] Harp S.A.; Samad T.; Guha A. Designing application-specific neural networks using the genetic algorithm. En D. S. Touretzky, ed., Advances in Neural Information Processing Systems 2, San Mateo, CA: Morgan Kaufamnn, 1990, 447-454

[Haykin 1999] Haykin, S. Neural Networks (2nd ed.). Englewood Cliffs, NJ: Prentice Hall, 1999

[Hernández 1993] Hernández P.J. Redes neuronales en la industria química, Ing. Quim., 1993, Vol. 8, 37-41

[Herrera 2004] Herrera F.; Hervás C.; Otero J.; Sánchez L. Un estudio empírico preliminar sobre los tests estadísticos más habituales en el aprendizaje automático. En R. Giraldez, J.C. Riquelme, J.S. Aguilar, eds., Tendencias de la Minería de Datos en España, Red Española de Minería de Datos y Aprendizaje (TIC2002-11124-E), 2004, 403-412

[Hilera 1995] Hilera J.R.; Martínez V.J. Redes neuronales artificiales. Fundamentos, modelos y aplicaciones. Addison-Wesley Iberoamericana S.A., España, 1995

[Hinton 1987] Hinton G.; Nowlan S. How learning can guide evolution. Complex Systems, 1987, Vol. 1, 495-502

[Holland 1975] Holland, J.J., Adaptation in natural and artificial systems. Ann Arbor, MI: University of Michigan Press, 1975

[Hopfield 1982] Hopfield, J.J., Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences of the USA, 1982, Vol. 79, No. 8, 2554-2558

[Houck 1997] Houck C.; Joines J.; Kay M.; Wilson J. Empirical investigation of the benefits of partial Lamarckianism. Evolutionary Computation, 1997, Vol. 5, No. 1, 31-60

[Husbands 1994] Husbands P.; Harvey I.; Cliff D.; Miller G. The use of genetic algorithms for the development of sensorimotor control systems. P. Gaussier and JD Nicoud, eds., From Perception to Action. Los alamitos CA: IEEE Press, 1994

[Hwang 1997] Hwang M.W.; Choi J.Y.; Park J. Evolutionary projection neural networks. En Proc. 1997 IEEE Int. Conf. Evolutionary Computation, ICEC'97, 1997, 667-671

[Janson 1993] Janson D.J.; Frenzel J.F. Training product unit neural networks with genetic algorithms. IEEE Expert, 1993, Vol. 8, 26-33

[Johanson 1997] Johanson, B. Automated fitness raters for GP-music system. Master's thesis. School of Computer Science, University of Birmingham, Birmingham, UK, 1997

[Johanson 1998] Johanson, B.; Poli, R. GP-music: An interactive genetic programming system for music generation with automated fitness raters. Technical Report CSRP-98-13, University of Birmingham, School of Computer Science. 1998

[Jolliffe 1986] Jolliffe, I. T. Principal Component Analysis. New York: Springer-Verlag, 1986

[Kantschik 1999a] Kantschik W.; Dittrich P.; Brameier M.; Banzhaf W. Empirical Analysis of Different Levels of Meta-Evolution. Proceedings of the Congress on Evolutionary Computation, IEEE Press, 1999, Vol. 3. 2086-2093

[Kantschik 1999b] Kantschik W.; Dittrich P.; Brameier M.; Banzhaf W. MetaEvolution in Graph GP. Proceedings of EuroGP'99, LNCS, SpringerVerlag, 1999, Vol. 1598, 15-28

[Kantschik 2002] Kantschik W.; Banzhaf W. Linear-Graph GP - A new GP Structure. Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002, 2002

[Kim 1996] Kim H.; Jung S.; Kim T.; Park K. Fast learning method for backpropagation neural network by evolutionary adaptation of learning rates. Neurocomputing, 1996, Vol. 11, No. 1, 101-106

[Kim 1997] Kim M.; Choi C. A new weight initialization method for the MLP with the BP in multiclass classification problems. Neural Processing Letters, 1997, Vol. 6, 11-23

