Universidad de Granada
E.T.S. Ingeniería Informática

Sistemas Evolutivos para Entrenamiento y Optimización de Modelos Neuronales Recurrentes Dinámicos. Aplicación en Modelado y Predicción de series de datos.

TESIS DOCTORAL

Manuel Pegalajar Cuéllar
Depto. Ciencias de Computación e Inteligencia Artificial
Universidad de Granada
E-Mail: manupc@decsai.ugr.es

Directores de tesis:

Miguel Delgado Calvo-Flores
María del Carmen Pegalajar Jiménez

Editor: Editorial de la Universidad de Granada
Autor: Manuel Pegalajar Cuellar
D.L.: Gr. 1308 - 2006
ISBN: 978-84-338-4019-6

Sistemas Evolutivos para Entrenamiento y Optimización de Modelos Neuronales Recurrentes Dinámicos. Aplicación en Modelado y Predicción de series de datos.

Manuel Pegalajar Cuéllar

La memoria titulada **"Sistemas Evolutivos para Entrenamiento y Optimización de Modelos Neuronales Recurrentes Dinámicos: Aplicación en Modelado y Predicción de Series de Datos"**, que presenta D. Manuel Pegalajar Cuéllar para optar al grado de Doctor, ha sido realizada en elDepartamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, bajo la dirección del Doctor D. Miguel Delgado Calvo-Flores y de la Doctora María del Carmen Pegalajar Jiménez.

Granada, 23 de Mayo de 2006.

El Doctorando	El Director	El Director
Fdo. M. Pegalajar	Fdo. M. Delgado	Fdo. M.C. Pegalajar

AGRADECIMIENTOS

Desde que comencé mi carrera laboral en el departamento de Ciencias de la Computación e Inteligencia Artificial, he conocido a mucha gente a la que le tengo que estar muy agradecido. Por regla general puedo decir que todos los miembros de esta casa, de una forma u otra, me han acompañado estos años, ayudándome a desarrollarme tanto a nivel laboral como a nivel personal. No voy a nombrarlos a todos, pero sí me gustaría hacer una mención especial a Ignacio y Nico.

Por otra parte, tengo que agradecerle a toda mi familia la gran paciencia que han tenido conmigo, siempre han estado ahí tanto en los momentos buenos y en los momentos malos. También a mis amigos, los cuales han tenido el privilegio de "tomarse unas vacaciones sin verme" durante bastante tiempo durante el desarrollo de la tesis. Gracias también por aguantar mis quejas insufribles.

Y, para terminar, dicen los maestros del espectáculo que lo mejor hay que dejarlo para el final. Dedico mis mejores y más sinceros agradecimientos a mis directores de tesis, Mª del Carmen y Miguel, los cuales han sido los que más paciencia han tenido que tener.

Gracias a todos

Sistemas Evolutivos para Entrenamiento y Optimización de Modelos Neuronales Recurrentes Dinámicos. Aplicación en Modelado y Predicción de series de datos.

Manuel Pegalajar Cuéllar

Indice

Introducción

1. Planteamiento.

El objetivo de esta tesis es el diseño de sistemas inteligentes para modelado y predicción de conjuntos de datos que presenten ciertas dependencias temporales. Nuestra propuesta se centrará en la utilización de modelos bioinspirados (redes neuronales [DAN01][DAY90], algoritmos evolutivos [BAC96][COE02], etc.), los cuales adaptaremos para llevar a cabo nuestro objetivo.

La finalidad del análisis de series de datos es el conocimiento de su patrón de comportamiento de modo que se pueda prever la evolución de los mismos, bajo el supuesto de que las condiciones que afectan a la serie no varíen con respecto a los datos anteriores y actuales, a lo largo del tiempo [BRO86][VIÑ01]. No obstante, si conociendo los valores anteriores y actuales de la serie se pudiesen predecir los valores futuros sin error, estaríamos frente a un proceso determinista y su estudio no tendría ningún interés. El hecho que motiva el estudio de las series temporales es la aparición de fenómenos de naturaleza aleatoria, parcial o total, los cuales hacen que la predicción de los valores futuros de la serie sólo pueda realizarse mediante métodos de aproximación.

La mayoría de las herramientas utilizadas hasta el momento, para análisis y predicción de series de datos, están basadas en técnicas de regresión estadística lineales (metodología Box-Jenkins) [BRA99], métodos de alisado, y algunos métodos no lineales [TER94], principalmente regresiones, modelos de Markov, redes neuronales, etc [POL99]. Gran parte de la bibliografía actual utiliza redes neuronales para predicción de series de datos, siendo las redes recurrentes [AUS99], RBF y Perceptrón Multicapa algunas de las más utilizadas [DAV99]. Las ventajas introducidas por la utilización de modelos neuronales para predicción de series temporales son numerosas: robustez frente al ruido, adaptabilidad, condición de aproximadores universales, no linearidad, etc.

Sin embargo, existen ciertos inconvenientes a la hora de modelar una serie temporal mediante redes neuronales feedforward: Encontrar la estructura de entrada adecuada para la red es una tarea compleja, debido al desconocimiento de las relaciones existentes en el conjunto de

datos. Esta tarea requiere múltiples experimentos y tests hasta dar con la estructura adecuada para la red. Además, si la serie temporal presenta un comportamiento dinámico; es decir, tiene diferentes comportamientos dependientes del tiempo, la utilización de redes neuronales feedforward se vuelve ineficaz, de modo que se debe utilizar un modelo de red para cada tipo de comportamiento distinto que se presente.

En nuestro trabajo abordaremos el problema del modelado y predicción de series de datos utilizando redes neuronales recurrentes dinámicas (RNRD) [DAN01]. Las RNRD son modelos de redes neuronales artificiales que incorporan cierta realimentación en su estructura. Esta característica hace que sean una herramienta adecuada para tratamiento de datos que se presentan con un cierto orden impuesto a lo largo del tiempo.

Sin embargo, el entrenamiento de este tipo de modelos es una tarea compleja, debido a la dificultad introducida por el tratamiento de la recurrencia [BEN94]. Los algoritmos clásicos de entrenamiento para este tipo de redes, principalmente basados en el gradiente [JAE02], presentan ciertas limitaciones debido a que tienen cierta tendencia a quedar atrapados en óptimos locales, dificultando así el correcto aprendizaje de la red neuronal. Nuestro trabajo se centrará, en parte, en el desarrollo de técnicas que ayuden a solventar estos problemas, abordándolos desde diversas perspectivas:

- Por una parte, los algoritmos evolutivos [BAC96] son algoritmos de búsqueda, optimización y aprendizaje, que han sido aplicados a un gran número de problemas, obteniendo resultados adecuados. Concretamente, en el campo del entrenamiento de redes neuronales, la utilización de algoritmos genéticos ha producido mejores resultados que los obtenidos por algoritmos clásicos de entrenamiento basados en el gradiente. La adaptación y estudio de diversas técnicas evolutivas, aplicadas al entrenamiento de RNRD en problemas de modelado y predicción de series de datos, constituirá la base para el desarrollo de la tesis. Partiendo de este punto, desarrollaremos nuevas técnicas evolutivas que puedan competir con los algoritmos de entrenamiento existentes, de modo que se mejore el aprendizaje de las RNRD.
- En segundo lugar, algunas técnicas clásicas que mejoran la potencia de los algoritmos basados en el gradiente, tales como ciertos métodos Quasi-Newton y otras técnicas de programación no lineal [SCH95], han probado ser herramientas eficaces para solucionar

problemas de optimización. Mientras que los algoritmos evolutivos realizan una búsqueda heurística a lo largo del espacio de soluciones, los algoritmos Quasi-Newton dirigen la búsqueda basándose en la información proporcionada por el gradiente y la segunda derivada. Por tanto, la utilización de técnicas clásicas de optimización no lineal de mayor potencia que los algoritmos basados en el gradiente, pueden ser adaptadas de modo que el entrenamiento de las RNRD se realice de forma adecuada. Complementando la eficacia de este tipo de técnicas con algoritmos evolutivos, pretendemos obtener nuevas técnicas híbridas que mejoren los resultados actuales en cuanto al aprendizaje de redes neuronales.

- Por último, otro aspecto clave en el diseño de redes neuronales, es hallar la estructura de la misma que permita un correcto aprendizaje, sin perder la capacidad de generalización de la red por sobreentrenamiento. El desarrollo de esta etapa requiere un gran número de experimentos y tests, siendo abordada en la actualidad principalmente por métodos de ensayo y error. En nuestro trabajo, el desarrollo de técnicas de optimización multiobjetivo [COE99][COE00][COE02] nos permitirá entrenar una red neuronal y conseguir la estructura óptima de la misma, de forma simultánea

2. Antecedentes.

2.1. Redes neuronales recurrentes dinámicas. El problema del entrenamiento.

Las redes neuronales artificiales son modelos matemáticos bioinspirados que han sido ampliamente utilizados en la resolución de problemas complejos, principalmente debido a sus características para modelar la no linearidad, no estacionariedad y su robustez frente al ruido, entre otras [DAY90][FRE91][LIN92]. Los modelos de redes neuronales más utilizados han sido principalmente las redes neuronales *feedforward*. No obstante, dependiendo de las características del problema a resolver, la selección del modelo adecuado de red neuronal a utilizar es un aspecto clave para obtener resultados exitosos. Las redes neuronales artificiales *recurrentes dinámicas* se obtienen a partir de modelos de redes *feedforward*, a las que se les han introducido conexiones retroalimentadas, formando ciclos dentro de la estructura de la red [DAN01][MED01].

La característica principal de este tipo de redes es la capacidad de procesamiento de patrones dinámico-temporales. La recurrencia de la red permite guardar en su memoria interna información acerca de la estructura temporal de los datos, lo cual hace que estos modelos de redes neuronales sean adecuados para aprender patrones temporales difíciles de modelar. Además, si los patrones de datos presentan un comportamiento dinámico; es decir, las relaciones entre los datos cambian con el tiempo, una red recurrente dinámica presenta la ventaja de poder detectar estos cambios y adaptarse al nuevo problema.

Al proceso mediante el cual una red neuronal aprende el comportamiento correcto para la resolución de un problema se le conoce como *entrenamiento*. Tradicionalmente, hay dos formas de realizar el entrenamiento de una red neuronal:

- *Modo secuencial*. Según este modelo, los pesos de la red se modifican tras la presentación de cada par entrada/salida.
- *Modo por lotes*. Según este modelo, los pesos de la red se modifican tras la presentación del conjunto completo de patrones de entrenamiento.

Tradicionalmente, el entrenamiento de una red neuronal recurrente dinámica se ha realizado utilizando métodos de optimización basados en el gradiente. Ejemplos de estas técnicas son los algoritmos *BackPropagation Through the Time* (BPTT) y *Real Time Recurrent Learning* (RTRL). Estos algoritmos presentan ciertas limitaciones, en el sentido de que la propagación del error y el cálculo del gradiente a través de la recurrencia es una tarea compleja, dando como resultado un entrenamiento poco adecuado de la red en problemas donde los datos presentan dependencias temporales complejas [BEN94]. Como alternativas a este tipo de técnicas, se han propuesto diversas metodologías, tales como métodos Quasi-Newton, algoritmos heurísticos (búsqueda tabú, enfriamiento simulado...), y ciertos modelos de algoritmos bioinspirados, principalmente de tipo genético. Las propuestas de algoritmos de entrenamiento que realizamos en este trabajo están orientadas a la hibridación de diversas técnicas anteriores, basadas en algoritmos evolutivos y técnicas de programación no lineal.

2.2. Series temporales. Modelado y predicción.

Una serie temporal es una secuencia de observaciones realizadas sobre un fenómeno concreto e indexadas en el tiempo [BRA99]. Las series temporales se caracterizan porque su evolución temporal no depende explícitamente de la variable tiempo, sino también de valores de la misma serie, medidos en instantes anteriores al actual. Ocasionalmente, también puede existir dependencia con otras variables temporales externas.

Los modelos clásicos para modelado y predicción de series temporales están basados principalmente en regresiones lineales y el modelado Box-Jenkins [BRO86]. El análisis de series temporales (utilizando la metodología Box-Jenkins), permite encontrar un modelo lineal que aproxima los datos anteriores y actuales de la serie. Sin embargo, este tipo de métodos presentan ciertas limitaciones:

- Sólo son aplicables cuando la serie temporal es estacionaria (su media, varianza y autocorrelación deben ser similares a lo largo del tiempo).
- No permiten modelar relaciones no lineales.
- Es recomendable disponer de, al menos, cincuenta valores de la serie temporal, para poder aproximar los parámetros debidamente.
- Se asume que el valor de los parámetros es constante a lo largo del tiempo.
- El horizonte de predicción no puede ser muy alto en los modelos que contengan componentes de medias móviles, debido a la suposición de que el error en la predicción es cero.

Debido a las limitaciones que presentan estos métodos, es necesario utilizar otros modelos para predicción de series temporales, entre los que están los sistemas difusos, métodos de regresión no lineal, redes neuronales, etc [DAV99][KAA96][THR94][VIÑ01]. Una parte importante de la bibliografía actual utiliza redes neuronales para predicción de series temporales, siendo las redes RBF y Perceptrón Multicapa los modelos *feedforward* de mayor uso. La metodología general utilizada para modelado y predicción de series de datos mediante redes *feedforward* es la siguiente:

1. Tratamiento de datos.
2. Selección de la estructura de las entradas y salidas de lared.
3. Selección de la estructura interna de la red (número de neuronas,capas y conexiones).
4. Entrenamiento.
5. Validación.

El anterior esquema implica un alto número de experimentos y tests hasta obtener la estructura de red adecuada y entrenada. Como problema añadido, siuna serie temporal presenta un comportamiento dinámico y las dependencias entre el conjunto de datos varían con el tiempo, el modelado de la serie se vuelve ineficaz utilizando este tipo de modelos. En la siguiente sección, introducimos los objetivos de la metodología a desarrollar en la tesis, de modo que las limitaciones de los modelos expuestos puedan ser abordadas y solventadasadecuadamente.

3. Objetivos.

Los objetivos de la tesis van dirigidos a realizar un aporte a la comunidad científica de un modo tanto teórico como práctico. Respecto al aporte teórico, nuestro objetivo consiste en el estudio y desarrollo de nuevas técnicas evolutivas que mejoren el entrenamiento de RNRD. Debido al buen comportamiento de los algoritmos genéticos, aplicados en este campo, se presenta un campo de investigación que promete resultados esperanzadores. Adicionalmente, pretendemos hibridar técnicas clásicas de optimización no lineal (algoritmos Quasi-Newton) con algoritmos evolutivos, de modo que se proporcionen algoritmos que posean las ventajas de ambas metodologías. Por último, el desarrollo de técnicas de optimización multiobjetivo nos permitirá obtener tanto la estructura óptima de la red, como la misma red entrenada. La propuesta mantiene las siguientes ventajas:

- Fácil adaptación a diferentes modelos de redes neuronales.

- Solventa el problema de obtención de soluciones locales, dado en los algoritmos clásicos basados en el gradiente y otras técnicas heurísticas, como búsquedatabú y enfriamiento simulado.
- Integración de las etapas de optimización de la estructura y entrenamiento de una RNRD, permitiendo la disminución del número de experimentos a realizar.
- Generalidad para su aplicación en diversos tipos de problemas.

Respecto al aspecto práctico, la metodología desarrollada en la tesis presenta ciertos aspectos de innovación para la resolución de problemas de predicción deseries de datos:

- Capacidad para modelar sistemas donde las relaciones temporalesentre los datos varíen con el tiempo (adaptación a sistemas dinámicos).
- Reducción del número de experimentos en la etapa de selección de la estructura de entradas y salidas de la red neuronal.
- Capacidad para procesar patrones de datos de longitud variable.
- Capacidad para aprender relaciones temporales complejas a largoy corto plazo.

4. Resumen.

La memoria está organizada según el esquema siguiente:

- En el capítulo 1, se introducen los conceptos básicos necesariospara el desarrollo de los capítulos posteriores. Se comienza con una introducción a los modelosde RNRD y los algoritmos clásicos de entrenamiento. Seguidamente, hacemos una breve revisión de los métodos de entrenamiento de naturaleza heurística más utilizadospara entrenar RNRD, haciendo especial énfasis en los algoritmos evolutivos. También estudiamos las bases de los algoritmos multiobjetivo, los cuales utilizaremos posteriormente para entrenar y optimizar simultáneamente RNRD. Por último, el capítulo finaliza mostrando la

aplicación de los diferentes modelos de redes estudiados para elproblema de predicción de series de datos.

- En el capítulo 2, abordamos el problema del entrenamiento de RNRD mediante algoritmos evolutivos. El estudio se centra en la capacidad de entrenamiento de diversos modelos de algoritmos genéticos existentes, y en varias propuestas realizadas para mejorar el equilibrio entre diversidad y convergencia del ciclo evolutivo. Además, modificamos el algoritmo CHC para operar con codificación real y así poder utilizarlo para el entrenamiento de RNRD.

- En el capítulo 3, proponemos un conjunto de algoritmos híbridos, basados en los resultados obtenidos en el capítulo anterior, para el entrenamiento de RNRD. En primer lugar, realizamos una introducción a las técnicas de hibridación que utilizamos. Seguidamente, hacemos un estudio de diferentes algoritmos de programación no lineal de modo que podamos seleccionar un método adecuado para la hibridación.Por último, estudiamos la hibridación de los algoritmos genéticos y CHC modifcado junto con los métodos de programación no lineal, obteniendo algoritmos meméticos.

- El capítulo 4 está dedicado a solventar el problema del entrenamiento y optimización simultáneos de RNRD, utilizando algoritmos evolutivos multiobjetivo. Enprimer lugar, introducimos la asociación genotipo-fenotipo de redes y cromosomas, para poder abordar el problema multiobjetivo planteado. Seguidamente, realizamos una propuesta de operadores de cruce y mutación, adaptados para entrenar y optimizar la estructura interna de las RNRD. A continuación, estudiamos la eficacia de diferentes modelos multiobjetivo para resolver el problema planteado. Por último, hacemosuna propuesta de algoritmos multiobjetivo híbridos para mejorar el entrenamiento y optimización simultáneos de las RNRD.

- El capítulo 5 muestra los resultados obtenidos por los diferentes métodos estudiados, aplicando los diferentes modelos de RNRD en problemas de predicción de series de datos. En primer lugar, comparamos los métodos desarrollados con otros métodos existentes, utilizando series de datos benchmark. Por último, mostramos la aplicación de estos modelos sobre series temporales de índole económica.

- A continuación exponemos algunas consideraciones y conclusiones finales obtenidas, resumiendo los logros alcanzados expuestos a lo largo del documento, y los futuros trabajos de investigación que han surgido durante el desarrollo del mismo.

- La memoria concluye con la bibliografía utilizada para poder desarrollar la investigación realizada.

Capítulo 1

Fundamentos

El propósito de este capítulo es introducir los conceptos elementales y técnicas necesarias para el posterior desarrollo de la investigación realizada. La estructura del capítulo es la siguiente:

- En primer lugar, realizaremos una introducción a las redes neuronales. Haremos especial énfasis en las redes neuronales cuya estructura presenta un grado de realimentación entre neuronas, detallando los modelos más comunes de redes neuronales recurrentes dinámicas (recurrente completa, Elman y Jordan). También comentaremos las características que hacen de estos modelos nuestro principal motivo de estudio.

- La sección segunda explica los algoritmos clásicos de entrenamiento para cada modelo de red, basados en métodos del gradiente. También presentaremos sus limitaciones, sirviendo como motivación para el desarrollo de la investigación realizada en los siguientes capítulos.

- Seguidamente, la sección tercera hace una breve revisión de los métodos de entrenamiento de naturaleza heurística más utilizados, para entrenamiento de redes neuronales recurrentes dinámicas. Haremos especial énfasis en los algoritmos bioinspirados, ya que éstos servirán como base para construir algoritmos de entrenamiento híbridos en el capítulo 3.

- La sección cuarta está dedicada a introducir las bases de los algoritmos evolutivos multi-objetivo. En el capítulo 4 haremos uso de este tipo de técnicas para entrenar y optimizar la estructura topológica de una red neuronal recurrente dinámica de forma simultánea.

- Por último, la sección quinta se centra en la aplicación de redes neuronales recurrentes dinámicas para predicción de series de datos, hacia la cual se orientan los experimentos realizados en los capítulos posteriores.

1. Generalidades sobre Redes Neuronales.

Las redes neuronales son modelos matemáticos, inspirados en el funcionamiento del cerebro humano [MEH92][PAT92][SMA96][VEE95]. Se puede considerar que los modelos de redes neuronales son máquinas numéricas para tareas cognitivas como aprendizaje u optimización. El origen de las redes neuronales artificiales se remonta a principios de los años 40, cuando los avances producidos en la biología permitieron a McCulloch y Pitts proponer una teoría general de procesamiento de información basada en redes de elementos de decisión o conmutadores binarios, a los que llamaron neuronas. No obstante, estos elementos eran mucho más simples que sus equivalentes biológicos.

En 1949, Donald Hebb propuso el primer algoritmo de aprendizaje para algunos modelos neuronales básicos existentes. Más tarde, Rosenblatt concibió la idea del perceptrón de una única capa (1958). En esta época, también fue notable el trabajo de Widrow y Hoff para formular reglas de aprendizaje, aplicables al modelo del perceptrón de Rosenblatt, mediante el algoritmo Adaline (1960). En 1969, Minsky y Papert realizaron un estudio de las posibilidades del perceptrón, señalando sus limitaciones. A partir de entonces, hubo un periodo de tiempo de aproximadamente entre 15 y 20 años en el que la redes neuronales artificiales perdieron popularidad, y su desarrollo quedó estancado. No obstante, durante este periodo hubo algunas investigaciones de gran relevancia: Considerando esencialmente modelos de redes neuronales recurrentes, destacan los trabajos de Hopfield y Kohonen [GUP03], con el desarrollo de la red de Hopfield y los mapas autoorganizativos, respectivamente.

A finales de la década de los 80 y principios de los 90, surgieron nuevos modelos de redes neuronales artificiales recurrentes, cuya principal característica es el procesamiento de patrones de datos de longitud variable. Destacan los trabajos de Jordan y posteriormente de

Elman [DAN01][ELM90], dando lugar a los modelos hoy conocidos como redes neuronales recurrentes dinámicas de Jordan y Elman, respectivamente.

Posteriormente, desde finales de la década de los 90 hasta la actualidad, han surgido nuevos modelos de redes neuronales recurrentes [DAN01][GUP03][HAY99][MED01], bien con el fin de emular alguna característica del cerebro humano, bien con el propósito de solucionar algún problema práctico. Como ejemplos, podemos citar los modelos LSTM [GER02][GER03], algunas arquitecturas de red basadas en modelos probabilísticos, y las máquinas de Boltzman [HAY99]. Adicionalmente, otras líneas de investigación han desembocado en el desarrollo de modelos híbridos que toman características de distintos tipos de redes, como por ejemplo las redes FIR [WAN95][PEG03], los perceptrones recurrentes o las redes RBF recurrentes [ZEM03].

Aunque las redes neuronales surgieron como un modelo para tratar de imitar el funcionamiento del cerebro humano, hoy en dia que ningún modelo de red conocido ha sido capaz de duplicarlo. Sin embargo, la aplicación de diferentes modelos de redes a diversos problemas (clasificación, control, reconocimiento de patrones, etc) [DAY90][FOO02] [FRE91][LIN92][MED01][THO91], con resultados muy prometedores, ha propiciado la expansión y la publicidad que esta herramienta tiene en la actualidad.

1.1. Redes neuronales recurrentes: Tipos de recurrencia.

La estructura topológica y el funcionamiento interno de una red neuronal artificial permiten la clasificación de los diferentes modelos de redes existentes. Considerando el tipo de conexión entre las neuronas de la red, podemos encontrar cuatro tipos de conexiones claramente diferenciables [HAY99][MED01]:

- **Conexión simétrica.** Se da cuando existe una conexión del nodo i al nodo j, entonces también existe una conexión del nodo j al nodo i, y los pesos asociados $w_{ij} = w_{ji}$.
- **Conexión asimétrica.** Se da cuando no se cumplen las condiciones anteriores.

- **Conexión feedforward.** Todas las conexiones van en una dirección, desde la capa de neuronas de entrada hacia la capa de neuronas de salida.
- **Conexión recurrente.** Produce un cierto grado de retroalimentación, formando ciclos en la estructura topológica de la red.

Atendiendo a la taxonomía anterior, llamaremos *red feedforward* a las redes neuronales que únicamente contengan conexiones de tipo feedforward, ya sean estas simétricas o asimétricas (por ejemplo: Perceptrón multicapa, redes de base radial).

Por el contrario, si la estructura de interconexión de las neuronas de la red contiene alguna conexión recurrente (bien simétrica o bien asimétrica), diremos que se trata de un modelo de *red recurrente* (por ejemplo: Mapa autoorganizativo de Kohonen, red recurrente dinámica de Elman, red LSTM). Podemos distinguir tres tipos elementales de conexiones recurrentes [DAN01]:

- **Recurrencia local.** La salida de una neurona es retroalimentada a la propia neurona (Figura 1.1).

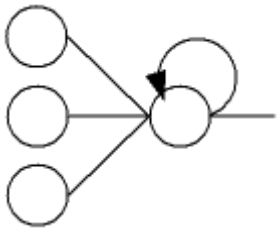

Figura 1.1. Recurrencia local

- **Recurrencia global.** La salida de una neurona es entrada a otras neuronas de capas anteriores (Figura 1.2).

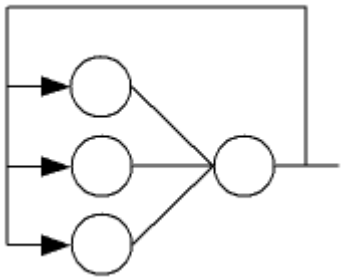

Figura 1.2. Recurrencia global

- **Recurrencia mixta**. Cuando los dos tipos de recurrencia anteriores tienen lugar concurrentemente.

La recurrencia local es un caso relativamente fácil de tratar. Sin embargo, la recurrencia global tiene una serie de implicaciones mucho más complejas, las cuales están íntimamente relacionadas con las características y dificultades de los métodos de entrenamiento [BEN94]. Dependiendo del funcionamiento interno de una red neuronal recurrente, estas pueden ser clasificadas a su vez de la siguiente forma [MED01]:

- **Redes neuronales recurrentes estacionarias**. En este tipo de modelos, las conexiones recurrentes se utilizan de modo que la red pueda ser utilizada como una memoria asociativa. Algunos ejemplos de redes recurrentes estacionarias son el modelo de Hopfield o los mapas autoorganizativos de Kohonen.
- **Redes neuronales recurrentes dinámicas.** En este tipo de modelos, las conexiones recurrentes sirven como memoria de almacenamiento de los patrones de entrada a lo largo del tiempo. Algunos ejemplos de redes neuronales recurrentes dinámicas son el modelo de Elman o las redes recurrentes completas.

Durante el desarrollo de la investigación, en los siguientes capítulos estudiaremos diferentes modelos de redes neuronales recurrentes dinámicas. Debido a la estructura topológica y el funcionamiento interno de la red, estos modelos son adecuados para resolver problemas asociados a aplicaciones que incluyan predicción de series de datos, como por ejemplo modelado de sistemas econométricos, procesamiento de la voz, medicina y control de procesos dinámicos.

1.2. Modelos de redes neuronales recurrentes dinámicas.

Las redes neuronales recurrentes dinámicas se caracterizan por poseer un grado de retroalimentación dentro de su estructura. Esta retroalimentación permite guardar información sobre los patrones de entrada proporcionados anteriormente a la red. Son, por tanto, redes que poseen un grado de memoria. Estos modelos pueden poseer conexiones recurrentes tanto en la

capa oculta como en la de salida, aunque nunca en la capa de entrada de datos. La Figura 1.3 muestra el esquema general de una red recurrente dinámica.

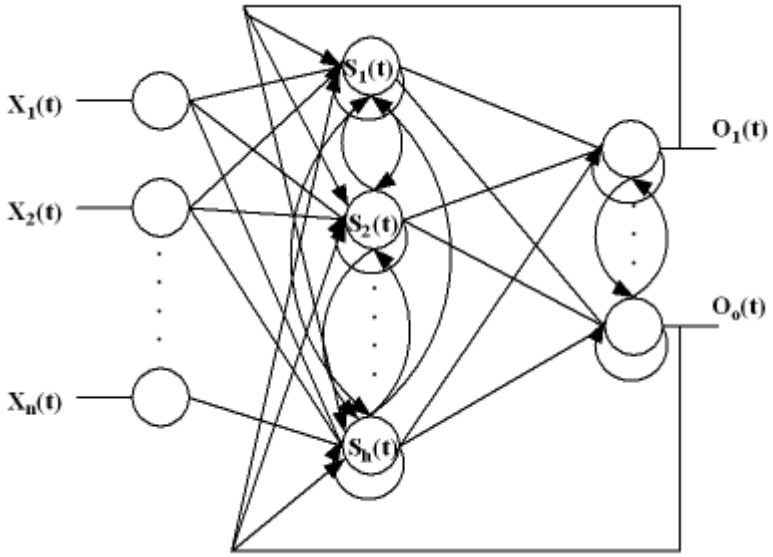

Figura 1.3. Esquema general de una red neuronal recurrente dinámica

La notación utilizada es la siguiente:

- Los valores $X_i(t)$ representan las entradas a la red en el instante t, donde $1 \leq i \leq n$.
- Los valores $S_j(t)$ representan las salidas de los nodos ocultos de la red en el instante t, donde $1 \leq j \leq h$.
- Los valores $O_k(t)$ representan las salidas de la red en el instante t, donde $1 \leq k \leq o$.
- Los valores n, h, o, representan el número de neuronas de entrada, ocultas y de salida, respectivamente.

En la estructura de interconexión de la red neuronal de la Figura 1.3 podemos distinguir tres tipos de capas: entrada de datos, capas ocultas y capa de salida. El objeto de estas capas es similar al que tienen sus equivalentes en redes *feedforward*, con el cálculo adicional que produce el procesamiento de las conexiones recurrentes.

Al conjunto de valores almacenados dentro de la estructura recurrente de este tipo de modelos en un instante dado *t*, $S_j(t)$, se le conoce como *estado interno* de la red. La salida de una red neuronal recurrente dinámica en el instante *t* depende de los valores almacenados en $S_j(t)$ y del patrón de entrada que se proporcione en ese instante. Como consecuencia, un mismo patrón de entrada puede producir diferentes patrones de salida, dependiendo del estado interno de las neuronas recurrentes. Esta característica proporciona a las RNRD la capacidad de procesamiento de patrones de longitud variable. Del mismo modo, también hace posible el análisis de secuencias de datos indexados en el tiempo.

Las diferentes arquitecturas de RNRD se derivan a partir de la estructura general de una red neuronal recurrente dinámica mostrada en la Figura 1.3. A continuación, explicaremos los diferentes modelos en los que centraremos la investigación.

Modelo de Red Neuronal Recurrente Completa

La *red neuronal recurrente completa* [DAN01][HAY99] consta de una capa de neuronas de entrada y una capa de neuronas ocultas/de salida. Cada neurona de la capa oculta/de salida puede tener una función de activación distinta y todas se realimentan a sí mismas y al resto de neuronas de la misma capa. La Figura 1.4 muestra un ejemplo de una red neuronal recurrente completa con *n* entradas, *h* nodos ocultos, y una salida.

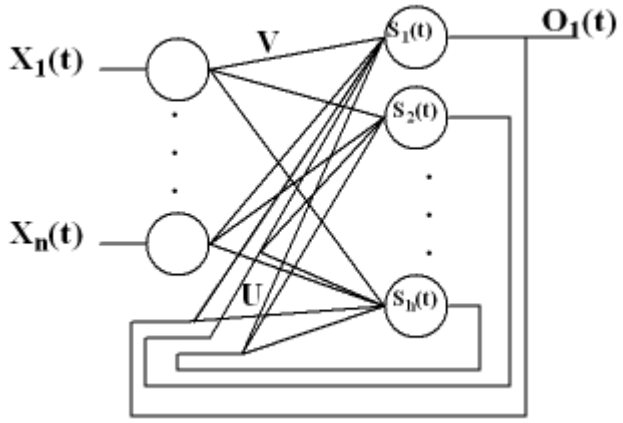

Figura 1.4. Esquema general de una red recurrente completa

Los valores de las neuronas en cada instante de tiempo dependen de las entradas que se proporcionan a la red y de los valores de las neuronas recurrentes en instantes anteriores. Por tanto, nos encontramos ante un sistema dinámico que se adapta con el tiempo a las características del entorno.

La noción de estado tiene una gran importancia en las formulaciones matemáticas de sistemas dinámicos. Considerando los modelos de RNRD, los valores de las neuronas recurrentes definen el estado actual de la red. Así, la salida podrá variar dependiendo del estado actual y de los datos que se presenten en la capa de entrada. Tradicionalmente, se define el estado de un sistema dinámico como un conjunto de valores, los cuales contienen toda la información necesaria para describir el futuro del mismo. Estos valores se calculan a partir de los estados anteriores por los que haya evolucionado el sistema.

Siendo *S(t)* el estado de nuestro sistema en el instante *t*, *V* y *U* matrices de valores desconocidos y *X(t)* la entrada al sistema en *t*, la evolución del estado de una red neuronal recurrente dinámica se rige por la siguiente fórmula:

$$S(t+1) = f(V \cdot X(t) + U \cdot S(t))$$

Así, el estado de la red recurrente en el instante *t+1* depende funcionalmente del estado en el tiempo actual *S(t)* y de las entradas *X(t)*.

Consideremos el modelo de sistema dinámico expuesto. Si *f(·)* es una función no lineal prefijada, y *X(t)* y *S(t)* son valores conocidos en el instante *t*, para conocer el funcionamiento del sistema se deberá calcular los valores de las matrices *V* y *U*. Este modelo puede ser fácilmente extrapolado al de una red neuronal recurrente completa, donde las matrices *V* y *U* se corresponden con los pesos de la red. Sea $S_i(t)$ el valor de la neurona oculta *i* en el instante *t*. Las ecuaciones siguientes muestran la dinámica del comportamiento de lared:

$$S_i(t) = f_i(neth_i(t)) \tag{1.1}$$

$$neth_i(t) = \sum_{j=1}^{n} V_{ij} X_j(t) + \sum_{j=1}^{h} U_{ij} S_j(t) \tag{1.2}$$

$$O_k(t) = S_k(t),\ 1 \le k \le o \tag{1.3}$$

El siguiente esquema describe la notación utilizada:

- V_{ij} son los pesos de la conexión entre el valor de entrada j y la neurona oculta/de salida i, con $1 \leq i \leq h$, $1 \leq j \leq n$.
- U_{ij} es el peso de la conexión entre el valor del estado dado por la neurona oculta j y la neurona oculta i, con , con $1 \leq i \leq h$, $1 \leq j \leq h$.
- Los valores n, h, o son el número de entradas, neuronas ocultas, y de salida de la red recurrente, respectivamente.
- $f_i(\cdot)$ es la función de activación de la neurona oculta i.

Como condición inicial para la recurrencia, el estado inicial tendrá valor $S_i(0) = 0 \ \forall i \neq 1$, $S_1(0) = 1$.

Modelo de Red Neuronal Recurrente de Elman

El modelo de Elman [ELM90] es similar a las redes neuronales recurrentes completas. La principal diferencia reside en que las salidas en una red de Elman dependen de las entradas en el tiempo actual $X(t)$ y del estado de la red calculado en el instante actual $S(t)$, mientras que las salidas de las redes recurrentes completas dependen de la entrada en el tiempo actual $X(t)$ y del estado de la red calculado en el instante anterior, $S(t-1)$. La Figura 1.5 muestra un ejemplo de una red recurrente de Elman con n entradas, h neuronas ocultas, y o neuronas de salida. La dinámica de una red recurrente de Elman se resume en las siguientes ecuaciones:

$$neth_j(t) = \sum_{i=1}^{h} U_{ji} S_i(t-1) + \sum_{i=1}^{n} V_{ji} X_i(t) \tag{1.4}$$

$$S_j(t) = f(neth_j(t)) \tag{1.5}$$

$$neto_k(t) = \sum_{j=1}^{h} W_{kj} S_j(t) \tag{1.6}$$

$$O_k(t) = g(neto_k) \tag{1.7}$$

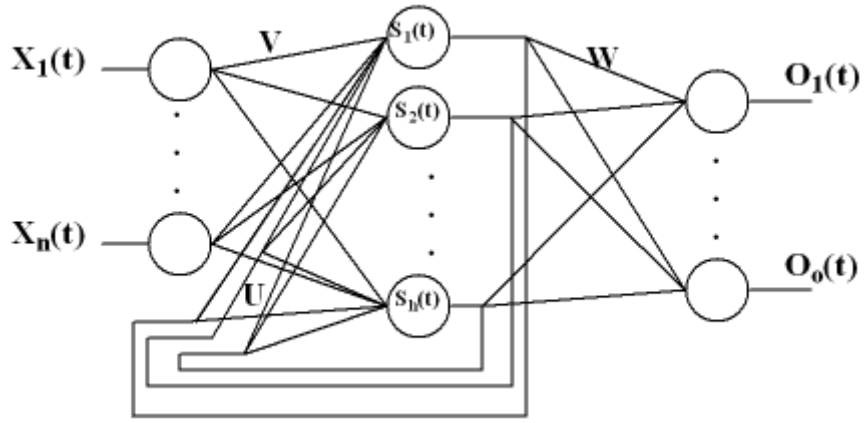

Figura 1.5. Esquema general de una red recurrente de Elman

La notación asociada a la Figura 1.5 y las ecuaciones 1.4 a 1.7 se muestra en el siguiente esquema:

- V_{ij} simboliza los pesos que van desde las entradas hasta las neuronasocultas ($1 \leq i \leq h$, $1 \leq j \leq n$);
- U_{ij} contiene los pesos entre el estado de la red y las neuronas ocultas($1 \leq i \leq h$, $1 \leq j \leq h$),
- W_{ij} engloba los pesos entre las neuronas ocultas y las neuronas de salida($1 \leq i \leq o$, $1 \leq j \leq h$).
- $O_k(t)$ es la salida de la neurona k de la capa de salida en el tiempo t.
- $neto_k(t)$ es la salida de la neurona k de la capa de salida, en el tiempo t, sin aplicarle aún la función de activación.
- El valor $S_j(t)$ es la salida de la neurona j de la capa oculta, en el instante t.
- n$eth_j(t)$ es la salida de la neurona j de la capa oculta sin aplicarle la función de activación, en el instante t.
- La función $f(\cdot)$ es la función de activación de las neuronas ocultas.
- La función $g(\cdot)$ es la función de activación de las neuronas de salida.
- $X_i(t)$ es el valor de la neurona i de la capa de entrada, en el tiempo t.
- Las variables n, h, o son el número de neuronas de entrada, ocultas y de salida, respectivamente.

Al igual que en las redes neuronales recurrentes completas, la condición inicial para la recurrencia es $S_i(0)= 0\ \forall i \neq 1$, $S_1(0)= 1$.

Modelo de Red Neuronal Recurrente de Jordan

Este modelo de red [DAN01][HAY01] posee las conexiones recurrentes entre las neuronas de la capa de salida y las de la capa oculta. De este modo, el estado de la red en el tiempo *t* está formado por las salidas que proporciona la red en el instante *t-1*.

La Figura 1.6 muestra un ejemplo de red de Jordan con *n* entradas, *h* neuronas ocultas y *o* neuronas de salida.

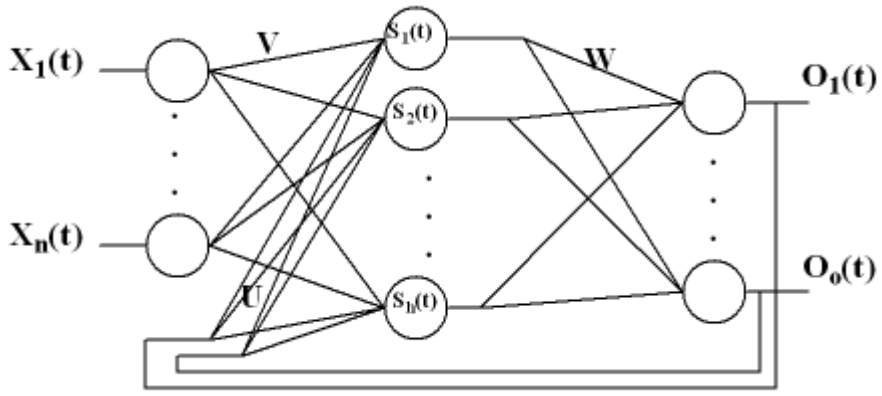

Figura 1.6. Esquema general de una red recurrente de Jordan

Las ecuaciones siguientes muestran la dinámica de la red recurrente de Jordan:

$$neth_j(t) = \sum_{i=1}^{o} U_{ji} O_i(t-1) + \sum_{i=1}^{n} V_{ji} X_i(t) \tag{1.8}$$

$$S_j(t) = f(neth_j(t)) \tag{1.9}$$

$$neto_k(t) = \sum_{j=1}^{h} W_{kj} S_j(t) \tag{1.10}$$

$$O_k(t) = g(neto_k) \tag{1.11}$$

La notación asociada a la Figura 1.6 y a las ecuaciones 1.8 a 1.11 se resume en el siguiente esquema:

- V_{ij} codifica los pesos que van desde las entradas hasta las neuronas ocultas ($1 \leq i \leq h$, $1 \leq j \leq n$).
- U_{ij} engloba los pesos entre el estado de la red y las neuronas ocultas ($1 \leq i \leq h$, $1 \leq j \leq o$).
- W_{ij} contiene los pesos entre las neuronas ocultas y las neuronas de salida ($1 \leq i \leq o$, $1 \leq j \leq h$).
- $O_k(t)$ es la salida de la neurona k de la capa de salida en el tiempo t.
- $neto_k(t)$ es la salida de la neurona k de la capa de salida, en el tiempo t, sin aplicarle aún la función de activación.
- $S_j(t)$ es la salida de la neurona j de la capa oculta, en el instante t.
- $neth_j(t)$ es *la* salida de la neurona j de la capa oculta en el instante t, sin aplicarle la función de activación.
- $f(\cdot)$ es la función de activación de las neuronas ocultas.
- $g(\cdot)$ es la función de activación de las neuronas de salida.
- $X_i(t)$ es el valor de la neurona i de la capa de entrada, en el tiempo t.
- Las variables n, h, o son el número de neuronas de entrada, ocultas y de salida, respectivamente.

La particularidad de la red de Jordan, en comparación con los modelos de redes recurrentes completa y de Elman, radica en que el estado está formado por las salidas de las neuronas de la última capa en el instante anterior, mientras que en las anteriores está formado por las salidas de las neuronas ocultas.

El número de neuronas de estado de la red de Jordan está limitado por la estructura del patrón de salida de la red neuronal. Al contrario, en los modelos recurrentes completa y de Elman éste puede ser modificado, mediante el aumento o la disminución del número de neuronas ocultas de la red.

Este hecho puede parecer una limitación de la red de Jordan con respecto a los otros modelos de redes estudiados; sin embargo, no es tal cuando tratamos de aprender secuencias de

datos que dependen de los valores de salida anteriores de la red. En términos puramente prácticos podemos afirmar, dada la estructura de las redes presentadas, que el modelo de Jordan memoriza la evolución de las salidas de la red, mientras que las redes de Elman y recurrente completa memorizan la secuencia de entradas a la misma.

2. Entrenamiento de Redes Neuronales Recurrentes Dinámicas

Dependiendo del tipo de aplicación sobre la que se desee emplear una RNRD, podemos diferenciar entre los casos en los que los pesos de la red deben ser actualizados dinámicamente (por ejemplo: control de plantas o procesos dinámicos), y los casos en los que los pesos de la red no se modifican una vez que esta ha sido entrenada (por ejemplo: reconocimiento de patrones). En ambos casos el modelo de entrenamiento de la red es diferente Pueden distinguirse dos tipos de entrenamiento para una red recurrente [HAY99][MED01]:

- **Entrenamiento por épocas.** Para una época dada, la red recurrente comienza inicializada con un estado predeterminado. Entonces se presentan ejemplos de entrenamiento hasta alcanzar un estado final esperado. Una vez llegados a este punto, el estado de la red es inicializado, y comienza una nueva época.
- **Entrenamiento continuo.** Este método es adecuado en situaciones en las que no hay estados de reset, o se requiere aprendizaje en línea. La característica principal del entrenamiento continuo es que la red recurrente aprende durante el procesamiento de los ejemplos de entrada. Este modelo de entrenamiento es útil, por ejemplo, en situaciones en las que la red ha de aprender a manejar un proceso no estacionario (como por ejemplo, una señal de voz).

La estructura de los patrones de datos de entrada/salida también juega un papel importante a la hora de seleccionar el método de entrenamiento más adecuado para la red. Dependiendo del tipo de problema que pretendamos resolver, podremos disponer de:

1. **Un conjunto de patrones de entrada/salida**, cuya longitud puede variar. Como ejemplo ilustrativo, podemos citar una aplicación de reconocimiento de ADN, dondedos patrones de entrada/salida (uno positivo y uno negativo) podrían tener una estructura similar a la siguiente:

   ACCTCAACATT / 1

   TTATCCACA / 0

   El entrenamiento de la red, en este tipo de problemas, consiste en hallar las relaciones entre la estructura de los diferentes patrones de entrada, para obtener las salidas de red adecuadas.

2. **Un único patrón de entrada/salida** El modelo de entrenamiento asociado a este tipo de problemas suele ser el entrenamiento continuo. Como ejemplo de aplicación, podemos citar el control de un proceso dinámico, como el de una planta industrial o, en general, cualquier problema de predicción de series de datos.

Durante el desarrollo de los siguientes capítulos haremos especial énfasis en problemas con características típicas del segundo tipo. Suponiendo un problema depredicción de series de datos, el patrón de entrada ha de presentarse a la red en orden cronológico. Este orden de presentación viene impuesto por la necesidad de la actualizacióndinámica del estado interno de la red recurrente. Ilustraremos esta idea mediante un ejemplo.Supongamos un patrón de datos X definido como:

$$X \in R^{n \cdot m} = X(t) \in R^m, 1 \le t \le T$$

Este patrón de datos se presenta a la red indexado en el tiempo y ordenado de la forma X(1), X(2), X(3), ..., X(t), X(t+1), ...X(T). De este modo, la evolución del estado interno de la red en un instante dado, *S(t)*, se realiza simultáneamente a la presentación de los patrones de entrada en cada instante: X(1)/S(1), X(2)/S(2), ..., X(t)/S(t), X(t+1)/S(t+1), ..., X(T), S(T).

Dada la estructura anterior de patrón de entrada a la red, en los problemas que trataremos a lo largo de los capítulos definiremos los conjuntos de datos de entrenamiento y de test de modo que el conjunto de entrenamiento estará formado por el porcentaje *p%* inicial de la serie de datos, mientras que el conjunto de test estará formado por el restante *(100-p)%*. Esta es la única subdivisión que podemos hacer del conjunto de datos, para entrenamiento y test. Esto es debido a la necesidad de indexar la serie de datos a lo largo del tiempo.