[Kim 2005] Jung-Hwan Kim; Sung-Soon Choi; Byung-Ro Moon, Normalization for neural network in genetic search. Genetic and Evolutionary Computation Conference, 2005, 1-10

[Kitano 1990] Kitano H. Designing neural networks using genetic algorithms with graph generation system, Complex Systems, 1990, Vol. 4, 461-476

[Kohonen 1988] Kohonen, T. Self-Organization and Associative Memory, Springer-Verlag, New York, second edition, 1988

[Kolen 1990] Kolen J.; Pollack J. Back Propagation is Sensitive to Initial Conditions. Technical Report TR 90-JK-BPSIC. Laboratory for Artificial Intelligence Research, Computer and Information Science Department, 1990

[Kothari 1996] Kothari B.; Paya B.; Esat I. Machinery fault diagnostics using direct encoding graph syntax for optimizing artificial neural network structure. En Proc. 1996 3rd biennial Joint Conf. Engineering Systems Design and Analysis, ESDA, Part 7 (of 9). New York: ASME, 1996, 205-210

[Koza 1990] Koza, J. R. Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems. Technical Report No. STAN-CS-90-314, Computer Sciences Department, Stanford University, 1990

[Koza 1992] Koza, J. R. Genetic Programming: On the Programming of Computers by Jeans of Natural Selection. Cambridge, MA: MIT Press, 1992

[Ku 1997] Ku K.; Mak M. Exploring the effects of Lamarckian and Baldwinian learning in evolving recurrent neural networks. En Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, IEEE, 1997, 159-163

[Lamarck 1809] Lamarck J. Philosophie zoologique, 1809

[Land 1998] Land M. Evolutionary algorithms with local search for combinatorial optimization. PhD Thesis, Computer Science and ENSR. Dept. – Univ. California. San Diego, 1998

[Lee 1993] Lee Y.; Oh S.; Kim M. An Analysis of Premature Sturation in Back-Propagation Learning, Neural Networks, 1993, Vol. 6, 719-728.

[Levin 1993] Levin A.U.; Narendra K.S. Control of nonlinear dynamical systems using neural networks: Controllability and Stabilization, IEEE Trans. Neural Networks, 1993, Vol. 4, No. 2, 192-206

[Liu 1996] Liu Y.; Yao X. Evolutionary design of artificial neural networks. En Proc. 1996 IEEE Int. Conf. Evolutionary Computation (ICEC'96), Nagoya, Japan, 1996, 670-675

[Lovell 1992] Lovell D.R.; Tsoi A.C. The Performance of the Neocognitron with various S-Cell and C-Cell Transfer Functions, Intell. Machines Lab., Dep. Elect. Eng., Univ. Queensland, Tech. Rep., 1002

[Luke 1996] Luke S.; Spector. L. Evolving Graphs and Networks with Edge encoding: Preliminary. Report. En J. Koza,, ed., Late Breaking Papers at the Genetic Programming 1996 Conference (GP96). Stanford: Stanford Bookstore, 1996, 117-124

[Luke 2000] Luke, S. Two Fast Tree-Creation Algorithms for Genetic Programming. IEEE Transactions on Evolutionary Computation, 2000

[Marin 1993] Marin F.J.; Sandoval F. Genetic synthesis of discrete-time recurrent neural network. En Proc. Int. Workshop Artificial Neural Networks (IWANN'93), Lecture Notes in Computer Science, vol. 686. Berlin, Germany: Springer-Verlag, 1993, pp. 179-184

[Marshall 1991] Marshall S. J.; Harrison R.F. Optimization and training of feedforward neural networks by genetic algorithms. En Proceedings of the Second International Conference on Artificial Neural Networks and Genetic Algorithms, Springer-Verlag, 1991, 39-43

[Martinez 2001] Martinez, A.; Goddard, J., Definición de una red neuronal para clasificación por medio de un programa evolutivo. Mexican Journal of Biomedical Engineering, 2001, Vol. 22, 4-11