El entrenamiento de una RNRD [DAN01] consiste en minimizar una medida de error dada por las ecuaciones 1.12 y 1.13, a lo largo del tiempo. La función error a minimizar será el error cuadrático medio entre la salida de la red *O(t)* y los valores de la serie temporal en el instante siguiente, *Y(t+1)*.

$$E = \frac{1}{T} \sum_{t=1}^{T} E(t) \tag{1.12}$$

$$E(t) = \sum_{k=1}^{o} (O_k(t) - y_k(t+1))^2 \tag{1.13}$$

La notación utilizada en las ecuaciones 1.12 a 1.13 se resume en el siguiente esquema:

- T es la longitud de la serie de datos.
- $O_k(t)$ es el valor de salida de la neurona k de la red recurrente, en el instante t.
- $y_k(t)$ es el valor de salida esperado para la neurona k en el instante t.
- La variable o es el número de neuronas de salida de la red neuronal.

Podemos resumir los procesos de entrenamiento y test en los esquemas ilustrados por los Algoritmos 1.1 y 1.2 [HAY99][MED01].

Entrenamiento:

1. Mientras (condición de parada = falso) hacer:

2. Inicializar Red Neuronal

3. Presentar los patrones de entrada siguiendo el orden temporal de los mismos (modificando a cada paso los pesos de la red, si es entrenamiento secuencial)

4.Modificar los pesos de la red, si es entrenamiento por lotes

5. Devolver la red neuronal entrenada

Algoritmo 1.1. Procedimiento general de entrenamiento de redes neuronales recurrentes dinámicas

Test:

1. Inicializar Red Neuronal

2. Presentar Patrones de entrenamiento

3. Presentar Patrones de Test

4. Devolver error de test

Algoritmo 1.2. Procedimiento general de presentación de patronesde test

Gran parte de los algoritmos clásicos de entrenamiento de redes neuronales están basados en el gradiente. Tras la presentación de un par entrada/salida a la red, el valor de los pesos se reajusta según el error de salida. Dado un peso w de una red neuronal, la regla general de actualización es la siguiente:

$$w = w \pm \mu \cdot \Delta w$$

donde Δw es una medida de la variación del peso a realizar, generalmentecalculada a partir de la información del gradiente y del error cometido en la salida de la red. El valor μ es la *tasa de aprendizaje.*

El cálculo del término Δw varía según el modelo entrenamiento utilizado (por épocas o continuo). Los algoritmos clásicos más utilizados para entrenamiento de redes neuronales recurrentes, en cada tipo de entrenamiento, son el BackPropagation Through Time (BPTT) [DAN01][HAY99][JAE02], y el RTRL [DAN01][WIL89][WIL90], ambos basados en la propagación de errores y el gradiente. Utilizaremos ambos algoritmos como métodos de comparación respecto con los que propondremos en los capítulos siguientes.

Algoritmo BackPropagation Through Time

El algoritmo BackPropagation Through Time es una extensión del algoritmo BackPropagation estándar para redes feedforward. Puede ser deducido a partir de una simplificación de la red recurrente, construyendo una red feedforward equivalente para cada instante de tiempo *t*. Existen diversas variantes del algoritmo BPTT, tales como el *Epochwised BPTT*, o el *Truncated BPTT*. De todas ellas, la variante más conocida es el algoritmo *Truncated BPTT*, propuesta por Williams y Peng en 1990. Las Figuras 1.7 y 1.8 muestran un ejemplo del desglose de una red recurrente de Elman para un intervalo de dos unidadesde tiempo.

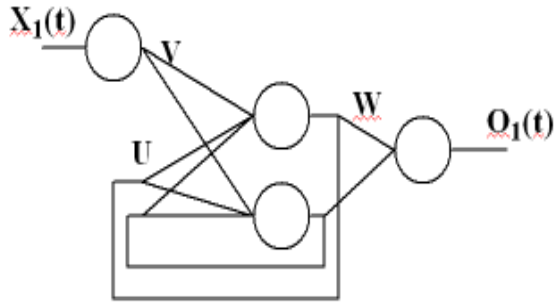

Figura 1.7. Ejemplo de red neuronal recurrente de Elman con una entrada, una salida y dos neuronas ocultas

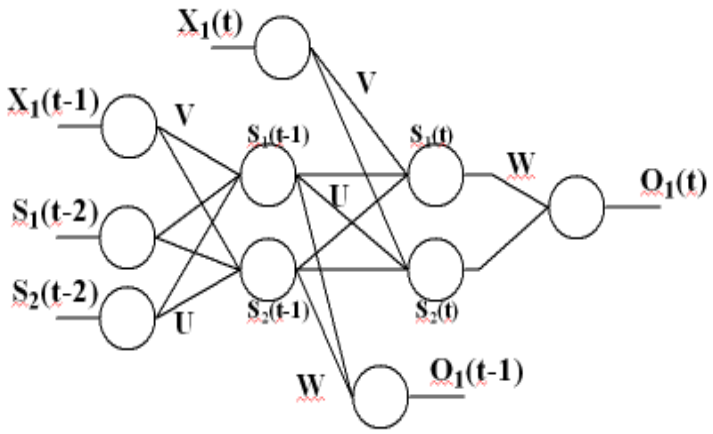

Figura 1.8. Red neuronal recurrente de Elman, desglosada dos unidades en el tiempo

La red de la Figura 1.7 posee una entrada, una salida, y dos neuronasocultas. La Figura 1.8 muestra la red neuronal feedforward equivalente, construida a partir del desglose en el tiempo de las conexiones recurrentes. Cada neurona de las capas ocultas representa el valor de las neuronas de estado en el tiempo correspondiente. Del mismo modo, las entradas a la red desglosada son las propias entradas a la red recurrente, en los tiempos correspondientes, junto con el estado de la red recurrente en el tiempo anterior al máximo considerado en el desglose.

Para poder utilizar el algoritmo de entrenamiento BPTT, es importante destacar que los pesos de las conexiones de la red feedforward simplificada son idénticos en todas las capas de desglose.

La variante del algoritmo BPTT más conocida es el *Truncated BackPropagation Through Time* (TBPTT). La idea subyacente del algoritmo TBPTT es que la información utilizada por la red, anterior al instante *(t-h),* es irrelevante para calcular la salida en el instante *t*. El valor *h* es llamado *horizonte de truncamiento,* El uso del horizonte de truncamiento reduce considerablemente el tiempo de entrenamiento, y la memoria necesaria para guardar los estados de la red recurrente en cada uno de los instantes de tiempo considerados.

En los campos de aplicación de las redes neuronales recurrentes dinámicas, el uso del truncamiento no es tan artificial como puede parecer en un principio. A menos que la red recurrente sea inestable, debería existir un punto de convergencia para las derivadas del error

con respecto a cada peso de la red, ya que los cálculos realizados en el tiempo se corresponden con la función de activación (típicamente sigmoide), aplicada multitud de veces.

En cualquier caso, el horizonte de truncamiento h debe ser suficientemente grande como para asegurar que la salida de la red sea la adecuada. El Algoritmo 1.3 muestra el esquema general del algoritmo de TBPTT. A continuación, explicamos la adaptación para realizar el entrenamiento de cada tipo de red recurrente, expuestos en apartados anteriores.

La ecuación 1.14 expone, de forma general, cómo han de modificarse los pesos de la red recurrente dinámica. Dependiendo del tipo de conexión asociada al peso P y del tipo de red, el cálculo de $\Delta P_{ij}(N)$ debe realizarse de una forma u otra.

$$P = P - \mu \sum_{N=I-H}^{I} \Delta P(N) \tag{1.14}$$

Considerando una *red recurrente de Elman*, podemos encontrar tres tipos de pesos diferentes. Utilizando la notación introducida hasta el momento, el conjunto de pesos se representa mediante W (correspondientes a las conexiones de las neuronas ocultas a las neuronas de salida), V (correspondientes a las conexiones entradas-neuronas ocultas), y U (correspondientes a las conexiones recurrentes).

En cada paso del algoritmo, los pesos V_{ij}, U_{is} y W_{ki} se actualizan según indica la ecuación 1.14, donde i es la neurona oculta destino, k corresponde al índice de la neurona de salida, y j y s son los índices referentes a las neuronas de entrada o recurrentes, respectivamente.

Atendiendo a la notación introducida en las secciones anteriores, los valores para $\Delta V_{ij}(N)$, $\Delta U_{is}(N)$ y $\Delta W_{kj}(N)$ se calculan según se indica a continuación:

$$\Delta W_{ki}(N) = \frac{2}{T}(O_k(N) - Y_k(N))g'(neto_k(N))S_i(N) \tag{1.15}$$

$$\Delta V_{ij}(N) = \delta_i(N)X_j(N) \tag{1.16}$$

$$\Delta U_{is}(N) = \delta_i(N)S_s(N-1), \tag{1.17}$$

donde

$$\delta_i(N) = f'(neth_i(N)) \sum_{k=1}^{o} W_{ki} \frac{2(O_k(N) - Y_k(N))}{T} g'(neto_k(N))$$

0. Entradas al algoritmo:
h= Horizonte de truncamiento
μ= Tasa de aprendizaje
X(n) (n= 1..T = entradas a la red, indexadas en el tiempo
Y(n) (n= 1..T)= salidas deseadas para la red, indexadas en el tiempo

1. Inicializar los pesos de la red
2. Mientras (condición de parada = falso), hacer:
3. Inicializar estado de la red
4. Para cada valor de entrada, X(i), i=1..T, hacer:
4.1. calcular la salida de la red, O(i)
4.2. Para cada instante n=i-h, ..., i, hacer:
4.3. Para cada peso P de la red, calcular ΔP(n)
4.4. Modificar el peso P, según la ecuación 3.14
5. Calcular el error cometido
6. Devolver la red entrenada

Algoritmo 1.3. Procedimiento general del algoritmo TBPTT

La *red recurrente de Jordan*, al igual que la red de Elman, tiene tres tipos de pesos, simbolizados mediante *V, U* y *W*. La actualización de tales pesos, mediante el algoritmo BPTT, también es muy similar:

El cálculo de los valores $\Delta V_{ij}(N)$ y $\Delta W_{kj}(N)$, en una red recurrente de Jordan, se lleva a cabo según las ecuaciones 1.15 y 1.16. En cambio, el valor $\Delta U_{is}(N)$ se obtiene a partir de la siguiente expresión:

$$\Delta U_{is}(N) = \delta_i(N) O_s(N-1) \tag{1.18}$$

La *red recurrente completa* tiene dos tipos de pesos, representados mediante V (correspondientes a las conexiones entradas-neuronas ocultas), y U (correspondientes a las conexiones recurrentes). La actualización de los pesos requiereel cálculo de los valores $\Delta V_{ij}(N)$ y $\Delta U_{is}(N)$, los cuales se obtienen a partir de las ecuaciones 1.19 y 1.20.

$$\Delta V_{ij} = \begin{cases} 2\dfrac{(O_j(N) - Y_j(N))}{T} f'(neth_i(N)) X_j(N) & ; 1 \le i \le o \\ 0 & ; i > o \end{cases} \tag{1.19}$$

$$\Delta U_{is} = \begin{cases} \dfrac{2}{T}(O_i(N) - Y_i(N)) f'(neth_i(N) S_s(N) & ; 1 \le i \le o) \\ 0 & ; i > o \end{cases} \tag{1.20}$$

Algoritmo Real-Time Recurrent Learning

El algoritmo Real-Time Recurrent Learning (RTRL) es un algoritmo basado en el método del gradiente descendente. Fue propuesto por Williams y Zipser en 1989, para entrenamiento en tiempo real de redes recurrentes completas. Los pesos de la red entrenada mediante este método se actualizan cada vez que reciben un par dedatos entrada/salida a la red. De este modo, se consigue un entrenamiento de la red en tiempo real.

Al contrario que el algoritmo TBPTT, el algoritmo RTRL no despliega la red en el tiempo, ni trunca los valores acumulados por la recurrencia de la red. En su lugar, RTRL calcula la variación en el valor de un peso, en un instante dado t, en función de la variación de los pesos en el instante anterior $t-1$. La actualización de los pesos se realiza de forma dinámica, de una forma eficiente en cuanto a cantidad de cálculo y memoria utilizada. Mientras que el algoritmo TBPTT debe memorizar los valores del estado de las neuronas ocultas en h instantes anteriores, el algoritmo RTRL sólo salva las variaciones realizadas enlos pesos, en el instante justo anterior al actual.

Para describir el algoritmo, utilizaremos el término $\nabla P^k_{ij}(t)$ para referirnos a la variación acumulada de un peso genérico de la red, producida por la neurona oculta k. El Algoritmo 1.4

muestra el esquema general del método de entrenamiento RTRL. En la ecuación 1.21, el valor $\delta_k(t)$ es el valor de la propagación del error, correspondiente a la neurona oculta k, en el instante t. Dependiendo del tipo de peso que sea P_{ij} y del tipo de red que se desea entrenar, $\delta_k(t)$ puede ser calculado de diferentes formas.

0. Entradas al algoritmo:
μ= Tasa de aprendizaje
X(n) (n= 1..T = entradas a la red, indexadas en el tiempo
Y(n) (n= 1..T)= salidas deseadas para la red, indexadas en el tiempo

1. Inicializar los pesos de la red
2. Mientras (condición de parada = falso), hacer:
3. Inicializar estado de la red
4. Hacer $\nabla P^k_{ij}(0)= 0$, *para cualquier combinación posible de i, j, k*
5. Para cada valor de entrada, X(i), i=1..T, hacer:
5.1. calcular la salida de la red, O(i)
5.2. Actualizar los valores $\nabla P^k_{ij}(I)$
5.3. Modificar el peso P_{ij} *según indica la ecuación 1.21*
6. Calcular el error cometido
7. Devolver la red entrenada

Algoritmo 1.4. Procedimiento general del algoritmo RTRL

Para entrenar una *red neuronal recurrente completa* mediante el algoritmo RTRL utilizaremos los símbolos $\nabla V^k_{ij}(t)$ y $\nabla U^k_{ij}(t)$ como representación de la variación de los pesos, acumulada a lo largo del tiempo desde el instante inicial hasta el tiempo t. Denominaremos $\nabla U^k_{ij}(t)$ al efecto que tiene la conexión recurrente que va desde la neurona i hasta la neurona j, respecto a la salida de la neurona k, en el instante t. A su vez, $\nabla V^k_{ij}(t)$ simboliza el efecto que tiene la conexión que va desde la neurona de entrada i hasta la neurona oculta j, respecto a la

salida de la neurona *k*, en el instante *t*. Los valores $\nabla V^k_{ij}(t)$ y $\nabla U^k_{ij}(t)$ se calculan según se indica a continuación.

$$P_{ij} = P_{ij} - \mu \sum_{t=1}^{I} \sum_{k=1}^{h} \delta_k(t) \nabla P^k_{ij}(t) \tag{1.21}$$

$$\nabla V^k_{ij}(t) = f'(neth_k(t)) \sum_{l=1}^{N} V_{kj} \nabla V^l_{ij}(t) + \delta_{ik} X_j(t) \tag{1.22}$$

$$\nabla U^k_{is}(t) = f'(neth_k(t)) \sum_{l=1}^{N} U_{ks} \nabla U^l_{is}(t) + \delta_k(t) S_s(t-1) \tag{1.23}$$

donde:

$$\delta_{ik} = \begin{cases} 0 ; i = k \\ 1 ; i \neq k \end{cases}$$

$$\delta_k(t) = \begin{cases} \frac{2}{T}(O_k(t) - Y_k(t)) & ; si\ k\ es\ salida \\ 0 & ; en\ otro\ caso \end{cases}$$

El entrenamiento de una *red recurrente de Elman*, mediante el algoritmo RTRL, también se rige por las ecuaciones 1.22 y 1.23. No obstante, el cálculo del valor de la propagación del error se realiza según indica en:

$$\delta_i(t) = f'(neth_i(N)) \sum_{k=1}^{o} W_{ki} \frac{2(O_k(N) - Y_k(N))}{T} g'(neto_k(N)) \tag{1.24}$$

El conjunto de pesos W_{ij}, no considerados en las ecuaciones anteriores, se actualiza según:

$$W_{ki} = W_{ki} - \mu \sum_{N=1}^{I} \frac{2(O_k(N) - Y_k(N))}{T} g'(neto_k(N)) S_i(N) \tag{1.25}$$

El entrenamiento de una *red recurrente de Jordan*, mediante el algoritmo RTRL, calcula la variación de los diferentes pesos de la red de acuerdo a las indicaciones de las ecuaciones 1.21, 1.22 y 1.25, con la salvedad de que el cálculo del gradiente acumulado, $\nabla U^k_{ij}(t)$, varía según:

$$\nabla U_{is}^{k}(t) = f'(neth_k(t)) \sum_{l=1}^{N} U_{ks} \nabla U_{is}^{l}(t) + \delta_{ik} O_s(t-1) \tag{1.26}$$

El problema principal que presentan los algoritmos clásicos de entrenamiento de redes neuronales radica en el hecho de que tienden a obtener soluciones locales con frecuencia. Por ello, es difícil encontrar una solución óptima:

- Si el valor de μ es excesivo, el algoritmo tiene capacidad de evitar óptimos locales, pero puede realizar cambios demasiado grandes en los pesos, de modo que la convergencia resulta más complicada.
- Por otra parte, si el valor de μ es muy pequeño, entonces la convergencia se ralentiza, y hacen falta muchas iteraciones para que el algoritmo obtenga una solución. Además, en este caso, debido a que las actualizaciones de los pesos son pequeñas, el algoritmo tiene mayor tendencia a caer en óptimos locales de los que no pueda salir.

Considerando las arquitecturas de redes neuronales recurrentes dinámicas, los problemas planteados son de mayor complejidad, debido esencialmente a la dificultad introducida por el tratamiento de la recurrencia. El problema principal del entrenamiento, en este tipo de redes, radica en que la propagación del gradiente a través de las conexiones recurrentes tiende a suavizarse, dificultando la obtención de soluciones óptimas [BEN94].

Este problema es aún mayor cuando las dependencias temporales entre los pares entrada/salida de la red son a largo plazo. En estos casos, es complicado obtener un entrenamiento aceptable de la red. El propósito principal de los capítulos posteriores nos llevará a desarrollar nuevas técnicas que solventen los problemas mencionados

3. Algoritmos heurísticos para entrenamiento de redes neuronales recurrentes dinámicas.

La utilización de métodos de naturaleza heurística para entrenamiento de redes neuronales surge en la década de los 90 [MON89]. Las motivaciones principales que sugieren el uso de este tipo de técnicas, para solucionar el problema del entrenamiento de redes neuronales, vienen determinadas en gran medida por las limitaciones presentadas por los algoritmos clásicos de aprendizaje:

- Imposibilidad de calcular el gradiente cuando la función de activaciónde las neuronas no es derivable.
- Ausencia de convergencia de los algoritmos clásicos de entrenamiento, cuando el número de bits utilizados para representar la función de activación o los pesos de la red no es suficientemente grande.
- Tendencia, por parte de los algoritmos de entrenamiento, para obtener excesivas soluciones no optimales en cada ejecución.

Debido en parte a la gran popularidad y auge que comenzaron a tener ciertas técnicas heurísticas en la década de los 90, y a las necesidades tecnológicas de la época, los métodos de naturaleza heurística comenzaron a ser considerados como alternativas prometedoras para entrenamiento de diversos modelos de redes neuronales, los cuales solventaban las limitaciones anteriores.

Algunos de los métodos heurísticos que se han utilizado para entrenarredes neuronales fueron el método de enfriamiento simulado, y la búsqueda tabú. Actualmente, aún podemos encontrar publicaciones recientes que aplican algoritmos basados en este tipo de heurísticas para entrenamiento y optimización de diversos modelos de redes neuronales [DIA96][MAR03].

Los algoritmos heurísticos que más impacto han tenido, relacionados con diversas arquitecturas y problemas de redes neuronales, han sido los basados en modelos evolutivos. A

partir de la segunda mitad de la década de los 90, podemos encontrar numerosas contribuciones en la bibliografía, tratando temas mayoritariamente relacionados con los problemas siguientes:

- Entrenamiento de redes neuronales. Este tipo de aplicaciones están consolidadas en la actualidad, siendo conocido como entrenamiento evolutivo de redes neuronales, o neuroevolución [SET96][DEL01] (ejemplo: entrenamiento de una red neuronal mediante algoritmos genéticos).
- Construcción de redes neuronales. Este tipo de contribuciones abordan el problema de encontrar la estructura óptima de una red neuronal que mejores resultados proporciona al aprender un problema concreto [HER02][LIU98][LIU99] (por ejemplo: coevolución de subelementos de una red neuronal, aplicación de algoritmos evolutivos multiobjetivo).
- Entrenamiento y optimización de la topología de una red neuronal. Este tipo de contribuciones pretenden optimizar la estructura topológica de una red neuronal, mientras la entrenan. Al igual que en la aplicación anterior, destacan los trabajos realizados con algoritmos de coevolución y algoritmos multiobjetivo [DEI00][HUS03].

En los últimos años, las metaheurísticas híbridas han tenido un gran impacto sobre la bibliografía que podemos encontrar [ISH03][JAS02][MEL03][MOS03][RAD94]. Este impacto también ha alcanzado a la aplicación de este tipo de técnicas para entrenar redes neuronales [CON02][KU97][PRU02][SHI99], obteniendo resultados muy prometedores en diversos modelos de red.

En los capítulos siguientes, utilizaremos los algoritmos evolutivos como base para la investigación realizada. Abordaremos el tema del entrenamiento evolutivo de una red neuronal, desde la perspectiva del problema de diversidad y convergencia propio de los algoritmos evolutivos. Del mismo modo, desarrollaremos algoritmos evolutivos híbridos que mejoren la etapa del entrenamiento de una red neuronal recurrente dinámica. También abordaremos el problema de la optimización de la estructura topológica de estos modelos de red, mediante la utilización de algoritmos evolutivos multiobjetivo, y mejoraremos este proceso mediante el desarrollo de algoritmos evolutivos multiobjetivo híbridos.

A continuación, las secciones siguientes hacen una breve descripción de las técnicas heurísticas de mayor impacto antes comentadas, utilizadas para entrenamiento de redes neuronales.

3.1. Algoritmo de Enfriamiento Simulado.

El método de enfriamiento simulado puede caracterizarse como una técnica de búsqueda global de soluciones, aplicado en problemas de optimización discretos y continuos. Es un método iterativo, el cual realiza una serie de operaciones en cada iteración, denominadas *movimientos*, sobre una solución [DIA96].

Considerando el problema del entrenamiento de una red neuronal, una solución está compuesta por un vector que codifica los pesos de la red a entrenar. Como suposición previa, la red debe tener una topología fija predefinida.

Los resultados son aceptados si la solución resultante cumple ciertas condiciones (por ejemplo, que se haya mejorado el aprendizaje de la red), o rechazados en caso contrario. Seguidamente, el algoritmo selecciona ciertas soluciones candidatas, escogidas de forma aleatoria a partir de una distribución uniforme en un entorno de la solución inicial. La dimensión de dicho entorno se ve afectada por un parámetro denominado *temperatura*, el cual se actualiza según la evolución del algoritmo.

El esquema anterior es repetido iterativamente, hasta que se cumpla una cierta condición de parada (por ejemplo, que la red alcance un comportamiento adecuado).

3.2. Algoritmo de Búsqueda Tabú.

Al igual que el método anterior, la búsqueda tabú puede caracterizarse como una técnica de búsqueda global de soluciones [GLO96][MAR03]. Los algoritmos de búsqueda tabú se

caracterizan por mantener en memoria una serie de movimientos realizados anteriormente, denominados *movimientos tabú.*

En cada iteración, el algoritmo realiza una búsqueda para mejorar la solución actual, teniendo en cuenta los movimientos anteriores, de modo que dirija la búsqueda siguiendo direcciones que hayan producido buenos resultados en el pasado, y evitando las que hayan sido poco productivas. Los resultados son aceptados si se cumplen las condiciones tabú y si la solución encontrada en mejor que la anterior.

3.3. Algoritmos evolutivos.

Los algoritmos evolutivos son métodos de búsqueda y optimización, basados en procesos naturales [BAC96][GOL89][RAW91]. Son modelos no determinísticos utilizados para buscar, mediante heurísticas, soluciones a problemas cuyo espacio de soluciones es muy amplio. En comparación con los mecanismos de evolución natural, podemos enumerar las siguientes características y similitudes:

- Deben existir individuos que puedan reproducirse.
- Dependiendo del algoritmo, puede haber un único individuo o un conjunto de ellos (población).
- Cada individuo debe poseer características propias que le haga diferenciarse del resto. Estas características permiten a un individuo adaptarse mejor opeor al entorno.

Tradicionalmente, han existido cuatro vertientes de investigación, dentro del campo de los algoritmos evolutivos:

- **Algoritmos genéticos**. Basados en procesos de evolución darwinianos, utilizan la recombinación de los individuos de una población, y un factor de mutación de los cromosomas que codifican a dichos individuos, para el proceso de evolución.

- **Estrategias de evolución**, basadas principalmente en los cambios que se producen al nivel de los individuos.
- **Programación evolutiva**, basada principalmente en los cambios producidos a nivel de las especies.
- **Programación genética**. Es un caso particular de los algoritmos genéticos donde los cromosomas codifican estructuras complejas como árboles.

En este trabajo, nos centraremos en los algoritmos desarrollados por la vertiente de *algoritmos genéticos*, debido a los prometedores resultados mostrados en los trabajos relacionados con el entrenamiento de redes neuronales.

Las ventajas de usar algoritmos evolutivos para resolver problemas de búsqueda y optimización son las siguientes:

- No tienen restricciones sobre el espacio de soluciones.
- Son aplicables a multitud de problemas.
- Se programan fácilmente.
- Tienden a ser fáciles de hibridar con otras técnicas
- Hay una relación única entre lo que codifica un cromosoma y la estructura que representa.
- Pueden ser ejecutados interactivamente con otros sistemas.
- Devuelven múltiples soluciones.

No obstante, los algoritmos evolutivos presentan también ciertas desventajas:

- Tienen una base teórica débil.
- No garantizan que se pueda alcanzar la solución óptima en un tiempo razonable.

- Tienen diversos parámetros que hay que ajustar. La bondad de las soluciones obtenidas dependerá en gran parte del valor de dichos parámetros.
- En función de la evaluación de una solución, pueden llegar a ser costosos computacionalmente.

Pese a estas desventajas, los algoritmos evolutivos han obtenido buenos resultados en la resolución de multitud de problemas complejos. En la última década, han surgido numerosos trabajos de investigación que proponen diversos esquemas evolutivos como algoritmos de entrenamiento, adaptados para ciertas arquitecturas de redes neuronales. En un gran número de casos, los resultados y las conclusiones obtenidas han resultado serprometedores, superando la precisión obtenida por los algoritmos clásicos, basados en el gradiente [MAN89][PLA98] [PLA99][PLA00][DEL01].

En capítulos posteriores, haremos uso de los algoritmos evolutivos como herramienta para entrenamiento de redes neuronales recurrentes dinámicas. Debido al problema de los algoritmos clásicos de entrenamiento para este tipo de redes, cuyo principal representante es el problema del sesgo del gradiente, los algoritmos evolutivos se presentan como una herramienta adecuada, capaz de sobrepasar las soluciones locales obtenidas por losalgoritmos clásicos y, por tanto, producir mejores resultados.

La investigación realizada se centra especialmente en diversos esquemas de evolución genética. En estos esquemas, existe un conjunto de soluciones potenciales, sobre las cuales se aplican operadores de selección, recombinación y/o mutación. Cada iteración de estos algoritmos puede ser comparada con un ciclo evolutivo natural, similar al ilustrado en la Figura 1.9.

Figura 1.9. Esquema evolutivo genético

Existe un vocabulario muy extendido en el campo de los algoritmos evolutivos,que toma prestado muchos términos de la genética para designar determinados conceptos. Se denominan *fenotipos* a los propios cromosomas, *genotipos* a las soluciones representadas por los cromosomas, *genes* a las unidades de un cromosoma, y *alelos* a los posibles estados de un gen. No obstante, para diseñar correctamente un algoritmo evolutivo, es necesario definir un conjunto de métodos y criterios:

- **Representación de una solución.** Dado que un algoritmo evolutivo opera sobre vectores de una dimensión *d* dada (cromosoma), es necesario especificar una aplicación que permita transformar cada punto del dominio de las soluciones en un vector de dimensión *d* (función de representación genotipo/fenotipo).
- **Identificación de individuos no factibles.** No siempre es posible establecer una correspondencia punto a punto entre el dominio de las soluciones de un problema y el conjunto de cromosomas. Por este motivo, no todos los vectores de representación se corresponden con una solución real al problema, y resulta necesario definir un procedimiento que identifique este tipo de vectores que representan a soluciones no factibles.
- **Método de inicialización.** Este criterio se asocia al procedimiento que genera la población de individuos inicial, sobre la que opera el algoritmo.
- **Criterio de parada.** Se deben concretar las condiciones bajo las cuales se consideraque el algoritmo evolutivo ha encontrado una solución aceptable o, en su defecto, ha fracasado en la búsqueda y no tiene ningún sentido continuar.
- **Función de evaluación**. La función de evaluación permite discriminar entre los individuos que mejor y peor resuelven un problema dado.
- **Operadores genéticos.** Deben concretarse los procedimientos mediante los que se realiza la recombinación y mutación de soluciones.
- **Método de selección.** La selección permite dirigir el proceso de búsqueda hacia zonas más o menos prometedoras del espacio de soluciones, teniendo como criterio la función de evaluación de cada individuo.
- **Método de reemplazamiento.** Los criterios con los que se seleccionan los progenitores (soluciones que servirán para generar nuevos individuos mediante los operadores

genéticos) no tienen que ser necesariamente los mismos con los quese se seleccionan los supervivientes; de ahí la necesidad de especificarlos por separado.

- **Parámetros.** Por regla general, un algoritmo evolutivo necesita que se le proporcionen ciertos parámetros de funcionamiento, los cuales sirven para ajustar el grado de importancia de cada operación (selección, cruce, mutación, reemplazamiento, ...) en el proceso genético.

Los capítulos siguientes estudiarán diversos esquemas de algoritmos evolutivos que aplican los métodos y procedimientos anteriores, siguiendo estrategias diferentes sobre el proceso de evolución, con el fin de conseguir mejores soluciones. El esquema principal que rige el funcionamiento básico de cada uno de estos métodos se detalla enel Algoritmo 1.5.

En capítulos posteriores, abordaremos el problema del entrenamiento evolutivo de una red neuronal recurrente dinámica, desde la perspectiva de los problemas relacionados con la diversidad en las soluciones y la convergencia del propio modelo dealgoritmo. A continuación, estudiaremos algunas metodologías utilizadas para mejorar dichos factores.

```
1. t= 0, P(t)= Inicializar conjunto inicial de individuos
2. Evaluar P(t)
3. Mientras (Condición de parada = falso)
        3.1. t= t+1
        3.2. Asignar P'(t)= selección sobre P(t-1)
        3.3. Aplicar cruce y/o mutación en P'(t)
        3.4. Evaluar P'(t)
        3.5. Reemplazamiento de P'(t) sobre P(t-1)
4. Devolver mejor solución
```

Algoritmo 1.5. Procedimiento evolutivo genético general

Diversidad mediante el operador de cruce

Para controlar la diversidad mediante el operador de cruce, hay que tener en cuenta tanto el método de selección de padres que generarán un descendiente, como el método utilizado para recombinar varias soluciones y generar nueva descendencia. Existen diversas técnicas para *seleccionar dos padres que generen un descendiente*, de forma que la diversidad no se vea afectada negativamente:

- **Prohibición de cruce basada en ascendencia**. Trata de evitar que un cromosoma se cruce con sus familiares (padres, hermanos, descendientes y el mismo cromosoma).
- **Prevención de incesto**. Este método evita que dos padres se crucen, si son soluciones similares. Se dice que dos soluciones son similares si la distancia entre sus cromosomas es inferior a un umbral. Para codificación real, la distancia utilizada es la distancia euclídea.
- **Emparejamiento variado**. Este método trata de emparejar un padre con otra solución suficientemente distinta del mismo. Una vez realizado el emparejamiento, las soluciones son recombinadas para obtener nueva descendencia.

Si se desea aumentar la diversidad en la población, ha de procurarse que los descendientes sean lo más diferentes posible a los padres. Para aumentar la diversidad en la población mediante el operador de cruce, es necesario tener en cuenta ciertos factores:

- Si el operador de cruce genera descendientes que hacen que la varianza de la población aumente, entonces la diversidad aumentará excesivamente, pudiendo producir como resultado una búsqueda aleatoria.
- Si el operador de cruce genera descendientes que hacen que la varianza de la población disminuya, entonces el algoritmo puede tender a caer en óptimos locales, debido a una convergencia prematura de la población de soluciones.

El procedimiento heurístico más común es utilizar un operador de cruce que no influya excesivamente en la desviación típica de la población, de modo que se consiga un equilibrio adecuado entre diversidad y convergencia.

Diversidad mediante el operador de mutación

El operador de mutación ayuda a prevenir la convergencia prematura a soluciones locales. Sin embargo, una probabilidad alta en la mutación no tiene por qué solucionar el problema de la convergencia prematura. Existen propuestas para mejorar el funcionamiento de un algoritmo evolutivo, basadas en el factor de mutación. La política de actuación a seguir es la siguiente:

- Durante las primeras fases del algoritmo evolutivo, aplicar una alta probabilidad de mutación para que explorar adecuadamente el espacio de soluciones.
- Según vaya aumentando el número de iteraciones del algoritmo, disminuir la probabilidad de mutación para explotar las soluciones encontradas y obtener soluciones óptimas.

Según la política anterior, en las primeras iteraciones del algoritmo se realiza una búsqueda extensiva a lo largo del espacio de búsqueda. Una vez localizada una zona prometedora, el método de búsqueda se centra, en las iteraciones más avanzadas del algoritmo, en la explotación de la zona encontrada.

Diversidad con la separación espacial de los cromosomas

La diversidad con la separación espacial de los cromosomas trata de mantener diversas poblaciones evolucionando concurrentemente, con o sin comunicación entre ellas. Los ejemplos más comunes son los *algoritmos genéticos distribuidos* (por ejemplo, basados en modelos de isla). Podrían englobarse, en cierto modo, en este punto también algunos algoritmos

multimodales basados en nichos, ya que dividen la población en conjuntos de soluciones, cruzando individuos de distintos conjuntos, como por ejemplo el algoritmo Clearing.

Diversidad mediante adaptación

Las técnicas de diversidad mediante adaptación tratan de mejorar la diversidad de la población adaptando parámetros, operadores genéticos, función de evaluación, etc., durante la ejecución de un algoritmo, con el fin de aumentar la diversidad o disminuirla.

Diversidad mediante estrategias de reemplazamiento

Dependiendo de la estrategia a utilizar, la elección entre un método de reemplazamiento y otro puede llevarnos a conseguir una convergencia prematura, o incluso contribuir para un aumento de la diversidad. Las técnicas que utilizan el método de reemplazamiento para controlar la diversidad en la población pueden englobarse en dos vertientes básicas:

- Métodos basados en la similitud entre los genes de cromosomas. Generalmente el reemplazamiento suele darse con el individuo más parecido (por ejemplo, método DDA).
- Métodos de competición entre cromosomas hijos, padres u hermanos pertenecientes a la misma familia (por ejemplo, family competition).

Combinando una elección inteligente de los operadores de selección, cruce, mutación y reemplazamiento, puede conseguirse un comportamiento adecuado para un algoritmo evolutivo, en la resolución de un problema concreto, realizando una correcta exploración y explotación del espacio de soluciones.

4. Métodos de optimización multiobjetivo.

La mayoría de los problemas de optimización en ingeniería suelen tener múltiples criterios susceptibles de ser optimizados. Generalmente, estos criterios son contrapuestos e independientes entre sí.

Los métodos de optimización multiobjetivo [COE99][COE00][COE02] [COL03][FON98][OSY85] engloban un conjunto de técnicas que tratan de solucionar problemas con este tipo de características. Formalmente, podemos definir un problema de optimización multiobjetivo como:

$$Max/Min\,(\vec{f}(\vec{x}))$$

sujeto a las restricciones:

$$g_i(\vec{x}) \geq 0;\ i = 1 \ldots p$$
$$h_i(\vec{x}) = 0;\ i = p \ldots m$$

El vector f está compuesto por k funciones objetivo a optimizar. Por otra parte, el vector x contiene un total de q variables que definen la solución al problema.

$$\vec{f}(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots f_k(\vec{x}))$$
$$\vec{x} = (x_1, x_2, x_3, \ldots x_q)$$

Siendo f_i, f_j dos objetivos notablemente contradictorios, el concepto de solución óptima engloba a todas aquellas soluciones tales que no exista una solución cuyas funciones objetivo sean mejores que las funciones objetivo de la solución óptima, paratodas las funciones objetivo f_i, f_j. Este criterio fue introducido por primera vez en 1896 por Wilfredo Pareto.

Un vector de variables solución, x^*, se dice que es Pareto-optimal si se cumple que no hay ninguna otra solución al problema tal que todos los valores de lasfunciones objetivo en esa solución sean mejores que los valores de las funciones objetivo en x^*.

Al conjunto de todas las soluciones que satisfacen el criterio de Pareto se llama *frontera de Pareto*. Los algoritmos de optimización multiobjetivo basados en el criterio de Pareto

realizan una búsqueda a lo largo de todo el espacio de soluciones para encontrar la frontera de Pareto. Usualmente, los algoritmos de optimización multiobjetivo basados en este esquema implementan el criterio de Pareto mediante los conceptos de *dominancia* y *cobertura* entre soluciones. Formalmente, suponiendo un problema de minimización, diremos que una solución x domina a otra solución y si, y sólo si:

$$\forall i \in \{1,2,\ldots,k\} \;\; f_i(x) \leq f_i(y) \;\wedge\; \exists j \in \{1,2,\ldots,n\} \,|\, f_j(x) < f_i(y) \qquad (1.27)$$

Según la ecuación 1.27, diremos que x domina a y si los valores de todas las funciones objetivo de x son mejores o iguales a las de la solución y, y al menos existe un objetivo cuyo valor es mejor en la solución x que en la solución y.

Al relacionar el concepto de dominancia con el criterio de optimalidad de Pareto, se puede deducir que el conjunto de la frontera de Pareto estará formado por todas las soluciones no dominadas del problema.

Por otra parte, el concepto de cobertura relaja la idea de dominancia. La cobertura entre soluciones puede ayudar a mejorar el proceso de búsqueda del algoritmo multiobjetivo. Suponiendo un problema de minimización, diremos que una solución x cubre a otra solución y si, y sólo si:

$$\forall i \in \{1,2,\ldots,k\} \;\; f_i(x) \leq f_i(y) \qquad (1.28)$$

Según la ecuación 1.28, diremos que x cubre a y si los valores de todas las funciones objetivo de x son mejores o iguales a las de la solución y.

Durante la última década se han desarrollado múltiples propuestas de algoritmos evolutivos para problemas multiobjetivo [BEN96][COE02][DEB99][DEB01][DIA02][VEL98] [VEL00], bien basados en el criterio de Pareto, o bien basados en otro tipo de criterios. Las técnicas actuales de optimización multiobjetivo tienden a utilizar el criterio de Pareto en el diseño del algoritmo de optimización.

Generalmente, los algoritmos no basados en el criterio de Pareto tienen ciertas características como la fácil implementación o la eficiencia [FON95][FON98]. Estas propuestas tratan de construir una función de agregación de los objetivos, mediante la asignación de un

valor de importancia a cada criterio a optimizar (orden lexicográfico), evolución conjunta con la ponderación de cada criterio, asignación de sub-objetivos en cada criterio, etc. Algunos ejemplos de este tipo de algoritmos son VEGA, MOWGA, o algoritmos de Goal Programming.

En los últimos años, el esfuerzo investigador ha ido dirigido, en gran medida, al desarrollo y mejora de los algoritmos evolutivos basados en el criterio de Pareto [COE02][DEB02][ZIT98][ZIT99][ZIT00][ZIT01]. Algunas propuestas de las más conocidas son los algoritmos de Fitness Sharing, MOGA, NPGA, PAES, M-PAES, NSGA, SPEA, NSGA-II y SPEA-II. Debido a los prometedores resultados obtenidos por SPEA-II y NSGA-II, en el desarrollo de nuestro trabajo haremos uso de los mismos para entrenamiento y optimización de redes neuronales recurrentes.

Los algoritmos evolutivos se muestran como una herramienta adecuada para resolver problemas multiobjetivo, ya que su esquema está estructurado para operar sobre un conjunto de soluciones de forma simultánea, llamado población. Esto permite poder obtener una frontera de Pareto en una única ejecución del algoritmo. Además, mediante los conceptos de dominancia y cobertura, podemos ordenar parcialmente un conjunto de soluciones, de modo que se pueda discriminar de forma simple entre las soluciones óptimas y no óptimas de la población, en el algoritmo evolutivo.

El papel desempeñado por los algoritmos de optimización multiobjetivo es de gran relevancia para la investigación que se detalla a lo largo de los siguientes capítulos. Haremos uso de estos métodos para encontrar la topología óptima de una red neuronal recurrente dinámica, y entrenar el conjunto de redes del frente de Pareto, de forma simultánea.

5. Predicción de Series Temporales mediante Redes Neuronales Recurrentes Dinámicas.

En los capitulos que siguen, vamos a contrastar los resultados obtenidos en aplicaciones para predicción de series temporales. Una serie temporal se puede definir como un conjunto de

datos o valores, obtenidos a partir de observaciones realizadas sobreun fenómeno concreto a lo largo del tiempo [BRA99][BRO86] [TSA02][VIÑ01].

El objetivo del análisis de series temporales es el conocimiento de su patrón de comportamiento. Posteriormente, podrá realizarse predicciones sobrela evolución de los datos, bajo el supuesto de que las condiciones que afectan a la serie temporal no varíen a lo largo del tiempo.

No obstante, si conociendo los valores anteriores y actuales de una serie temporal se pudiesen predecir los valores futuros sin error, estaríamos frente a un proceso determinista y su estudio no tendría ningún interés. El hecho que motiva el estudio delas series temporales es la aparición de fenómenos de naturaleza aleatoria, parcial o total, los cuales hacen que la predicción de los valores futuros de la serie sólo puedan realizarse mediante métodos probabilísticos.

El análisis y predicción de series temporales es un problema ampliamente estudiado, el cual encuentra aplicaciones dentro de múltiples campos [DIG90][LEN98][POL99], tales como el procesado de señales, comunicaciones, biología, astronomía, meteorología, control de procesos, sismología, medicina, economía, etc.

Las series temporales se caracterizan porque su evolución temporal no depende explícitamente de la variable tiempo, sino también de los valores de la misma serie, tomados en instantes anteriores al actual. Ocasionalmente, también puede existir dependencia con otras variables temporales externas. La suposición que permite el análisis y la predicción de series temporales es que el valor de la serie en el instante t depende del valor de las observaciones tomadas en instantes anteriores, y de un error introducido en el sistema, de origen desconocido.

Dependiendo del número de valores de la serie utilizados para el modelado, se dirá que es de orden 1 si únicamente se considera el valor inmediatamente anterior, de orden 2 si se consideran los dos valores inmediatamente anteriores, y de orden k si se consideran los k valores inmediatamente anteriores:

$$y(t)=F(y(t-1),y(t-2),\ldots,y(t-k)) \tag{1.29}$$

La notación utilizada en la ecuación 1.29 se resume en el siguienteesquema:

- $y(t) \in \Re^n$ es el valor de la serie temporal observado en el instante t;
- $F: \Re^n x ...^{(k)} ... x \Re^n \rightarrow \Re^n$ es una función desconocida, cuyo dominio es el valor de las observaciones de la serie temporal en los k instantes anteriores, y cuyo rango es el valor de las observaciones de la serie en el instante actual.
- $e(t)$ es el error que se produce en el modelado de la serie en el instante t, debido a perturbaciones o a variables no medibles, externas al sistema.

El análisis de series temporales (utilizando las metodologías dásicas para resolver este problema, esencialmente Box-Jenkins [VIÑ01]), permite encontrar un modelo lineal que aproxima los datos de la serie. Este modelo permitirá realizar predicciones sobre los valores futuros. La metodología que permite encontrar este modelo se puede simplificar en tres pasos:

- **Análisis de componentes de los datos**. En este primer paso, la serie temporal es estudiada de modo que se puedan identificar cuatro componentes básicas: Tendencia, estacionariedad, periodicidad, y residuos.
- **Generación del modelo**. En este segundo paso, se genera un modelo para la serie temporal, basado en el estudio realizado en el primer paso.
- **Predicción y validación**. Una vez generado el modelo que aproxima la serie temporal, se procede a su validación mediante la predicción de valores futuros, y la comparación de los mismos con los datos reales.

No obstante, el modelado y predicción de series de datos medianteel uso de las técnicas clásicas presenta ciertas desventajas:

- Este tipo de modelos sólo son aplicables cuando la serie temporal es estacionaria (su media, varianza y autocorrelación deben ser similares a lo largodel tiempo).
- No permiten modelar relaciones no lineales.

- Es recomendable disponer de, al menos, cincuenta valores de la serie temporal, para poder aproximar los parámetros debidamente.
- Se asume que el valor de los parámetros es constante a lo largo del tiempo.
- El horizonte de predicción no puede ser muy alto en los modelos que contengan componentes de medias móviles, debido a la suposición de que el error en la predicción es cero.

Debido a estas limitaciones [TER94], es necesario utilizar otros modelos para predicción de series temporales, entre los que están los sistemas difusos, métodos de regresión no lineal, redes neuronales, etc.

Durante el desarrollo de los siguientes capítulos, abordaremos el problema de la predicción de series temporales, utilizando RNRD. Las ventajas introducidas por la utilización de modelos neuronales, para predicción de series de datos, son numerosas:

- No se requiere que la serie a tratar sea estacionaria, dada su condición de aproximadores universales;
- Modelan adecuadamente la no linearidad;
- Es posible realizar predicciones a largo plazo (solventando así el problema del horizonte de predicción del modelo Box-Jenkins); y
- No se necesita una cantidad de datos elevada para entrenar una red neuronal.

La utilización de diferentes modelos de redes neuronales ha proporcionado resultados muy prometedores, en el campo de la predicción y el modelado de series de datos. Como ejemplo, podemos citar el uso de redes Perceptrón multicapa, RBF, RBF recurrentes, SOM, FIR, RNRD, LSTM [LIN92][KAA96][AUS99][DAV99][DAN01][MIC03][SAN02][ZEM03].

Los apartados siguientes exponen cómo utilizar modelos de red feedforward y recurrentes, para resolver el problema de la predicción de series de datos. En la siguiente sección, la explicación del modelado y predicción de series de datos mediante redes neuronales de tipo feedforward, ayudará a ilustrar las diferentes limitaciones de este tipo de modelos en este problema.

Del mismo modo, servirá como justificación para utilizar redes neuronales recurrentes dinámicas en problemas de predicción de series de datos, tal como desarrollaremos en los siguientes.

5.1. Redes neuronales feedforward para predicción de series de datos.

La utilización de modelos de redes neuronales *feedforward*, para problemas de predicción de series temporales, puede considerarse como una adaptación de la ecuación 1.29 a la ecuación que rige el comportamiento de la red [DAV99][KAA96][LIN92]. Tomando como modelo una estructura de red feedforward, este apartado detalla cómo realizar tal adaptación. El procedimiento puede ser fácilmente extrapolado a modelos de red *feedforward* con características similares al Perceptrón multicapa o redes RBF, por ejemplo.

De acuerdo con la ecuación 1.29, el objetivo es encontrar un orden k para modelar correctamente la serie, y calcular una función $F(\cdot)$ tal que el valor de la serie $Y(t)$ pueda ser calculado a partir de los k valores anteriores: $F(Y(t-1), Y(t-2), \ldots, Y(t-k))$.