[McCulloch 1943] McCulloch W.S.; Pitts W. A logical Calculus of Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics, 1943, No. 5, 115-133

[Meert 1998] Meert K.; Rijckaert M. Intelligent modelling in the chemical process industry with neural networks: A case study, Comp.Chem.Eng., 1998, Vol. 22, 587-593

[Menczer 1992] Menczer F.; Parisi D. Evidence of hyperplanes in the genetic learning of neural networks. Biological Cybern., 1992, Vol. 66, 283-289

[Merelo 1993] Merelo J.; Patón M.; Canas A.; Prieto A.; Morán F. Genetic optimization of a multilayer neural network for cluster classification tasks. Neural Network World, 1993, Vol. 3, 175-186

[Merrill 1991] Merrill J.W.L.; Port R.F. Fractally configured neural networks. Neural Networks, 1991, Vol. 4, No. 1, 53-60

[Mertz 2002] Mertz C.J.; Murphy P.M. UCI repository of machine learning databases, 2002, *http://www-old.ics.uci.edu/pub/machine-learning-databases*

[Merz 1997] Merz P.; Freisleben B. Genetic local search for the TSP: New results. En Proceedings of the 1997 IEEE International Conference on Evolutionay Computation, IEEE, 1997, 159-163

[Michalewicz 1996] Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs, Third, Extended Edition, Springer-Verlag, 1996

[Miller 1989] Miller G.F.; Todd P.M.; Hedge S.U. Designing neural networks using genetic algorithms. In Proceedings of the Third International Conference on Genetic algorithms. San Mateo, CA: Morgan Kaufmann, 1999, 379-384

[Montana 1989] Montana D.; David L. Training feed-forward neural networks using genetic algorithms. En Proc. 11th Int. Joint Conf. Artificial Intelligence. San Mateo, CA: Morgan Kaufmann, 1989, 762-767

[Montana 1995] Montana D.J.. Strongly typed genetic programming. Evolutionary Computation, 1995, Vol. 3, No. 2, 199-200

[Nolfi 2000] Nolfi S.; Floreano D. Evolutionary Robotics: The Biology, Intelligence and Technology of Self-Organizing Machines, Cambridge, MA: MIT Press/Bradford Books, 2000

[Nolfi 2002a] Nolfi S.; Parisi, D. Evolution and Learning in neural networks. In M. A. Arbib (Ed.), Handbook of brain theory and neural networks, Second Edition. Cambridge, MA: MIT Press, 2002, 415-418

[Nolfi 2002b] Nolfi S.; Parisi D. Evolution of Artificial Neural Networks. In M. A.Arbib, Handbook of brain theory and neural networks, Second Edition. Cambridge, MA: MIT Press, 2002, 418-421

[Oakley 1993] Oakley, H. Signal filtering and data processing for laser rheometry. Technical report. Institute of Naval Medicine, Portsmouth, UK, 1993

[Oakley 1994a] Oakley, H. Two scientific applications of genetic programming: Stack filters and non-linear equation fitting to chaotic data. Advances in Genetic Programming. MIT Press. 1994, 369-389

[Oakley 1994b] Oakley, H. The application of genetic programming to the investigation of short, noisy, chaotic data series. Evolutionary Computing, Springer-Verlag. Lecture Notes in Computer Science. Leeds, UK. Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour Workshop on Evolutionary Computing. Leeds, UK, 1994

[Oliveira 1999] Oliveira M.; Barreiros J.; Costa E.; Pereira F. LamBaDa: An Artificial Environment to Study the Interaction between Evolution and Learning. En Crongress on Evolutionary Computation, Washington D.C., USA, 1999, Vol. I, 145-152

[Oller 1998] Oller C.A.; Guidici R. Neural Network based approach for optimisation applied to an industrial nylon-6,6 polymerisation process, Comp.Chem.Eng., 1998, Vol. 22, 595-600

[Orchad 1993] Orchad, G. Neural Computing. Research and Applications. Ed. Institute of Physics Publishing, Londres, 1993

[Ozdemir 2001] Ozdemir M.; Embrechts F.; Breneman C.M.; Lockwood L.; Bennett K.P. Feature selection for in-silico drug design using genetic algorithms and neural networks. En IEEE Mountain Workshop on Soft Computing in Industrial Applications, IEEE Press, 2001, 53-57

[Patel 1996] Patel D. Using genetic algorithms to construct a network for finantial prediction. En Proceedings of SPIE: Applications of Artificial Neural Networks in Image Processing, (Bellingham, WA, USA), Society of Photo-Optical Instrumentation Engineers. 1996, 204-213

[Pazos 1996] Pazos A. Redes de Neuronas Artificiales y Algoritmos Genéticos. Ed. Tórculo, 1996

[Pazos 1999] Pazos A.; Dorado J.; Santos A.; Rabuñal J.R.; Pedreira N. Training of Recurrent ANN with Time Decreased Activation by GA to the Forecast in Dynamic Problems, Proceedings of the 5th International Conference on Information Systems, Analysis and Synthesis (ISAS'99), Orlando, Florida, 1999

[Pearlmutter 1990] Pearlmutter B. A. Dynamic Recurrent Neural Networks. Technical Report CMU-CS, School of Computer Science, Carnegie Mellon Univ. 1990, 88-191

[Pereira 1999] Pereira F. B.; Machado P.; Costa E.; Cardoso A. Graph based crossover-A case study with the busy beaver problem. Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-99. Morgan Kaufmann. Orlando, Florida, USA. 1999, 1149-1155

[Perry 1994] Perry, J. E. The effect of population enrichment in genetic programming. En Proceedings of the 1994 IEEE World Congress on Computational Intelligence. IEEE Press. Orlando, Florida, USA. 1994, pp 456-461

[Pineda 1987] Pineda, F.J. Generalization of backpropagation to recurrent neural networks. Physical Review Letters, 1987, Vol. 59, No. 19, 2229--2232

[Poli 1996] Poli, R. Some Steps Towards a Form of Parallel Distributed Genetic Programming. Proceedings of the First On-line Workshop on Soft Computing, 1996

[Poli 1997] Poli R. Evolution of Graph-like Programs with Parallel Distributed Genetic Programming. Genetic Algorithms: Proceedings of the Seventh International Conference, 1997

[Poli 1998] Poli, R.; Langdon, W. B. On the ability to search the space of programs of standard, one-point and uniform crossover in genetic programming. Technical Report CSRP-98-7. University of Birmingham, School of Computer Science, 1998

[Prechelt 1996] Prechelt L. Early Stopping-But When? Neural Networks: Tricks of the Trade, 1996, 55-69

[Prechelt 1998] Prechelt L. Automatic early stopping using cross validation: qualifying the criteria. Neural Networks 11. 1998, 761-767

[Purves 1994] Purves D. Neural activity and the growth of the brain. Cambridge: Cambridge University Press, 1994

[Rabuñal 1999] Rabuñal, J. R. Entrenamiento de redes de neuronas artificiales mediante algoritmos genéticos. Universidade da Coruña, 1999

[Rabuñal 2003] Rabuñal J.R.; Dorado J.; Puertas J.; Pazos A.; Santos A.; Rivero D. Prediction and Modelling of the Rainfall-Runoff Transformation of a Typical Urban Basin using ANN and GP, Applied Artificial Intelligence, 2003

[Rabuñal 2004] Rabuñal J.R.; Dorado J.; Pazos A.; Pereira J.; Rivero D., A New Approach to the Extraction of ANN Rules and to Their Generalization Capacity Through GP, Neural Computation, 2004, Vol. 16, 1483-1524

[Rabuñal 2005] Rabuñal, J.R., Dorado J., eds. Artificial Neural Networks in Real-Life Applications. Idea Group Inc., 2005

[Randall 1994] Randall, M.C.; Thorne, C. E.; Wild, C. A Standard Comparison of Adaptive Controllers to Solve the Cart Pole Problem. Proceedings of the Second IEEE Australian and New Zealand Conference on Intelligent Information Systems. 1994, 61-65