Considerando una red neuronal de tipo *feedforward*, el problema puede ser planteado de modo que los valores $Y(t-1), Y(t-2), \ldots, Y(t-k)$ sean utilizados como patrones de entrada a la red, y el valor $Y(t)$ sea la salida asociada a dicho patrón. De este modo, la función $F(\cdot)$ se identifica como la función que debe aprender la red neuronal durante el proceso de entrenamiento.

Esta idea se expresa en la Figura 1.10, donde cada entrada a la red, $X_i(p)$, se corresponde con el valor de la serie temporal $Y(p-i)$. La salida de la red, $O(p)$, corresponde a la aproximación que la red entrenada realiza para el valor de la serie en el instante t, $Y(t)$.

Figura 1.10. Ejemplo de uso de redes neuronales feedforward para predicción de series temporales

Existen dos problemas fundamentales en el modelado de una serie temporal, mediante el uso de redes neuronales de tipo *feedforward*:

- **Selección del número de entradas a la red** Este problema está relacionado con el cálculo del orden del ajuste, *k*. Generalmente, el orden del ajuste se calcula mediante técnicas estadísticas, como por ejemplo en la metodología Box-Jenkins. Sin embargo, cuando aumenta la complejidad de la serie, tales técnicas sevuelven ineficaces y resulta necesario recurrir a otro tipo de herramientas, como análisis de Fourier, análisis de componentes principales, regresiones no lineales, etc.

  No obstante, cuando estas técnicas no producen resultados adecuados, se suele acudir a la experimentación mediante el método de ensayo y error.

- **Ajuste de la función F(·).** Este problema se identifica con el problema del entrenamiento de la red neuronal. Además, el buen aprendizaje de la red puede verse dificultado mediante el uso de entradas a la red redundantes o irrelevantes. Explicaremos esta situación mediante un ejemplo:

  Supongamos que se ha seleccionado un orden de ajuste $k=3$. En este caso, según la ecuación 1.29, las entradas a la red deben corresponderse con los valores de la serie $Y(p-1), Y(p-2), Y(p-3)$, para poder generar la salida $O(p)$. Sin embargo, las dependencias posibles que pueden tener lugar en la serie de datos para un orden de ajuste 3 son las siguientes:

  - $O(p)$ depende de $Y(t-1), Y(t-2)$ y $Y(t-3)$,
  - $O(p)$ depende de $Y(t-1)$ y $Y(t-3)$,
  - $O(p)$ depende de $Y(t-2)$ y $Y(t-3)$.
  - $O(p)$ depende de $Y(t-1)$ y $Y(t-2)$,
  - $O(p)$ depende de bien únicamente $Y(t-1)$, $Y(t-2)$ o $Y(t-3)$

  Si la dependencia real de la serie temporal fuese

  $$Y(t) = F(Y(t-1), Y(t-3)),$$

entonces se estaría introduciendo una entrada irrelevante a la red, *Y(t-2)*, dificultando de esta forma su correcto entrenamiento.

La relevancia de este tipo de problemas aumenta cuando la serie temporal contiene relaciones a largo plazo entre los datos. De este modo, la complejidad del problema aumenta considerablemente, necesitando un número de experimentos y tests que puede crecer en orden exponencial.

Adicionalmente, las redes neuronales de tipo *feedforward* no tienen la capacidad de modelar series temporales donde el orden del ajuste y las dependencias entre los datos varíen a lo largo del tiempo; es decir, cuando la serie temporal presenta un comportamiento dinámico. En este caso, sería necesario disponer de un conjunto de redes neuronales para cada comportamiento, y de un mecanismo que nos permitiese conocer cuándo se producen dichas variaciones.

Para solventar las carencias comentadas, nuestro trabajo abordará el problema de la predicción de series temporales mediante el uso de redes neuronales recurrentes dinámicas. Este tipo de redes reúne una serie de características que permiten solucionar el problema, sin contar con las limitaciones existentes en redes feedforward:

- Las conexiones recurrentes de la red permiten memorizar información acerca de las entradas anteriores a la red, a corto y largo plazo.
- Si la serie de datos presenta comportamiento dinámico, la red puede ser fácilmente adaptada al nuevo comportamiento, debido a la flexibilidad introducida, para estos caso, por las conexiones recurrentes.
- El diseño de la estructura de entradas/salidas a la red se simplifica, con respecto a redes neuronales de tipo feedforward.

El apartado siguiente detalla cómo abordar el problema de la predicción de series temporales, mediante el uso de redes neuronales recurrentes dinámicas.

5.2. Redes neuronales recurrentes dinámicas para predicción de series de datos.

Este apartado detalla el uso de modelos de redes neuronales recurrentes dinámicas para predicción de series de datos [AUS99][DAN01][MIC03][PEG01][ZEM03]. Atendiendo a las arquitecturas de redes estudiadas en el apartado 1, sabemos que el estado de la red en un instante dado es función de las entradas a la red, presentadas en instantes anteriores. Llamando H a tal función, podemos decir que la salida de la red neuronal en un instante t será función del estado de la red y de las entradas en el mismo tiempo t:

$$O(t)=G(X(t),H(X(t-1),X(t-2),...))=F(X(t),X(t-1),X(t-2),...)$$

Llegados a este punto, podemos asumir que la entrada a la red neuronal, $X(t)$, sea el valor de la observación de la serie temporal en ese mismo instante, y que la salida a modelar sea el valor de la observación en el instante $t+1$. El esquema del sistema modelado es el que se muestra en la figura 1.11.

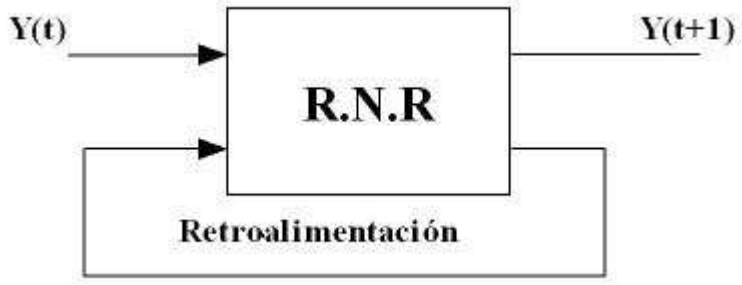

Figura 1.11. Esquema de una red neuronal recurrente dinámica, para resolver problemas de series temporales

La ecuación que rige el comportamiento del sistema es la siguiente:

$$Y(t+1)=F(Y(t),Y(t-1),Y(t-2),...)$$

Para modelar una serie temporal mediante redes neuronales recurrentes, estableceremos el valor de la serie en el instante t como entrada a la red en el mismo tiempo t. Estableceremos, a su vez, el valor de salida de la red como el valor de la serie temporal en el instante $t+1$. De este

modo, el entrenamiento de la red neuronal deberá minimizar una medida de error entre la salida de la red, para cualquier instante *t*, y el valor de la serie temporal en el instante *t+1*.

Una vez modelada la serie temporal mediante una red recurrente, para realizar la predicción de los valores futuros bastará con utilizar las propias salidas de la red como entradas en el tiempo siguiente, de modo que se sigan cumpliendo las ecuaciones iniciales supuestas:

$$Y(t+1)=F(Y(t),Y(t-1),...)$$
$$Y(t+2)=F(Y(t+1),Y(t),Y(t-1),...)$$
$$...$$

La Figura 1.12 muestra la estructura del sistema planteado. Inicialmente, las entradas de la red se corresponden con los valores iniciales de la serie, *Y(1), Y(2), …, Y(T-1)*. Suponiendo que se desea predecir los datos a partir de *Y(T+1), Y(T+2), …*, se habilitará la retroalimentación entre las salidas de la red y las entradas, asumiendo que la salida de la red se corresponde con *Y(T), Y(T+1), Y(T+2), …*, respectivamente.

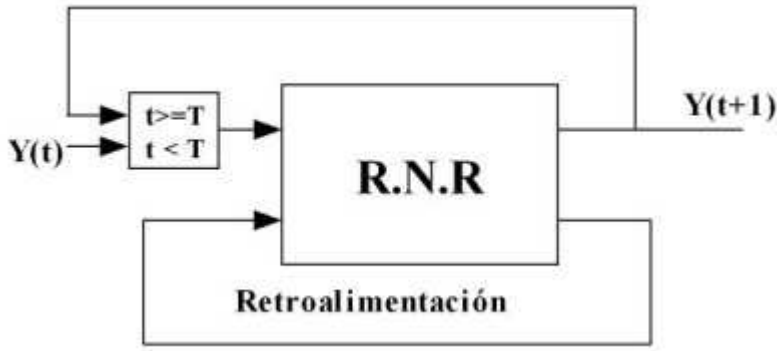

Figura 1.12. Esquema de entradas y salidas de una red neuronal recurrente dinámica, para predicción de series temporales

Capítulo 2

Métodos evolutivos para entrenamiento de Redes Neuronales Recurrentes Dinámicas.

1. Introducción.

En este capítulo se introducen las herramientas que hemos desarrollado para entrenar Redes neuronales Recurrentes Dinámicas (RNRD), utilizando algoritmos evolutivos clásicos. El objetivo perseguido consiste en encontrar una configuración o conjunto de configuraciones de algoritmos evolutivos que proporcionen buenos resultados para entrenamiento de RNRD, en los problemas de predicción de series temporales que empleamos como contraste. Utilizaremos diversos modelos de algoritmos genéticos clásicos [BAC96][ESH93][GOL89][SCH00] con estrategia evolutiva generacional, estacionaria, y un modelo mixto propuesto por nosotros en un trabajo previo. Además, adaptaremos otros esquemas evolutivos (Clearing y CHC) para poder aplicarlos al problema del entrenamiento de una red neuronal recurrente dinámica, suponiendo una estructura topológica de red predefinida y fija.

Los algoritmos evolutivos suponen una alternativa a los métodos basados en el gradiente y la propagación de errores, incorporando ciertas ventajas: Pueden utilizarse cuando la información del gradiente no está disponible o es computacionalmente costosa. Estos casos suelen darse cuando la función de activación de las neuronas no es diferenciable o es discontinua. Además, los algoritmos genéticos pueden ser adaptados fácilmente para entrenar una amplia gama de redes neuronales, incluyendo estructuras recurrentes complejas.

El problema del entrenamiento evolutivo de una red neuronal ha sido estudiado ampliamente en la última década. Atendiendo al tipo de codificación utilizada, podemos clasificar los métodos existentes en:

- **Métodos que utilizan codificación entera.** El cromosoma que representa a la red neuronal está formado por genes cuyos alelos son números enteros. Este tipo de codificación está orientado a problemas donde la función de activación es peculiar y se hace necesario la utilización de números enteros para los pesos de la red, o en problemas donde los pesos de la red han de ser necesariamente discretos [PLA98][PLA00].
- **Métodos que utilizan codificación binaria.** Las restricciones de algunos problemas de aplicación llevan al diseñador a utilizar codificación binaria para representar las conexiones de una red neuronal en un cromosoma. Este tipo de métodos se utilizan en propuestas que requieren implementación en hardware de la propia red neuronal, optimización del número de conexiones de la red, o cuando los pesos de la red son discretos y únicamente pueden tomar los valores {0, 1} [PLA99].
- **Métodos que utilizan codificación real.** Las propuestas que utilizan codificación real son las más extendidas de las tres clasificaciones. En estas, los pesos de la red neuronal son codificados en un cromosoma, donde los alelos de los genes son números reales [DEL00][DEL01][GAR02].

Independientemente de la codificación utilizada para representar una red neuronal en un algoritmo evolutivo, los diferentes modelos y técnicas utilizados para el entrenamiento son muy numerosos. Entre estos podemos destacar los métodos basados en búsqueda tabú, descenso en colinas evolutivos, enfriamiento simulado, estrategias de evolución y algoritmos genéticos.

Es conocido que uno de los temas de mayor interés, considerando un algoritmo evolutivo, es conseguir un equilibrio adecuado entre diversidad y convergencia [RAW91]. Mediante este equilibrio se consigue explorar y explotar adecuadamente el espacio de soluciones. Si el algoritmo evolutivo diseñado no posee una relación adecuada entre diversidad y convergencia, es probable que el proceso de búsqueda quede atrapado en óptimos locales y no consiga encontrar una solución óptima global.

Abordaremos el problema del entrenamiento de una RNRD desde la perspectiva de la evolución de la diversidad y convergencia de cada modelo evolutivo. Para ello, estudiaremos el efecto de la combinación de diferentes esquemas y operadores genéticos en diferentes modelos evolutivos. Concretamente, estudiaremos el efecto de los operadores en tres modelos de algoritmos genéticos (generacional, estacionario y mixto). Además, utilizaremos técnicas de separación espacial de cromosomas y diversidad mediante cruce y reemplazamiento, para descubrir qué tipo de estrategias y modelos evolutivos son más eficaces para entrenar una red neuronal recurrente dinámica. Este tipo de técnicas no habían sido aplicadas con anterioridad para entrenar RNRD.

El funcionamiento de los algoritmos evolutivos considerados será validadomediante las capacidades que muestren los diferentes modelos de red para aprender problemas de predicción de series temporales. El apartado siguiente introduce los conjuntos de datos utilizados para realizar la validación.

2. Validación de los algoritmos de entrenamiento evolutivos: Problemas de predicción de series temporales. El benchmark CATS.

El problema de la predicción de series temporales despierta el interés de diferentes tipos de áreas de conocimiento, tales como medicina, economía, electrónica, ingeniería industrial, etc. Es por ello que actualmente se realizan numerosos esfuerzos para mejorar o crear técnicas y metodologías que permitan solucionar los problemas existentes en el campo de la predicción de series de datos.

Las redes neuronales artificiales han obtenido resultados muy prometedores en aplicaciones de predicción de series de datos [DAV99]. En concreto, las características de la estructura topológica de RNRD hacen de estos modelos una potente herramienta para ser aplicada en este tipo de problemas [AUS99][DAN01].

No obstante, los algoritmos clásicos de entrenamiento para redes recurrentes dinámicas, cuya limitación principal es el problema del sesgo del gradiente, tienden a quedar estancados en soluciones óptimas locales e impiden que se obtengan mejores resultados con estos modelos de red. De este modo, las dependencias temporales que la red debe aprender a largo plazo, en problemas de predicción de series de datos, no son correctamente modeladas en la estructura interna de la red recurrente.

Los problemas de predicción de series temporales permiten validar los algoritmos propuestos en la investigación realizada, en contraposición a los algoritmos clásicos de entrenamiento. Podremos diferenciar los casos en los que se aprenden correctamente las relaciones temporales entre los datos y, por tanto, se superan soluciones óptimas locales, y los casos en los que no.

El conjunto de datos principal que utilizaremos a lo largo de la investigación es el benchmark *CATS* de series temporales [LEN04]. Este conjunto de datos fue propuesto por primera vez en el año 2004, en una competición de modelos para resolver problemas de series temporales (sesión especial en la Conferencia Internacional de Redes Neuronales (IJCNN), en colaboración con la Sociedad Europea de Redes Neuronales, el grupo de investigación en Machine Learning de la Universidad de Lovaina (Bélgica), y el centro de investigación de Redes Neuronales de la Universidad tecnológica de Helsinki). La Figura 2.1 ilustra la evolución temporal de la serie de datos CATS.

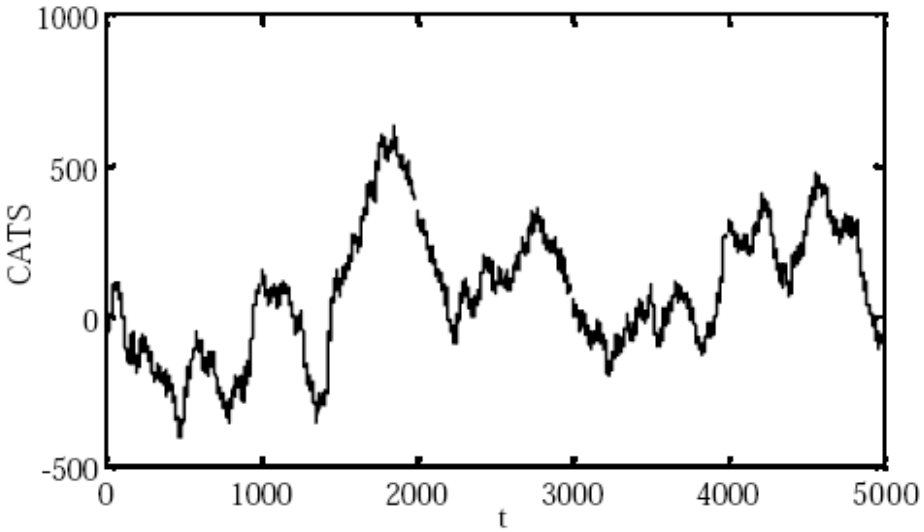

Figura 2.1. Serie temporal del benchmark CATS

El benckmark CATS es una serie temporal de 5000 valores, donde cada componente es un número real. Está dividida en 5 subconjuntos:

- **CATS1**: componentes 1-1000 de la serie temporal. Se proporcionan los valores para las componentes 1-980, y hay que predecir los valores de las componentes restantes (981-1000). Estos valores se ilustran en la Figura 2.2 (zona punteada).

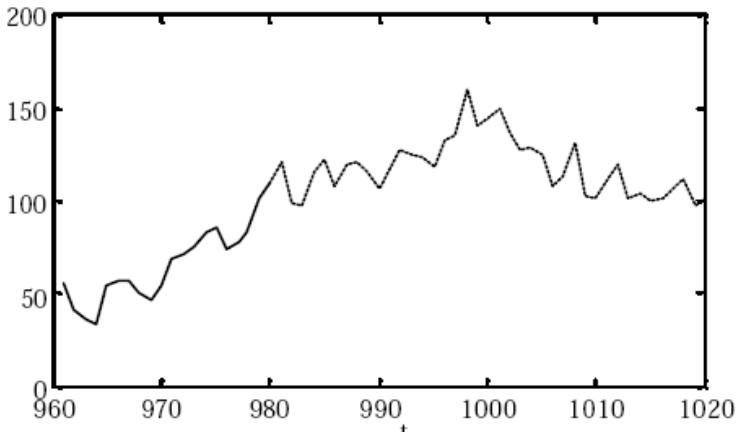

Figura 2.2. Datos a predecir en el subproblema CATS1

- **CATS2**: componentes 1001-2000 de la serie temporal. Se proporcionan los valores para las componentes 1001-1980. Los valores a predecir son los correspondientes a las componentes 1981-2000 (Figura 2.3).

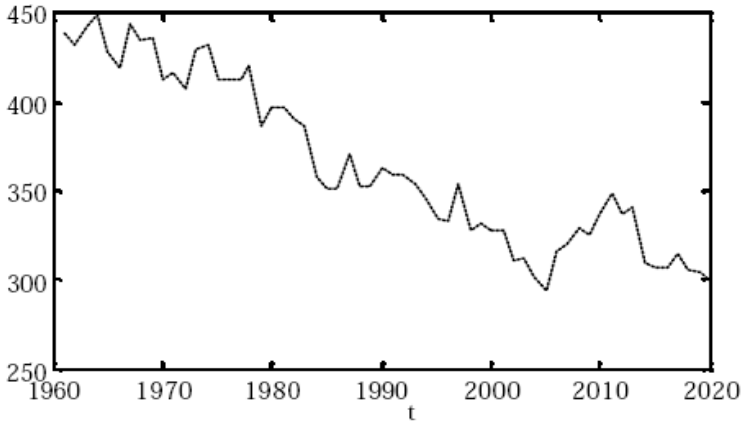

Figura 2.3. Datos a predecir en el subproblema CATS2

- **CATS3**: componentes 2001-3000 de la serie temporal. Se proporcionan los valorespara las componentes 2001-2980. Se predicen los valores restantes (Figura 2.4).

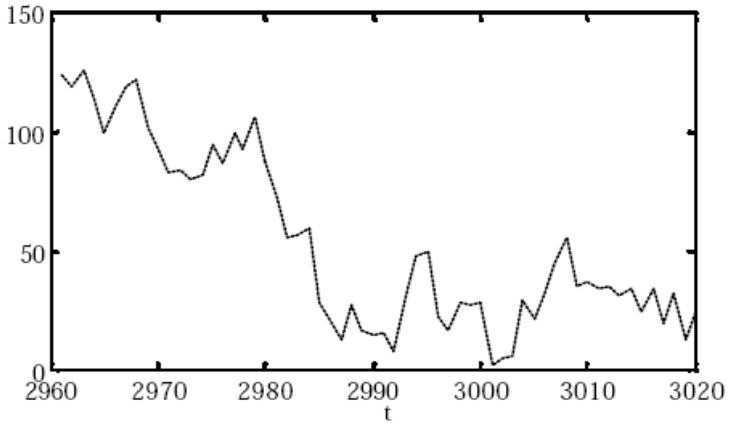

Figura 2.4. Datos a predecir en el subproblema CATS3

- **CATS4**: componentes 3001-4000 de la serie temporal. Se proporcionan los valorespara las componentes 3001-3980. Se predicen los valores restantes (Figura 2.5).

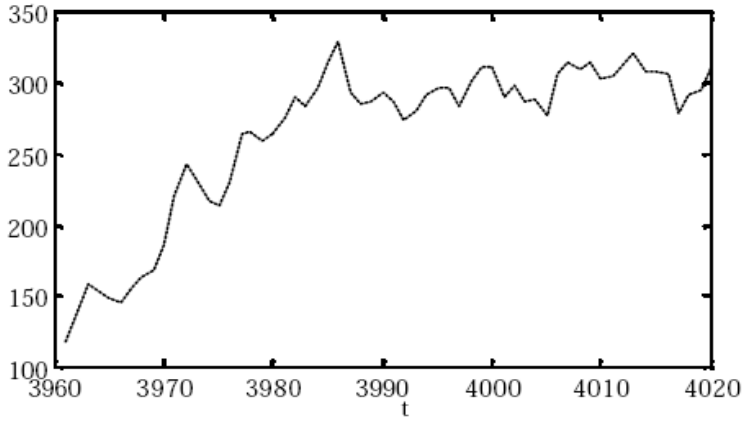

Figura 2.5. Datos a predecir en el subproblema CATS4

- **CATS5**: componentes 4001-5000 de la serie temporal. Se proporcionan los valorespara las componentes 4001-4980. Se predicen los valores restantes (Figura 2.6).

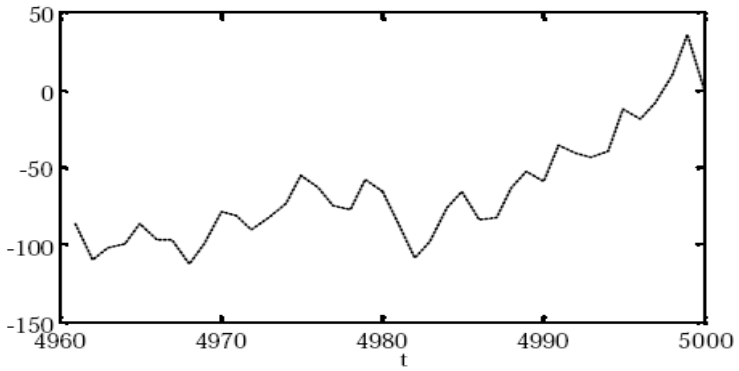

Figura 2.6. Datos a predecir en el subproblema CATS5

Los valores que componen la serie temporal, así como una mayor información relacionada con el benchmarck CATS, pueden obtenerse de la URL http://www.cis.hut.fi/~lendasse/competition/competition.html

Durante el desarrollo de los capítulos, los subconjuntos en los que se divide la serie temporal serán tratados como series temporales independientes. Los usaremos para validar los resultados de los algoritmos de entrenamiento propuestos. Finalmente, el capítulo 5 comparará la eficacia de los metodos desarrollados, frente a los resultados producidos por el resto de técnicas aplicadas al problema.