[Raudys 2007] Raudys S. Evolution of Multi-class Single Layer Perceptron. Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Proceedings, Part II. Springer. 2007. 1-10

[Reed 1993] Reed R. Pruning algorithms – a survey. IEEE Transactions on Neural Networks, 1993, Vol. 4, No. 5, 740-747

[Reynolds 1992] Reynolds, C. W. An evolved, vision-based behavioral model of coordinated group motion. From Animals to Animals (Proceedings of Simulation of Adaptive Behaviour), MIT Press, 1992

[Reynolds 1994] Reynolds, C. W. Evolution of obstacle avoidance behaviour: using noise to promote robust solutions. Advances in Genetic Programming, MIT Press, 1994, 221-241

[Ribert 1994] Ribert A.; Stocker E.; Lecourtier Y.; Ennaji A. Optimizing a Neural Network Architecture with an Adaptive Parameter Genetic Algorithm. Lecture Notes in Computer Science, Springer-Verlag , 1994, Vol. 1240, 527-535

[Ripley 1996] Ripley, B.D. Pattern recognition and neural networks. Cambridge: Cambridge University Press, 1996

[Ritchie 2003] Ritchie M. D.; White B. C.; Parker J. S.; Hahn L. W.; Moore J. H. Optimization of neural network architecture using genetic programming improves detection and

modelling of gene-gene interactions in studies of human diseases. BMC Bioinformatics, 2003, Vol. 3, No. 1

[Rivas 1999] Rivas V.; Merelo J.; Rojas I.; Castillo P.; Carpio J.; Romero G. Evolving 2-Dimensional Fuzzy Logia Controllers, Fuzzy Sets and Systems, 1999

[Rivero 2004] Rivero D.; Rabuñal J.R.; Dorado J.; Pazos A. Using Genetic Programming for Character Discrimination in Damaged Documents. Applications of Evolutionary Computing, EvoWorkshops2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, EvoSTOC, Proceedings, 2004, 349-358

[Rivero 2005] Rivero D.; Rabuñal J.R.; Dorado J.; Pazos A. Time Series Forecast with Anticipation using Genetic Programming. IWANN 2005, 2005, 968-975

[Rivero 2006a] Rivero, D.; Dorado, J.; Rabuñal, J.; Pazos, A.; Pereira, J. Artificial Neural Network Development by means of Genetic Programming with Graph Codification, ENFORMATIKA. Transactions on Engineering, Computing and Technology, World Enformatika Society, 2006, Vol. 15, pp. 209-214

[Rivero 2006b] Rivero D.; Dorado J.; Rabuñal J.; Pazos A. Using Genetic Programmning for Artificial Neural Network Development and Simplification, Proceedings of the 5th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics (CIMMACS'06), WSEAS Press, 2006, 65-71

[Rivero 2007] Rivero D.; Rabuñal J.; Dorado J.; Pazos A.; Automatically Design of ANNs by means of GP for Data Mining tasks: Iris Flower Classification Problem, Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Warsaw, Poland, April 2007, Proceedings, 2007, 276-285

[Ross 1999] Ross B. A lamarckian evolution strategy for genetic algorithms. Complex Coding systems, Boca Raton, FL: CRC Press, 1999, Vol. III, 1-16

[Rumelhart 1986] Rumelhart D.E.; Hinton G.E.; Williams R.J. Learning internal representations by error propagation. En D. E. Rumelhart, J.L. McClelland, eds., Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Cambridge, MA: MIT Press, 1986, Vol. 1, 318-362

[Sarkar 1997] Sarkar M.; Yegnanarayana B. Evolutionary programming-based probabilistic neural networks construction technique. En Proceedings of the 1997 IEEE International Conference on Neural Networks. Part 1 (of 4), (Piscataway, NJ, USA), IEEE Press, 1997, 456-461

[Schwefel 1995] Schwefel H.P. Evolution and Optimum Seeking. New York: Wiley Press, 1995

[Sebald 1998] Sebald A.V.; Chellapilla K. On making problems evolutionary friendly, part I: Evolving the most convenient representations. En V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben, eds., Evolutionary Programming VII: Proc. 7th Annu. Conf.