3. Entrenamiento evolutivo de RNRD.

En este apartado, presentamos los resultados de utilizar diferentes modelos de algoritmos genéticos [BAC96][DEL01] para entrenar RNRD con estructura topológica predefinida y fija. Nos centraremos en el estudio del equilibrio entre diversidad y convergencia de cada algoritmo, y en su adecuación para entrenar las diferentes arquitecturas de redes neuronales consideradas.

El objetivo que perseguimos es encontrar una configuración de algoritmo evolutivo y un conjunto de operadores genéticos que presenten un comportamiento estable para entrenar RNRD en el conjunto completo de funciones de test utilizado, y que sea extrapolable a otros problemas sin necesidad de realizar excesivos cambios. Para ello, estudiaremos el comportamiento de

diferentes modelos genéticos clásicos (generacional, estacionario y mixto), y propondremos nuevos métodos evolutivos para entrenar RNRD, los cuales solventan el problema de la diversidad en la población. Estos métodos están basados en algoritmos genéticos multimodales y en la estrategia CHC, la cual hemos modificado para poder utilizar codificación real.

En primer lugar, la sección 3.1 explica los mecanismos de representación de los diferentes modelos de RNRD estudiados en el capítulo primero. Seguidamente, la sección 3.2 expone la función de evaluación, que servirá para guiar el proceso de búsqueda de cada algoritmo evolutivo.

La experimentación comienza en la sección 3.3, donde analizamos el efecto de diferentes operadores de selección, cruce, mutación y reemplazamiento en modelos clásicos de algoritmos genéticos. A continuación, la sección 3.4 evalúa el efecto de modelos evolutivos que mejoran la diversidad de la población mediante la separación espacial de cromosomas. La sección 3.5 analiza el efecto de la estrategia de evolución CHC, la cual hemos modificado para poder utilizar codificación real y así entrenar los modelos de redes neuronales estudiados. Finalmente, la sección 3.6 realiza un estudio comparativo de todos los modelos de entrenamiento evolutivos considerados.

3.1. Representación.

La representación de una red neuronal es un aspecto clave a tener en cuenta al abordar el problema del entrenamiento mediante algoritmos evolutivos. Para que un cromosoma represente una red neuronal es necesario que dicho cromosoma contenga la información de los pesos dentro de su estructura. En nuestro trabajo, un gen perteneciente a un cromosoma está unívocamente asociado a una conexión concreta de la red neuronal. Los alelos posibles de un gen serán los valores posibles para el peso de la conexión asociada. La codificación utilizada es la *codificación real* [DEL01][HER98][LOZ04][HER05]. A continuación expondremos la asociación *genotipo-fenotipo*, para las arquitecturas de redes expuestas en el capítulo primero.

Representación del modelo de Elman

La red recurrente de Elman está formada por tres capas de neuronas: entrada, salida, y capa oculta (capítulo 1.1). Utilizaremos la siguiente notación:

- $\mathbf{V_{ij}}$, es el peso asociado a la conexión entre la neurona de entrada *j* y la neurona oculta *i*.
- $\mathbf{U_{jr}}$, es el peso asociado a la conexión recurrente entre la neurona oculta *r* y la neurona oculta destino *j*.
- $\mathbf{W_{kj}}$, es el peso asociado a la conexión entre la neurona oculta *j* y la neurona de salida *k*.

La codificación de una red recurrente de Elman se realiza mediante la asignación de cada conexión de la red a un gen específico dentro del cromosoma. El alelo de dicho gen contiene el valor del peso de la conexión asociada.

La figura 2.7 muestra la estructura de un cromosoma que codifica una red recurrente de Elman con *n* entradas, *h* neuronas ocultas, y *o* neuronas de salida.

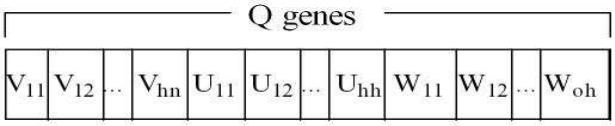

Figura 2.7. Codificación general de una red recurrente de Elman

Un cromosoma que codifique una red neuronal recurrente completa con *n* neuronas de entrada, *h* neuronas ocultas y *o* neuronas de salida, contendrá un total de *Q* genes, donde *Q* viene dado por:

$$Q = n \cdot h + h \cdot h + o \cdot h \tag{2.1}$$

La figura 2.8 muestra un ejemplo de la codificación de una red de Elman con una entrada, una salida, y dos neuronas ocultas. Dado un cromosoma $C=(c_1, c_2, \ldots, c_Q)$ que codifica una red neuronal recurrente de Elman, el Algoritmo 2.1 muestra el esquema de asignación de conexiones a genes.

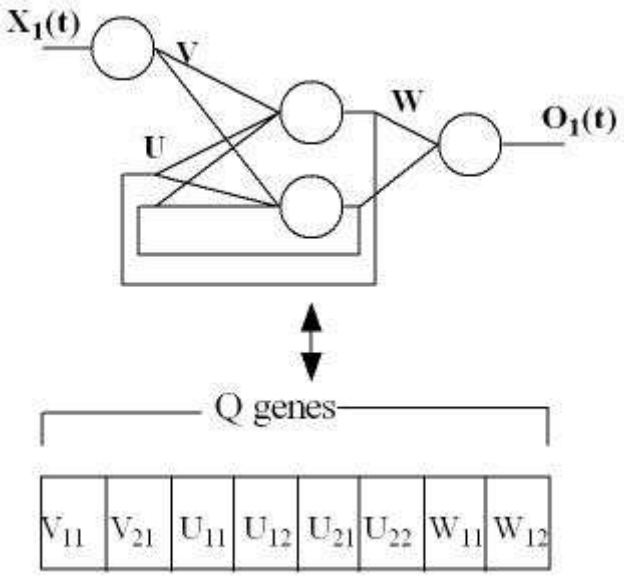

Figura 2.8. Ejemplo de codificación de una red recurrente de Elman con una entrada, una salida y dos neuronas ocultas

1. asignar m= 1

2. Para cada neurona de entrada i=1..n

2.1. para cada neurona oculta j=1..h

2.2. Asignar peso V_{ij} al gen c_m

2.3. Asignar m=m+1

3. Para cada neurona oculta i=1..h

3.1. para cada neurona oculta j=1..h

3.2. Asignar peso U_{ij} al gen c_m

3.3. Asignar m= m+1

4. Para cada neurona de salida i=1..o

4.1. para cada neurona oculta j=1..h

4.2. Asignar peso W_{ij} al gen c_m

4.3. Asignar m= m+1

Algoritmo 2.1. Procedimiento de asignación de conexiones de una red deElman a genes de un cromosoma

Representación del modelo recurrente completo

La red recurrente Completa está formada por dos capas de neuronas: entrada, y capa de neuronas ocultas/de salida (capítulo 1.1). Utilizaremos la siguiente notación:

- $\mathbf{V_{ij}}$, es el peso asociado a la conexión entre la neurona de entrada *j* y la neurona oculta/de salida *i*.
- $\mathbf{U_{jr}}$, es el peso asociado a la conexión recurrente entre la neurona oculta/de salida *r* y la neurona oculta/de salida destino *j*.

La Figura 2.9 muestra la estructura de un cromosoma que codifica una red recurrente completa con *n* entradas, *h* neuronas ocultas, y *o* neuronas de salida.

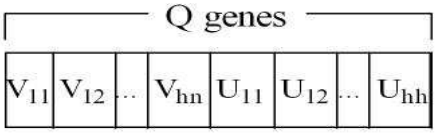

Figura 2.9. Codificación general de una red recurrente completa

Atendiendo a la Figura 2.9, un cromosoma que codifique una red neuronal recurrente completa con *n* neuronas de entrada y *h* neuronas ocultas/de salida, contendrá un total de *Q* genes, donde *Q* viene dado por:

$$Q = n \cdot h + h \cdot h \tag{2.2}$$

La Figura 2.10 muestra un ejemplo de la codificación de una red recurrente completa con una entrada, una salida, y dos neuronas ocultas. Al igual que ocurre en la representación de la red de Elman, cada gen del cromosoma codifica una conexión concreta de la red neuronal. El valor del gen se corresponde con el valor del peso de la conexión. Dado un cromosoma $C=(c_1, c_2, \ldots, c_Q)$, el Algoritmo 2.2 muestra el esquema de asignación de conexiones a genes, para redes neuronales recurrentes completas.

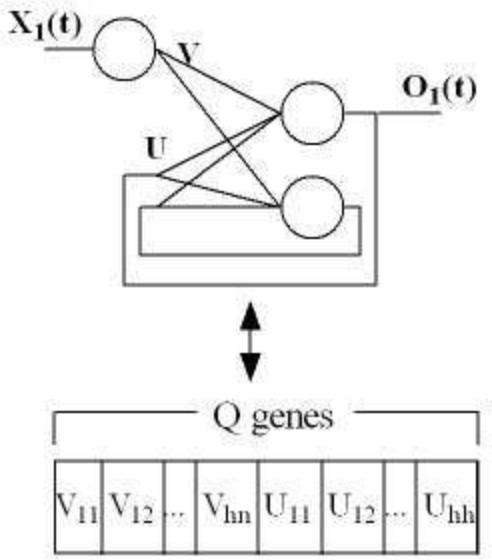

Figura 2.10. Ejemplo de codificación de una red recurrente completa con una entrada, una salida y dos neuronas ocultas

1. *m= 1*
2. *Para cada neurona de entrada i=1..n*
   - *2.1. para cada neurona oculta j=1..h*
     - *2.2. Asignar peso V_{ij} al gen c_m*
     - *2.3. Asignar m=m+1*
3. *Para cada neurona oculta i=1..h*
   - *3.1. para cada neurona oculta j=1..h*
     - *3.2. Asignar peso U_{ij} al gen c_m*
     - *3.3. Asignar m= m+1*

Algoritmo 2.2. Procedimiento de asignación de conexiones de una red recurrente completa a genes de un cromosoma

Representación del modelo de Jordan

La red recurrente de Jordan está formada por tres capas de neuronas: entrada, salida, y capa oculta. Utilizaremos la siguiente notación:

- $\mathbf{V_{ij}}$, es el peso asociado a la conexión entre la neurona de entrada *j* y la neurona oculta *i*.
- $\mathbf{U_{jr}}$, es el peso asociado a la conexión recurrente entre la neurona de salida *r* y la neurona oculta destino *j*.
- $\mathbf{W_{kj}}$, es el peso asociado a la conexión entre la neurona oculta *j* y la neurona de salida *k*.

La codificación de una red recurrente de Jordan en un cromosoma sigue el mismo esquema utilizado para los dos modelos de red anteriores. La figura 2.11 muestra la estructura de un cromosoma que codifica una red recurrente de Jordan con *n* entradas, *h* neuronas ocultas, y *o* neuronas de salida.

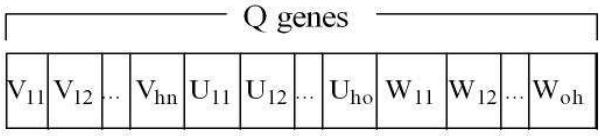

Figura 2.11. Codificación general de una red recurrente de Jordan

Atendiendo a la figura 2.11, un cromosoma que codifique una red neuronal recurrente de Jordan con *n* neuronas de entrada, *h* neuronas ocultas y *o* neuronas de salida, contendrá un total de *Q* genes, donde *Q* viene dado por:

$$Q = n \cdot h + 2 \cdot h \cdot o \tag{2.3}$$

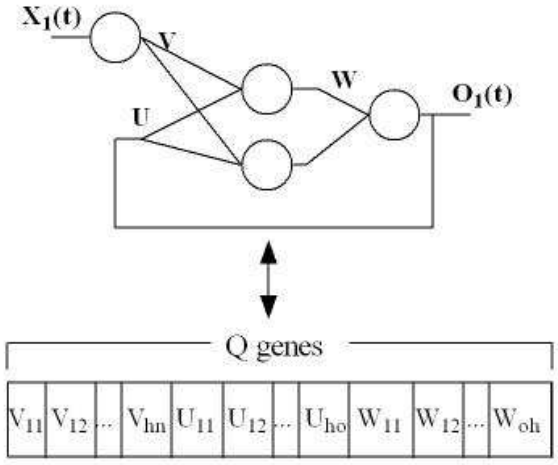

Figura 2.12. Ejemplo de codificación de una red recurrente de Jordan con una entrada, una salida y dos neuronas ocultas

La figura 2.12 muestra un ejemplo de la codificación de una red recurrente de Jordan con una entrada, una salida, y dos neuronas ocultas. El valor de un gen del cromosoma se corresponde con el valor del peso de la conexión. Dado un cromosoma $C=(c_1, c_2, \ldots, c_Q)$, el algoritmo 2.3 muestra el esquema de asignación de conexiones a genes para redes neuronales recurrentes de Jordan.

1. *m= 1*
2. *Para cada neurona de entrada i=1..n*
   - *2.1. para cada neurona oculta j=1..h*
     - *3.2. Asignar peso V_{ij} al gen c_m*
     - *3.3. Asignar m=m+1*
3. *Para cada neurona oculta i=1..h*
   - *3.1. para cada neurona de salida j=1..o*
     - *3.2. Asignar peso U_{ij} al gen c_m*
     - *3.3. Asignar m= m+1*
4. *Para cada neurona de salida i=1..o*
   - *4.1. para cada neurona oculta j=1..h*
     - *4.2. Asignar peso W_{ij} al gen c_m*
     - *4.3. Asignar m= m+1*

Algoritmo 2.3. Procedimiento de asignación de conexiones de una redde Jordan a genes de un cromosoma

3.2. La función objetivo

La función objetivo proporciona el criterio que el algoritmo evolutivo utilizará para dirigir la búsqueda hacia zonas prometedoras del espacio de soluciones.En nuestro caso, la tarea que se aborda consiste en entrenar una red recurrente aplicada aun problema de predicción de series temporales concreto. El objetivo a conseguir es minimizar una medida de error entre el comportamiento de la red y la evolución de la serie de datos.

El objetivo impuesto a los algoritmos de entrenamiento es minimizar el error cuadrático medio (MSE) entre la salida de la red, en el instante de tiempo *t*, y el valor de la serie temporal en el instante siguiente *t+1*. Utilizaremos el MSE debido a que este es un estimador estadístico ampliamente aceptado y utilizado en el entrenamiento de redes neuronales.

Continuando con la notación utilizada hasta el momento, si *C* es un cromosoma que codifica la red recurrente, *O(t)* es la salida de la red en el instante *t*, e *Y(t)* es el valor de la serie temporal en el instante *t*, el objetivo a conseguir por medio de los algoritmos evolutivos será:

$$f(C^*) = Min\, f(C) = Min\{\frac{1}{T}\sum_{i=1}^{T}(O(t) - Y(t+1))^2\} \tag{2.4}$$

La serie temporal estará dividida en dos conjuntos de datos:

- El conjunto de entrenamiento, que comprende los valores de la serie *Y(1), Y(2), Y(3), …, Y(T+1)*.
- El conjunto de test, que comprende el resto de los valores finales de la serie temporal.

Durante el proceso evolutivo, la función objetivo será aplicada sobre el conjunto de entrenamiento. Al finalizar, el comportamiento de la red será contrastado con la evolución de la serie temporal completa (entrenamiento y test), con el fin de comprobar la capacidad de generalización de la que ha sido provista la red en el aprendizaje

De este modo, en el capítulo de experimentación contaremos con dos valores estadísticos de error MSE: uno correspondiente al error producido en el conjunto de entrenamiento, y otro correspondiente al producido en el conjunto de test.

3.3. Algoritmos genéticos para entrenamiento de RNRD. Diversidad y convergencia mediante los operadores de evolución.

En este apartado realizamos un estudio de la eficacia de diferentes modelos evolutivos genéticos [BAC96][ESH93], para entrenar RNRD. Del mismo modo, hacemos uso de diferentes operadores genéticos para conseguir un equilibrio adecuado entre diversidad y convergencia en los algoritmos.

Recordemos que un algoritmo genético actúa sobre un conjunto de soluciones, denominado población, siguiendo un ciclo evolutivo similar al mostrado en la Figura 2.13.

Figura 2.13. Esquema evolutivo genético genera

A continuación, realizamos una breve revisión del funcionamiento de los esquemas evolutivos genéticos estudiados.

Modelo generacional

El modelo generacional (GGA) [BAC96][ESH93] se caracteriza por generar en cada iteración un número de descendientes igual al tamaño de la población,mediante el operador de cruce. En la siguiente generación, los descendientes reemplazanpor completo a la población de la iteración actual.

No obstante, los mejores individuos encontrados hasta la iteración actual pueden perderse en este intercambio. Para evitar esta situación, suele incorporarse un factor de elitismo que impide que las mejores soluciones encontradas puedan perderse en lasiguiente generación.

El Algoritmo 2.4 muestra el esquema general del modelo GGA.

1. t= 0, P(t)= inicializar población con N soluciones

2. Evaluar P(t)

3. Mientras (condición de parada = FALSO)

3.1. asignar t= t+1

3.2. P'(t)= Seleccionar N soluciones de P(t)

3.3. Cruce de soluciones en P'(t), con probabilidad p_c

para generar N nuevas soluciones, $\{H\}_N$

3.4. Mutación de soluciones $\{H\}_N$*, con probabilidad* p_m

3.5. Evaluar soluciones en $\{H\}_N$

3.6. Reemplazamiento de P(t) con $\{H\}_N$

3.7. Elitismo, si procede

4. Devolver resultados

Algoritmo 2.4. Procedimiento general de algoritmo genético con estrategia generacional

El algoritmo genético generacional ha sido utilizado para entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 2000 soluciones evaluadas
- **Probabilidad de cruce:** 0.8
- **Probabilidad de mutación:** 0.01
- **Factor de elitismo:** Los 2 mejores individuos permanecen en la población
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** número real en el intervalo [-5, 5]
- **Operador de selección:** Torneo binario (TO)/Sorteo (SO)/Equitativo (EQ)/Lineal (LI)
- **Operador de cruce:** BLX-α (BLX)/Heurístico (HEU)/Aritmético (ARI)

- **Operador de mutación:** Desplazamiento (DES)/Aleatorio (ALE)
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** 8 (Elman), 4 (FRNN) y 11 (Jordan)

Se han realizado 30 ejecuciones del algoritmo genético generacional con los parámetros anteriores, para cada tipo de red. Analizaremos los resultados estadísticamente.

Las tablas A.1 a A.3 (Anexo I) muestran los resultados del test de normalidad de Kolmogorov-Smirnoff, realizado sobre el error cuadrático medio obtenido en el conjunto de resultados de test, en cada combinación de operadores para el algoritmo genético generacional. Se puede comprobar que la mayoría no ha superado el test de normalidad.

Para comprobar la eficacia de cada modelo, hemos realizado tests de equivalencia estadística entre los resultados obtenidos por las diferentes combinaciones de operadores, en el algoritmo genético generacional. Debido a que los tests de normalidad han concluido que la mayoría de los conjuntos de datos no siguen una distribución normal, utilizaremos tests no paramétricos para la comprobación (test de Kruskal-Wallis). El estimador estadístico utilizado será, por tanto, la mediana.

Los resultados de los tests de equivalencia estadística, utilizando el algoritmo genético generacional, se muestran en las tablas A.25, A.26 y A.27 para cada modelo de red aplicado al problema CATS, respectivamente. Adicionalmente, la tabla 2.1 muestra un resumen de las mejores combinaciones encontradas, estadísticamente equivalentes, para cada modelo de red y problema.

Tabla 2.1: Mejores combinaciones de operadores para el algoritmo genético generacional, en cada tipo de red y subproblema CATS

Problema	Elman	FRNN	Jordan
CATS1	*GGA/TO/HEU/DES*	*GGA/TO/BLX/DES* *GGA/TO/ARI/ALE* *GGA/LI/BLX/DES*	*GGA/TO/BLX/DES* *GGA/TO/HEU/DES* *GGA/LI/BLX/DES* *GGA/LI/HEU/DES*

CATS2	*GGA/TO/HEU/ALE* *GGA/TO/HEU/DES*	*GGA/TO/ARI/ALE* *GGA/TO/ARI/DES*	*GGA/TO/BLX/DES* *GGA/TO/BLX/ALE*
CATS3	*GGA/TO/HEU/DES* *GGA/TO/HEU/ALE* *GGA/LI/HEU/ALE* *GGA/TO/BLX/DES* *GGA/LI/HEU/DES* *GGA/TO/BLX/ALE*	*GGA/TO/ARI/ALE* *GGA/TO/BLX/DES* *GGA/TO/ARI/DES* *GGA/TO/BLX/ALE* *GGA/LI/ARI/ALE* *GGA/LI/BLX/DES* *GGA/LI/BLX/ALE* *GGA/LI/ARI/DES*	*GGA/TO/BLX/DES* *GGA/TO/HEU/DES*
CATS4	*GGA/TO/HEU/ALE* *GGA/TO/HEU/DES* *GGA/LI/HEU/DES* *GGA/LI/HEU/ALE* *GGA/TO/BLX/ALE*	*GGA/TO/ARI/ALE* *GGA/LI/ARI/DES* *GGA/LI/ARI/ALE* *GGA/TO/ARI/DES* *GGA/TO/BLX/ALE* *GGA/TO/BLX/DES* *GGA/LI/BLX/DES*	*GGA/TO/HEU/ALE* *GGA/TO/HEU/DES* *GGA/LI/HEU/ALE* *GGA/LI/HEU/DES*
CATS5	*GGA/TO/BLX/DES* *GGA/LI/HEU/ALE* *GGA/LI/HEU/DES* *GGA/TO/HEU/DES* *GGA/TO/HEU/ALE*	*GGA/TO/BLX/ALE*	*GGA/LI/BLX/DES* *GGA/TO/BLX/DES* *GGA/TO/HEU/DES* *GGA/TO/BLX/ALE*

Según la información proporcionada en la tabla 2.1, podemos observar que el operador de selección que con mayor frecuencia se repite es la *selección por torneo binario*, aunque la *selección lineal* también está contenida en gran parte de las soluciones mostradas. Por tanto, la combinación de operadores que presumiblemente deben proporcionar la mejor solución, independientemente del tipo de red entrenada, debe contener alguno de estos operadores de selección (principalmente, el operador de selección por *torneo binario*).

Al contrario, el operador de cruce depende en mayor parte del modelo de red a entrenar. Mientras que el *cruce Heurístico de Wright* ha proporcionado las mejores soluciones para los modelos de red de Elman, los operadores de *cruce aritmético* y *BLXα* lo han hecho para las redes Completas. Finalmente, las redes de Jordan han proporcionado los mejores resultados con los operadores *heurístico de Wright* y *BLXα*.

El operador de mutación tiene un aspecto irrelevante en la selección de operadores para el modelo de algoritmo genético generacional, en la mayoría de los problemas. No obstante, la mutación por desplazamiento ayuda a conseguir la mejor solución en un número de problemas mayor que la mutación aleatoria.

Las figuras 2.14 a 2.28 muestran la evolución de la diversidad de la mejor ejecución del algoritmo genético generacional en cada tipo de red, para las combinaciones de operadores que han proporcionado resultados no estadísticamente equivalentes en cada problema. Podemos observar que las combinaciones que mejor convergencia han tenido poseen una alta diversidad en las primeras iteraciones del algoritmo, mientras que se ve fuertemente disminuida en las últimas. Al contrario, las soluciones que peores resultados proporcionan suelen tener un fuerte descenso de la diversidad en las primeras iteraciones del algoritmo.

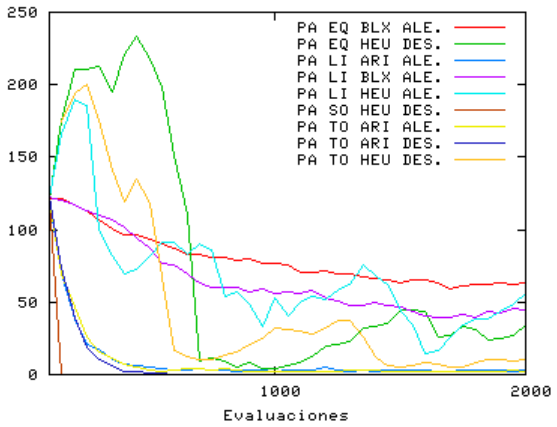

Figura 2.14. Red de Elman. Problema CATS1

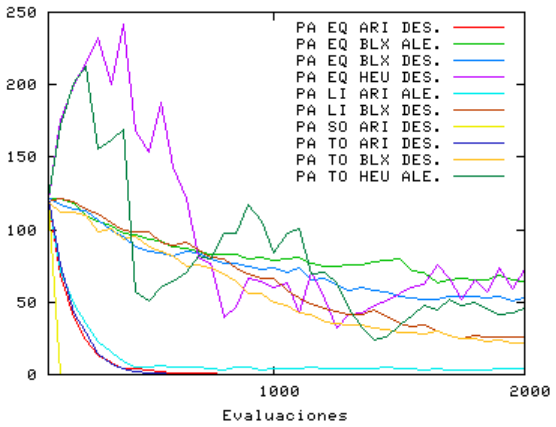

Figura 2.15. Red de Elman. Problema CATS2

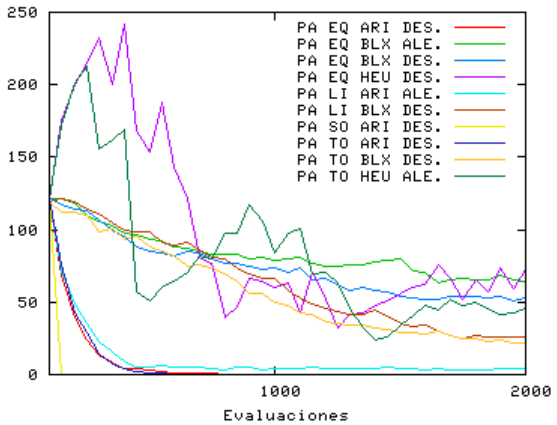

Figura 2.16. Red de Elman. Problema CATS3

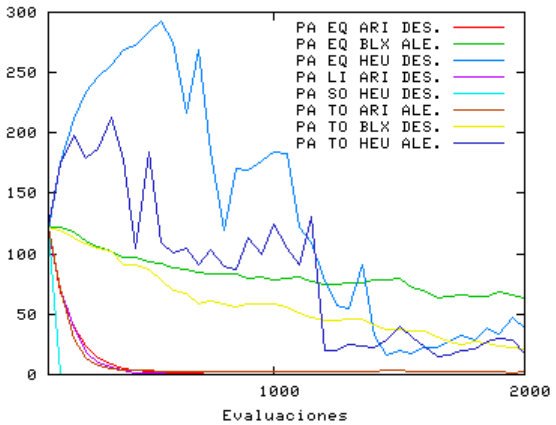

Figura 2.17. Red de Elman. Problema CATS4

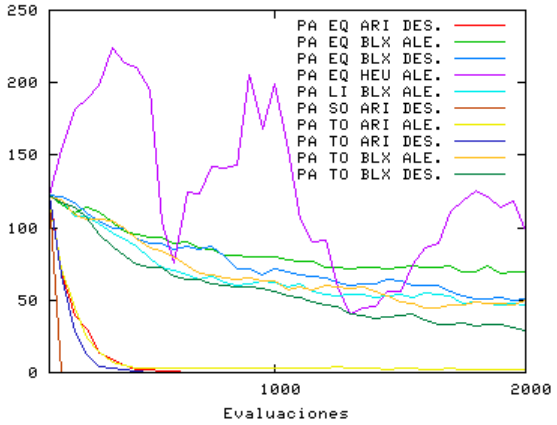

Figura 2.18. Red de Elman. Problema CATS5

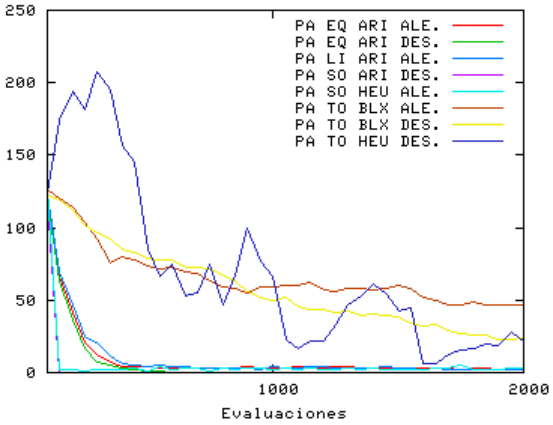

Figura 2.19. Red Completa. Problema CATS1

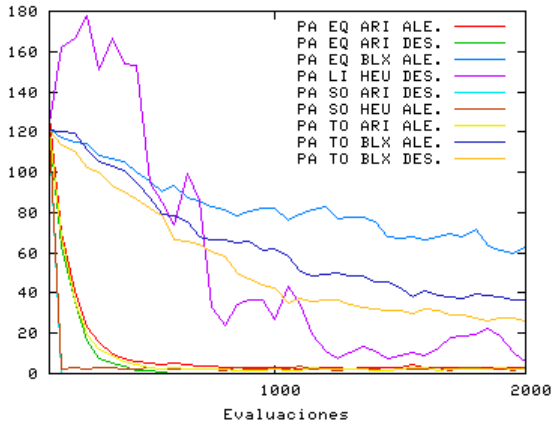

Figura 2.20. Red Completa. Problema CATS2

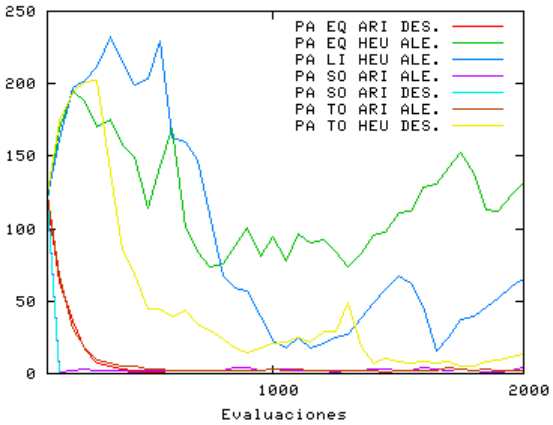

Figura 2.21. Red Completa. Problema CATS3

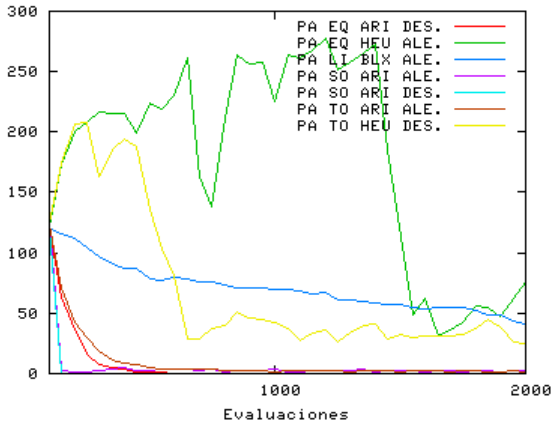

Figura 2.22. Red Completa. Problema CATS4

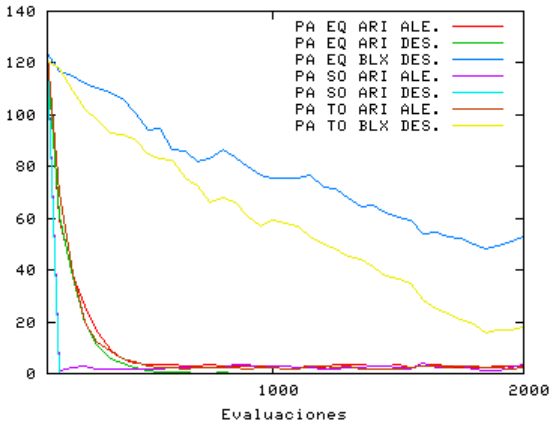

Figura 2.23. Red Completa. Problema CATS5

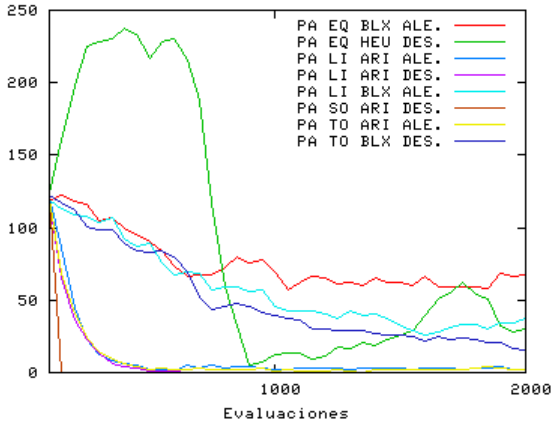

Figura 2.24. Red de Jordan Problema CATS1

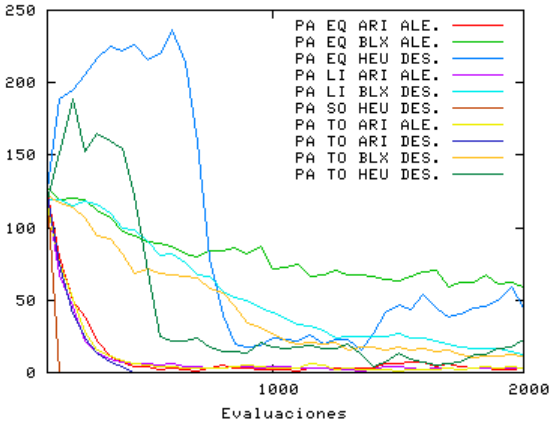

Figura 2.25. Red de Jordan Problema CATS2

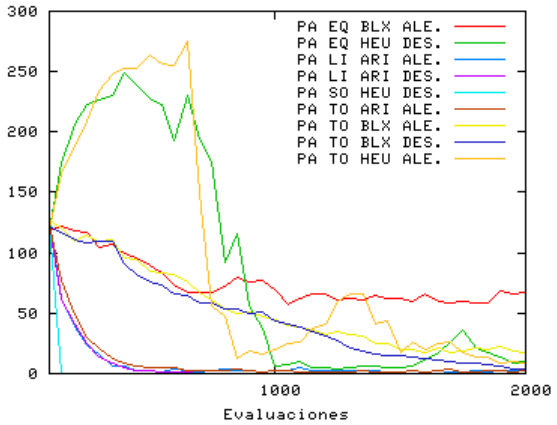

Figura 2.26. Red de Jordan Problema CATS3

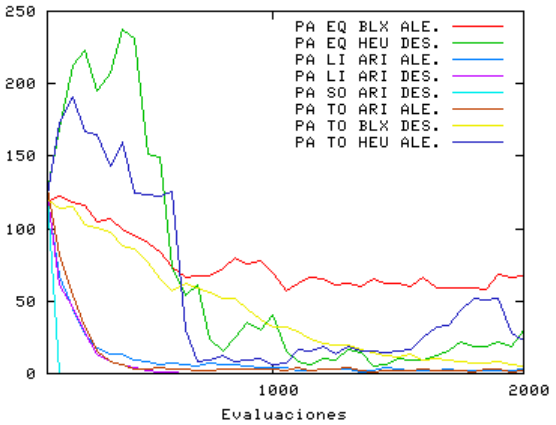

Figura 2.27. Red de Jordan Problema CATS4

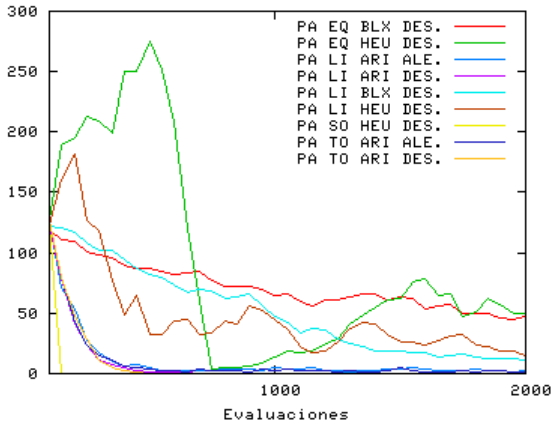

Figura 2.28. Red de Jordan Problema CATS5

Modelo estacionario

La novedad principal introducida en el modelo estacionario influye especialmente en los mecanismos de selección y cruce [BAC96][ESH93]. En el modelo estacionario, se seleccionan un total de K padres a partir de la población $P(t)$. De entre ellos se seleccionan dos padres aleatoriamente para generar dos descendientes con probabilidad $p_c = 1.0$, mediante el operador de cruce. Este proceso es repetido hasta obtener un total de K nuevas soluciones.

Otra novedad del modelo estacionario, con respecto al generacional, tiene relación con el reemplazamiento de soluciones de una generación a la siguiente. En el esquema estacionario, dependiendo de la estrategia de reemplazamiento, los hijos podrán reemplazar a los padres, a los peores individuos, a individuos seleccionados al azar, etc. El algoritmo 2.5 muestra el esquema general de un algoritmo genético estacionario.

El algoritmo genético estacionario ha sido utilizado para entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 2000 soluciones evaluadas
- **Probabilidad de cruce:** 1.0
- **Probabilidad de mutación:** 0.01
- **Factor de elitismo:** Los 2 mejores individuos permanecen en la población
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** número real en el intervalo [-5, 5]
- **Operador de selección:** Torneo binario (TO)/Sorteo (SO)/Equitativo (EQ)/Lineal (LI)
- **Operador de cruce:** BLX-α (BLX)/Heurístico (HEU)/Aritmético (ARI)
- **Operador de mutación:** Desplazamiento (DES)/Aleatorio (ALE)
- **Reemplazamiento:** Padres (PA)/Peores (PE)
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)

- **Neuronas ocultas:** 8 (Elman), 4 (FRNN) y 11 (Jordan)

1. t= 0, P(t)= inicializar población con N soluciones

2. Evaluar P(t)

3. Mientras (condición de parada = FALSO)

3.1. asignar t= t+1

3.2. P'(t)= Seleccionar K soluciones de P(t)

3.3. Cruce de soluciones en P'(t), con probabilidad p_c *= 1.0, para generar K nuevas soluciones,* $\{H\}_K$

3.4.Mutación de soluciones en $\{H\}_K$*, con probabilidad* p_m

3.5. Evaluar soluciones en $\{H\}_K$

3.6. Reemplazamiento de P(t) con $\{H\}_K$

3.7. Elitismo, si procede

4. Devolver resultados

Algoritmo 2.5. Procedimiento general de un algoritmo genético con estrategia estacionaria

Se han realizado 30 ejecuciones del algoritmo genético estacionario con los parámetros anteriores, para cada tipo de red. Las tablas A.4 a A.6 (AnexoI) muestran los resultados del test de normalidad de Kolmogorov-Smirnoff, realizado sobre el error cuadrático medio obtenido en el conjunto de resultados de test, en cada combinación de operadores para el algoritmo genético estacionario.

Los resultados de los tests de equivalencia estadística, utilizando el algoritmo genético estacionario, se muestran en las tablas A.28, A.29 y A.30 (Anexo I) para cada modelo de red aplicado al problema CATS, respectivamente. Adicionalmente, la tabla 2.2 muestra un resumen de las mejores combinaciones encontradas, estadísticamente equivalentes, para cada modelo de red y problema.

Tabla 2.2: Mejores combinaciones de operadores para el algoritmo genétco estacionario, en cada tipo de red y subproblema CATS

Problema	**Elman**	**FRNN**	**Jordan**
CATS1	SGA/PA/TO/HEU/DES SGA/PA/TO/HEU/ALE	SGA/PE/TO/ARI/ALE SGA/PE/TO/ARI/DES SGA/PE/EQ/ARI/ALE SGA/PE/LI/ARI/ALE SGA/PE/LI/ARI/DES SGA/PA/LI/ARI/ALE	SGA/PE/TO/HEU/DES SGA/PA/LI/HEU/DES SGA/PE/LI/HEU/DES SGA/PA/TO/HEU/DES SGA/PA/LI/HEU/ALE
CATS2	SGA/PA/LI/HEU/ALE SGA/PA/TO/HEU/DES SGA/PA/TO/HEU/ALE SGA/PA/EQ/HEU/ALE	SGA/PE/TO/ARI/DES SGA/PE/TO/ARI/ALE SGA/PE/LI/ARI/ALE SGA/PE/LI/ARI/DES SGA/PA/TO/ARI/ALE SGA/PE/EQ/ARI/ALE SGA/PE/EQ/ARI/DES	SGA/PE/LI/HEU/ALE SGA/PE/TO/HEU/DES SGA/PE/LI/HEU/DES SGA/PE/TO/HEU/ALE SGA/PA/LI/HEU/ALE SGA/PA/EQ/HEU/DES
CATS3	SGA/PA/TO/HEU/DES SGA/PA/EQ/HEU/DES SGA/PA/LI/HEU/DES SGA/PA/TO/HEU/ALE SGA/PA/LI/HEU/ALE SGA/PA/EQ/HEU/ALE	SGA/PE/TO/ARI/DES SGA/PA/TO/ARI/ALE SGA/PE/LI/ARI/ALE SGA/PE/LI/ARI/DES SGA/PE/TO/ARI/ALE SGA/PE/EQ/ARI/DES	SGA/PA/TO/HEU/DES SGA/PA/EQ/HEU/DES SGA/PE/LI/HEU/DES SGA/PA/LI/HEU/ALE SGA/PE/TO/HEU/DES SGA/PE/LI/HEU/ALE
CATS4	SGA/PA/TO/HEU/ALE SGA/PA/LI/HEU/DES SGA/PA/EQ/HEU/ALE SGA/PA/EQ/HEU/DES SGA/PA/TO/HEU/DES SGA/PA/LI/HEU/ALE SGA/PE/TO/BLX/ALE	SGA/PE/TO/ARI/DES SGA/PE/LI/ARI/ALE SGA/PE/TO/ARI/ALE SGA/PE/LI/ARI/DES SGA/PA/LI/ARI/ALE	SGA/PA/EQ/HEU/DES SGA/PA/LI/HEU/DES SGA/PA/TO/HEU/ALE SGA/PA/TO/HEU/DES SGA/PA/LI/HEU/ALE
CATS5	SGA/PA/EQ/HEU/ALE SGA/PA/TO/HEU/DES SGA/PA/LI/HEU/DES SGA/PA/TO/HEU/ALE SGA/PA/EQ/HEU/DES SGA/PA/LI/HEU/ALE	SGA/PE/TO/ARI/ALE SGA/TO/BLX/DES SGA/PE/TO/ARI/DES SGA/PE/LI/ARI/ALE SGA/PE/EQ/ARI/ALE SGA/PE/LI/ARI/DES	SGA/PE/TO/HEU/ALE SGA/PE/TO/HEU/DES SGA/PE/LI/HEU/ALE SGA/PE/LI/HEU/DES SGA/PE/EQ/HEU/DES

Las conclusiones obtenidas para el modelo genético estacionariosobre los operadores de selección y mutación son similares a las obtenidas por el algoritmo genético generacional. Sin embargo, la tabla 2.2 muestra que el operador de cruce *Heurístico de Wright* es la mejor opción para entrenar redes de Elman y Jordan, en los problemas planteados. Con respecto a redes recurrentes completas, el cruce que mejores resultados ha proporcionado ha sido el cruce *aritmético.*

Con respecto a la estrategia de reemplazamiento utilizada, se puede comprobar que el reemplazamiento a los padres ha producido los mejores resultados en los modelos de redes de Elman, y parte de Jordan, mientras que el reemplazamiento por los peores individuos ha tenido éxito al entrenar redes recurrentes completas.

Las figuras 2.29 a 2.43 muestran la evolución de la diversidad de la mejor ejecución del algoritmo genético estacionario en cada tipo de red, para las combinaciones de operadores que han proporcionado los mejores resultados y el resto de soluciones no estadísticamente equivalentes, en cada problema.

El dato más significativo de estas gráficas se observa en las ejecuciones correspondientes a la estrategia de reemplazamiento a los peores individuos, las cuales muestran que la diversidad no varía a lo largo de la ejecución, sino que oscila alrededor de un valor concreto, cercano al valor de la distancia Euclídea media de la población inicial.

No obstante, las mejores soluciones las han obtenido aquellas combinaciones de operadores que permiten una gran diversidad al comienzo de la ejecución, y desciende a lo largo de la misma.