Evolutionary Programming, Lecture Notes in Computer Science, Berlin, Germany: Springer-Verlag, 1998, Vol. 1447, 271-280

[Segovia 1997] Segovia, J.; Isasi, P. Genetic Programming For Designing Ad Hoc Neural Network Learning Rules. Genetic Programming 1997: Proceedings of the Second Annual Conference. Morgan-Kauffman. Stanford University, San Freancisco, CA, 1997

[Seys 2006] Seys C.W.; Beer, R.D. Effect of Encoding on the Evolvability of an Embodied Neural Network. GECCO 2006 Workshop Proceedings, Workshop on Complexity through Development and Self-Organizing Representations (CODESOAR), 2006

[Sietsma 1991] Sietsma J.; Dow R. J. F. Creating Artificial Neural Networks that generalize. Neural Networks, 1991, Vol. 4, No. 1, 67-79

[Smith 1994] Smith R.E.; Cribbs III H.B. Is a learning classifier system a type of neural network. Evolutionay Computation, 1994, Vol. 2, No. 1

[Smith 1997] Smith R.E., Cribbs I.H.B., Combined biological paradigms: A neural, genetics-based autonomous systems strategy. Robot. Autonomous Syst., 1997, Vol. 22, No. 1, 65-74

[Soule 1997] Soule, T.; Foster, J. A. Code Size and Depth Flows in Genetic Programming. Genetic Programming 1997: Proceedings of the Second Annual Conference. Morgan Kauffmann. San Francisco, CA, 1997, 313-320

[Soule 1998] Soule, T. Code Growth in Genetic Programming. PhD thesis. University of Idaho, 1998

[Spencer 1994] Spencer, G. F. Automatic generation of programs for crawling and walking. Advances in Genetic Programming, MIT Press, 1994, 335-353

[Srinivas 1991] Srinivas M.; Patnaik L.M. Learning neural network weights using genetic algorithms – Improving performance by search-space reduction. En Proc. 1991 IEEE Int. Joint Conf. Neural Networks (IJCNN'91 Singapore), 1991, Vol. 3, 2331-2336

[Stone 1978] Stone M., Cross-validation: A review Matemastische Operationsforschung Statischen, Serie Statistics, 1978, Vol. 9, 127-139

[Stork 1990] Stork D.G.; Walter S.; Burns M.; Jackson B. Preadaptation in neural circuits. En Proc. Int. Joint Conf. Neural Networks, Washington, DC, 1990, Vol. 1, 202-205

[Sutton 1986] Sutton, R.S. Two problems with backpropagation and other steepest-descent learning procedure for networks. En Proc. 8th Annual Conf. Cognitive Science Society. Hillsdale, NJ: Erlbaum, 1986, 823-831

[Tackett 1994] Tackett, W. A.; Carmi, A. The donut problem: Scalability and generalization in genetic programming. Advances in Genetic Programming. MIT Press. 1994, 143-176

[Teller 1996] Teller, A. Evolving Programmers: The Co-evolution of Intelligent Recombination Operators. In P. Angeline, K. Kinnear, eds., Advances in Genetic Programming II, Cambridge: MIT Press, 1996

[Teller 2000] Teller A.; Veloso M. Internal reinforcement in a connectionist genetic programming approach. Artificial Intelligence, 2000, Vol. 120, No. 2, 165-198

[Teller 2000] Teller A.; Veloso M. Efficient Learning Through Evolution: Neural Programming and Internal Reinforcement. Proceedings of the Seventeenth International Conference on Machine Learning, 2000

[Thierens 1996] Thierens D., Non-redundant genetic coding of neural networks. En Proc. 1996 IEEE Int. Conf. Evolutionary Computation, ICEC'96, 1996, 571-575

[Thimm 1995] Thimm G.; Fiesler E. Neural Network initialization. Lecture Notes in Computer Science, Springer-Verlag, 1995, Vol. 930, 535-542

[Topchy 1997] Topchy A.; Lebedko O. Neural network training by means of cooperative evolutionary search. Neural Instruments and Methods in Physics Research, Sectio A: Accelerators, Spectrometers, Detectors and Associated Equipment, 1997, Vol. 389, No. 1-2, 240-241

[Turney 1996] Turney P.; Whitley D.; Anderson R. Special issue on the baldwinian effect, Evolutionary Computation, 1996, Vol. 4, No. 3, 213-329

[Vonk 1995] Vonk E.; Jain L.C.; Johnson R. Using genetic algorithms with grammar encoding to generate neural networks. En Proc. 1995 IEEE Int. Conf. Neural Networks, Part 4 (of 6), 1995, 1928-1931

[Waddington 1942] Waddington C. Canalization of development and the inheritance of acquired characteristics, Nature, 1942, Vol. 3811, 563-565

[Wang 1998] Wang H.; Oh Y.; Yoon E.S. Strategies for modeling and control of nonlinear chemical process using neural networks, Comp.Chem.Eng., 1998, Vol. 22, 823-826

[Wasserman 1989] Wasserman Philip D. Neural Computing. Ed. Van Nostrand Reinhold, New York, 1989

[Wetzel 1983] Wetzel, A. Evaluation of the Effectiveness of Genetic Algorithms in Combinational Optimization. University of Pittsburg (unpublished), 1983

[White 1993] White D.; Ligomenides P. GANNet: A genetic algorithm for optimizing topology and weights in neural network design. En Proc. Int. Workshop Artificial Neural Networks (IWANN'93), Lecture Notes in Computer Science, Berlin, Germany: Springer-Verlag, 1993, Vol, 686, 322-327

[Whitley 1990] Whitley D.; Starkweather T.; Bogart C. Genetic algorithms and neural networks: Optimizing connections and connectivity. Parallel Comput., 1990, Vol. 14, No 3, 347-361

[Whitley 1994] Whitley D.; Gordon V.; Mathias K. Lamarckian Evolution, The Baldwin Effect and Function Optimization. En Y. Davidor, H.P. Schwefel, R. Manner, eds., Parallel Problem Solving from Nature-PPSN III, Springer-Verlag, 1994, 6-15

[William 1994] William Wei W. S. Time Series Analysis. Univariate and Multivariate Methods, ed. Addison Wesley, California, Redwood, 1994

[Yan 1997] Yan W.; Zhu Z.; Hu R., Hybrid genetic/BP algorithm and its application for radar/target classification. En Proc. 1997 IEEE National Aerospace and Electronics Conf., NAECON. Part 2 (of 2), 1997, 981-984

[Yang 1998] Yang J.; Honavar V. Feature subset selection using a genetic algorithm, IEEE Intelligent systems, 1998, Vol. 13, 44-49

[Yao 1995] Yao X.; Shi Y. A preliminary study on designing artificial neural networks using co-evolution. En Proc. IEEE Singapore Int. Conf. Intelligence Control and Instrumentation, Singapore, 1995, 149-154

[Yao 1997] Yao X.; Liu Y. EPNet for chaotic time-series prediction. En X. Yao, J.-H. Kim, T. Furuhashi, eds., Select. Papers 1st Asia-Pacific Conf. Simulated Evolution and Learning (SEAL'96), Lecture Notes in Artificial Intelligence, Berlin, Germany: Springer-Verlag, 1997, Vol. 1285, 146-156

[Yao 1998] Yao X.; Liu Y., Toward designing artificial neural networks by evolution, Appl. Math. Computation, 1998, Vol. 91, No. 1, pp. 83-90

[Yao 1999] Yao X. Evolving artificial neural networks. Proceedings of the IEEE, 1999, Vol. 87, No. 9, 1423-1447

[Zomorodian 1995] Zomorodian, A. Context-free Language Induction by Evolution of Deterministic Push-down Automata Using Genetic Programming. In Working Notes of the Genetic Programming Symposium, AAAI-95. AAAI Press., 1995