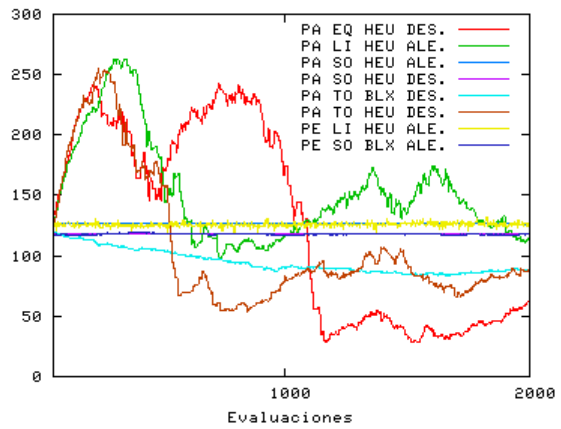

Figura 2.29. Red de Elman Problema CATS1

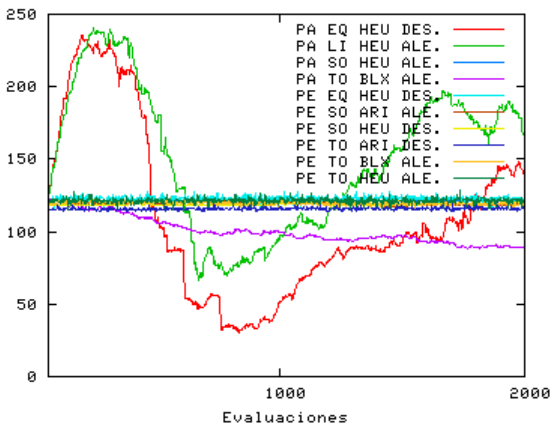

Figura 2.30. Red de Elman Problema CATS2

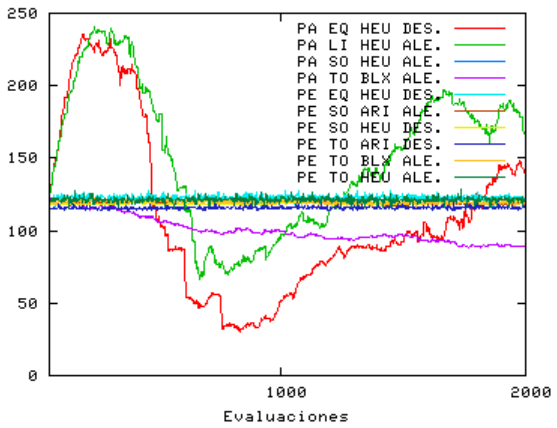

Figura 2.31. Red de Elman Problema CATS3

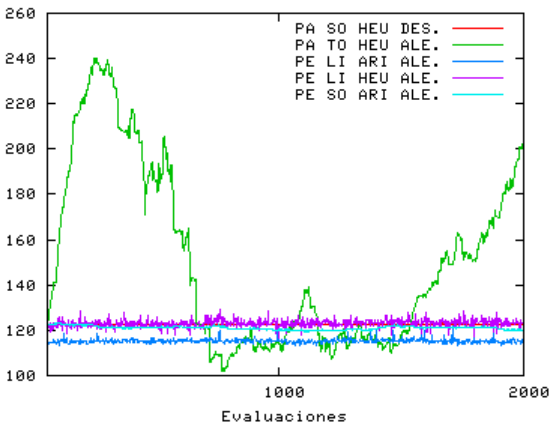

Figura 2.32. Red de Elman Problema CATS4

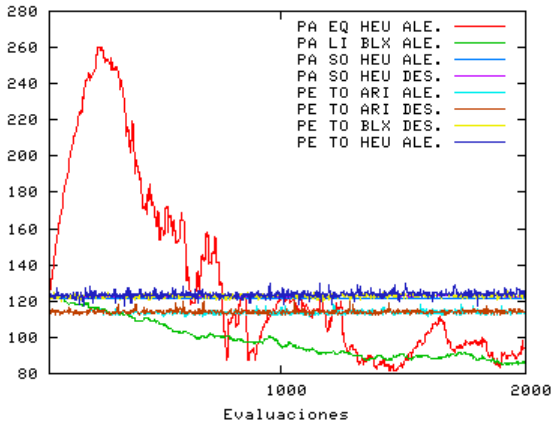

Figura 2.33. Red de Elman Problema CATS5

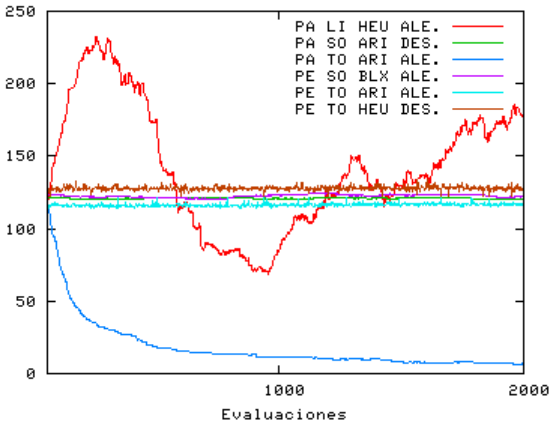

Figura 2.34. Red Completa Problema CATS1

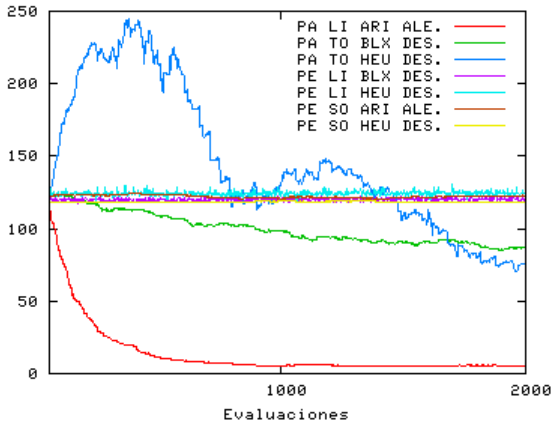

Figura 2.35. Red Completa Problema CATS2

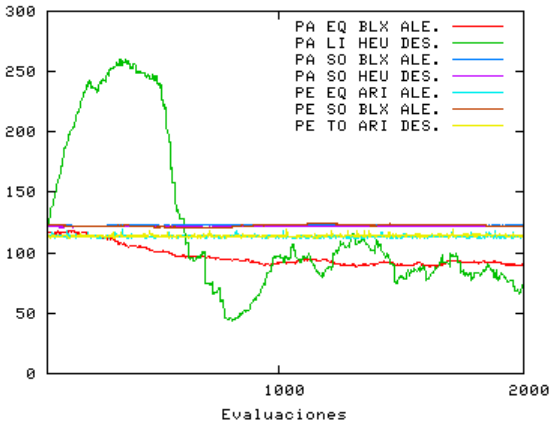

Figura 2.36. Red Completa Problema CATS3

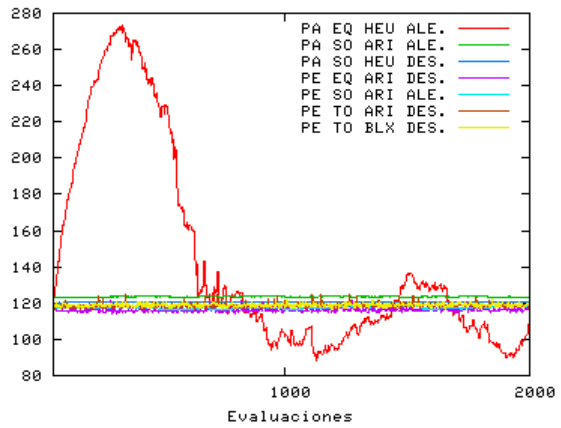

Figura 2.37. Red Completa Problema CATS4

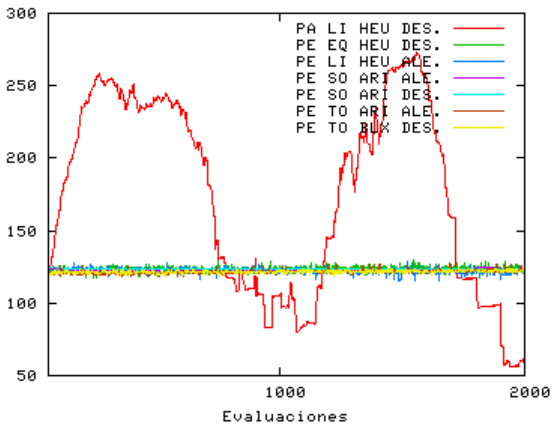

Figura 2.38. Red Completa Problema CATS5

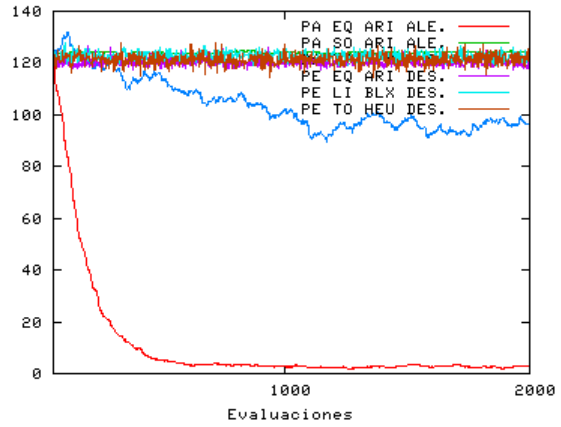

Figura 2.39. Red de Jordan Problema CATS1

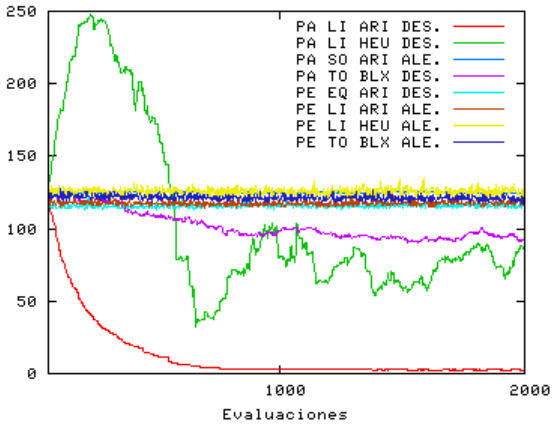

Figura 2.40. Red de Jordan Problema CATS2

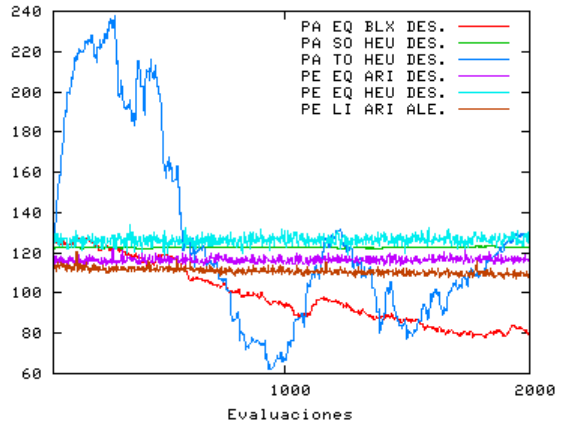

Figura 2.41. Red de Jordan Problema CATS3

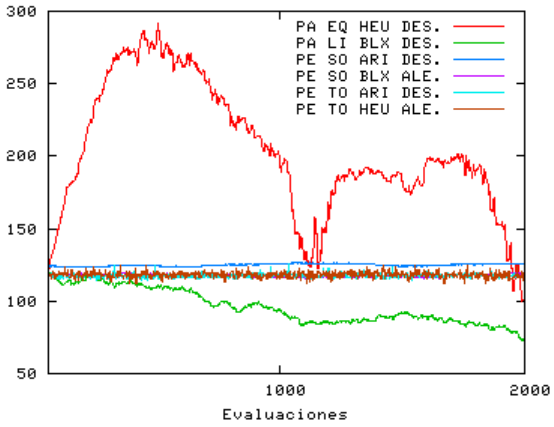

Figura 2.42. Red de Jordan Problema CATS4

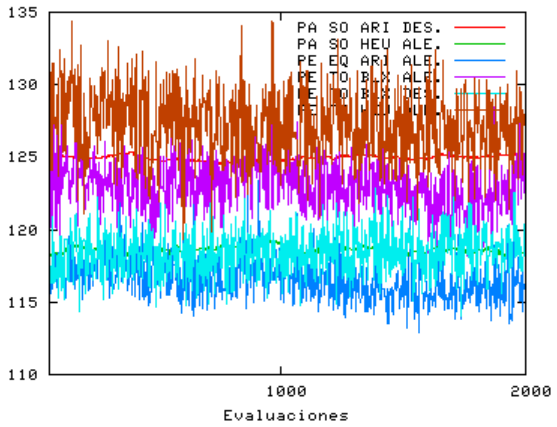

Figura 2.43. Red de Jordan Problema CATS5

Modelo mixto

La propuesta de algoritmo genético que se detalla en este apartado es un híbrido entre el modelo generacional y el estacionario. Mientras que los operadoresde selección y cruce siguen un esquema estacionario, la mutación se aplica a todos los elementos de la población. De este modo, tanto la mutación como el reemplazamiento pueden ser considerados dentro de una estrategia generacional.

Mediante este mecanismo, conseguimos aumentar la diversidad en la población y solventar parcialmente el problema del algoritmo genético estacionario con reemplazamiento a los peores individuos de la población, producido en el apartado anterior. El proceso completo del modelo genético mixto se detalla en el Algoritmo 2.6. El algoritmo genético mixto ha sido utilizado para entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 2000 soluciones evaluadas
- **Probabilidad de cruce:** 1.0
- **Probabilidad de mutación:** 0.01
- **Factor de elitismo:** Los 2 mejores individuos permanecen en la población
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** número real en el intervalo [-5, 5]

- **Operador de selección:** Torneo binario (TO)/Sorteo (SO)/Equitativo (EQ)/Lineal (LI)
- **Operador de cruce:** BLX-α (BLX)/Heurístico (HEU)/Aritmético (ARI)
- **Operador de mutación:** Desplazamiento (DES)/Aleatorio (ALE)
- **Reemplazamiento:** Padres (PA)/Peores (PE)
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** 8 (Elman), 4 (FRNN) y 11 (Jordan)

1. t= 0, P(t)= inicializar población con N soluciones

2. Evaluar P(t)

3. Mientras (condición de parada = FALSO)

3.1. asignar t= t+1

3.2. P'(t)= Seleccionar K soluciones de P(t)

3.3. Cruce de soluciones en P'(t), con probabilidad

p_c *= 1.0, para generar K nuevas soluciones,* $\{H\}_K$

3.4. Asignar $\{H\}_{N+K}=\{H\}_K+P(t)$

3.5. Mutación de soluciones en $\{H\}_{N+K}$*, con probabilidad* p_m

3.6. Evaluar soluciones en $\{H\}_{N+K}$

10. Reemplazamiento de P(t) con $\{H\}_{N+K}$

11. Elitismo, si procede

4. Devolver resultados

Algoritmo 2.6. Procedimiento general de un algoritmo genético con estrategia mixta

Se han realizado 30 ejecuciones del algoritmo genético mixto con los parámetros anteriores, para cada tipo de red. Las tablas A.7 a A.9 (AnexoI) muestran los resultados del test de normalidad realizado sobre el conjunto de resultados de test, en cada combinación de operadores para el algoritmo genético mixto. Del mismo modo, los resultados de los tests de

equivalencia estadística se muestran en las tablas A.31, A.32 y A.33 para los modelos de redes recurrentes aplicado al problema CATS. Adicionalmente, la tabla 2.3 muestra un resumen de las mejores combinaciones encontradas, estadísticamente equivalentes, para cada modelo de red y problema.

Tabla 2.3: Mejores combinaciones de operadores para el algoritmo genético mixto, en cada tipo de red y subproblema CATS

Problema	**Elman**	**FRNN**	**Jordan**
CATS1	MGA/PE/TO/BLX/ALE MGA/PE/LI/HEU/ALE MGA/PE/LI/BLX/ALE MGA/PE/TO/HEU/ALE	MGA/PE/LI/BLX/ALE MGA/PE/LI/ARI/ALE MGA/PE/TO/BLX/ALE MGA/PE/EQ/BLX/ALE	MGA/PE/LI/HEU/ALE MGA/PE/LI/BLX/ALE MGA/PE/TO/HEU/ALE
CATS2	MGA/PE/LI/BLX/ALE MGA/PE/TO/BLX/ALE	MGA/PE/TO/BLX/ALE MGA/PE/LI/ARI/ALE MGA/PE/LI/BLX/ALE	MGA/PE/LI/BLX/ALE MGA/PE/EQ/BLX/ALE MGA/PE/TO/BLX/ALE
CATS3	MGA/PE/TO/HEU/ALE MGA/PE/LI/HEU/ALE MGA/PE/LI/BLX/ALE MGA/PE/TO/BLX/ALE	MGA/PE/EQ/ARI/ALE MGA/PE/TO/HEU/ALE MGA/PE/TO/BLX/ALE	MGA/PE/LI/HEU/ALE MGA/PE/LI/BLX/ALE MGA/PE/TO/BLX/ALE MGA/PE/TO/HEU/ALE MGA/PE/EQ/BLX/ALE
CATS4	MGA/PE/LI/HEU/ALE MGA/PE/TO/HEU/ALE MGA/PE/EQ/HEU/ALE	MGA/PE/LI/BLX/ALE	MGA/PE/LI/HEU/ALE MGA/PE/TO/HEU/ALE MGA/PE/EQ/HEU/ALE
CATS5	MGA/PE/LI/BLX/ALE MGA/PE/TO/BLX/ALE MGA/PE/LI/HEU/ALE	MGA/PE/EQ/ARI/ALE MGA/PE/TO/BLX/ALE MGA/PE/LI/BLX/ALE	MGA/PE/LI/HEU/ALE MGA/PE/LI/BLX/ALE MGA/PE/TO/BLX/ALE MGA/PE/EQ/BLX/ALE MGA/PE/TO/HEU/ALE

En contraposición a los modelos genéticos anteriores, la tabla 2.3 muestra que el operador de mutación juega un papel importante en el modelo mixto, obteniéndose los mejores resultados con la mutación *aleatoria*.

Del mismo modo, las limitaciones encontradas en el modelo genético estacionario (con respecto a la estrategia de reemplazamiento a los peores individuos) también han sido superadas. De hecho, la estrategia de reemplazamiento a los peores individuos forma parte de las mejores

soluciones encontradas en el algoritmo genético mixto, independientemente del modelo de red entrenada.

En cuanto a los operadores de selección, la tabla 2.2 muestra que aquellos que mejor resultados proporcionan son el *lineal* y el *torneo binario* (al igual que en los modelos genéticos anteriores).

Finalmente, el operador de cruce *BLXα* muestra un buen comportamiento en la mayoría de los problemas, independientemente del tipo de red entrenada. Este comportamiento suele ir ligado a la combinación junto con el operador de selección lineal (ver tabla 2.2). No obstante, si discriminamos por modelo de red, puede observarse que los mejores resultados para las redes de Elman y Jordan se alcanzan con los cruces *Heurístico* y *BLXα*, mientras que para las redes recurrentes completas se obtienen con los cruces *aritmético* y *BLXα*.

Las figuras 2.44 a 2.58 muestran la evolución de la diversidad de la mejor ejecución del algoritmo genético estacionario en cada tipo de red, para las combinaciones de operadores que han proporcionado los mejores resultados y el resto de soluciones no estadísticamente equivalentes, en cada problema.

El dato más significativo de estas gráficas, con respecto a las del modelo genético estacionario, se observa en las ejecuciones correspondientes a la estrategia de reemplazamiento a los peores individuos. En el algoritmo genético estacionario, el uso de esta estrategia de reemplazamiento impedía la variación en la diversidad de soluciones. Sin embargo, en el modelo mixto suele disminuirla levemente. Esto favorece la exploración del espacio de soluciones, llegando a proporcionar mejores soluciones en el caso del algoritmo genético mixto.

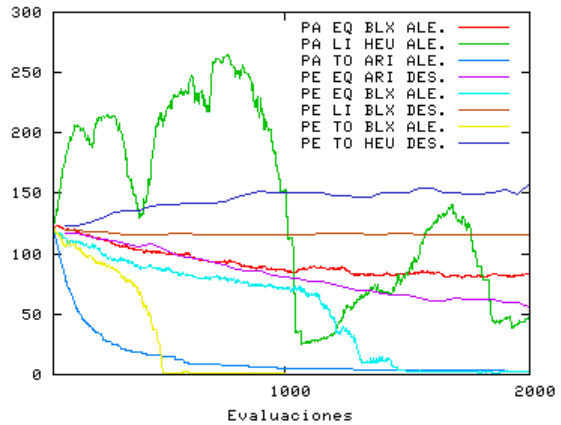

Figura 2.44. Red de Elman Problema CATS1

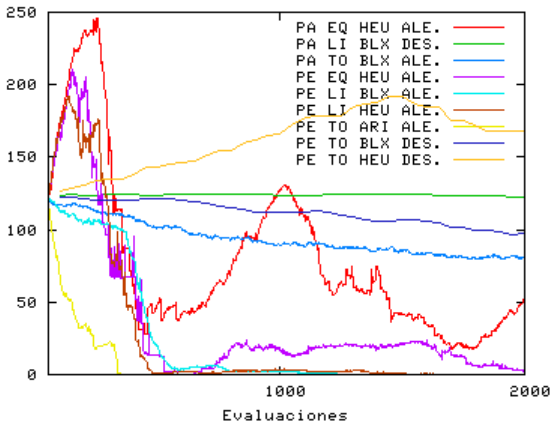

Figura 2.45. Red de Elman Problema CATS2

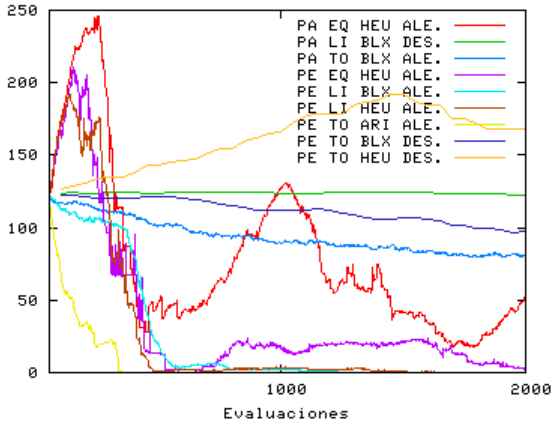

Figura 2.46. Red de Elman Problema CATS3

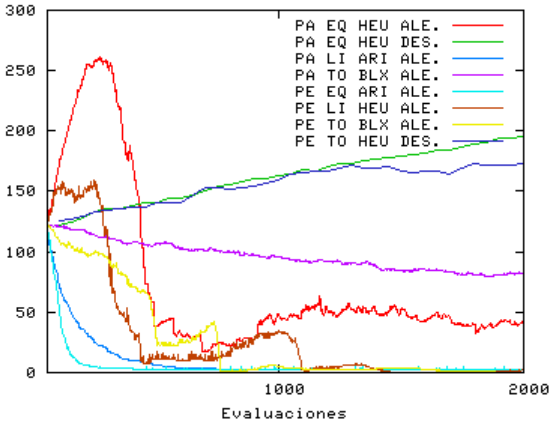

Figura 2.47. Red de Elman Problema CATS4

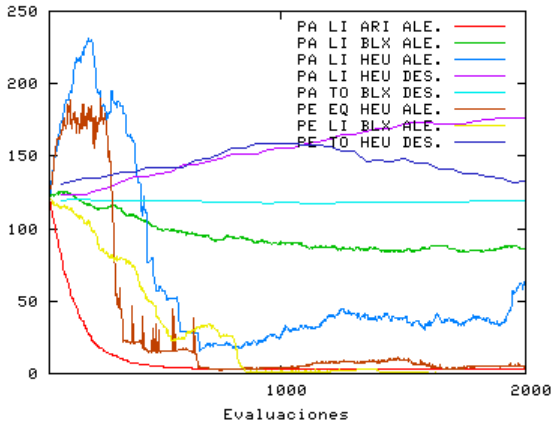

Figura 2.48. Red de Elman Problema CATS5

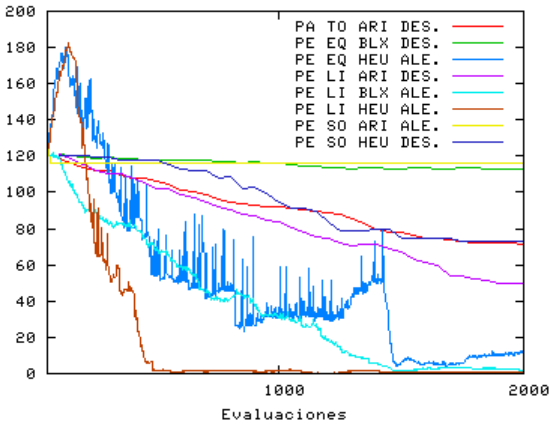

Figura 2.49. Red Completa Problema CATS1

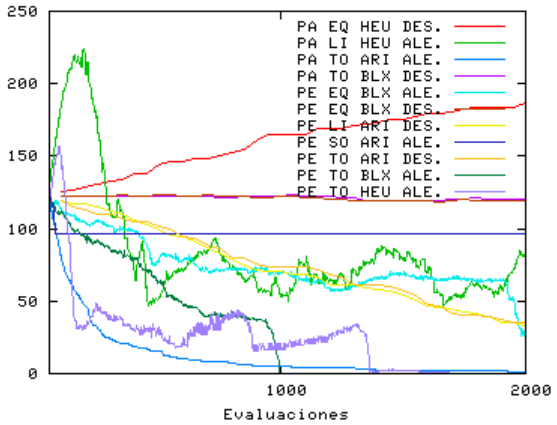

Figura 2.50. Red Completa Problema CATS2

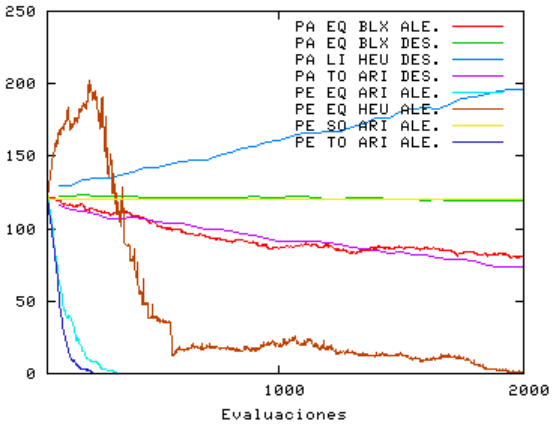

Figura 2.51. Red Completa Problema CATS3

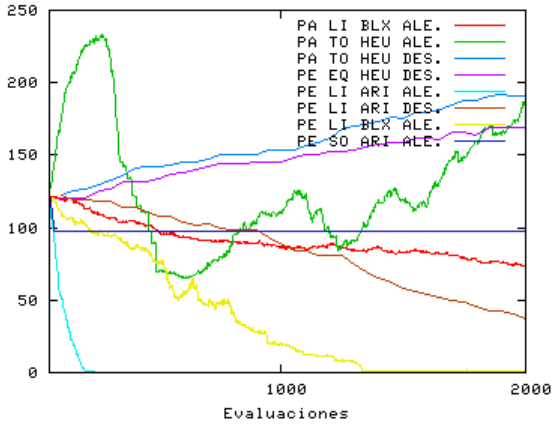

Figura 2.52. Red Completa Problema CATS4

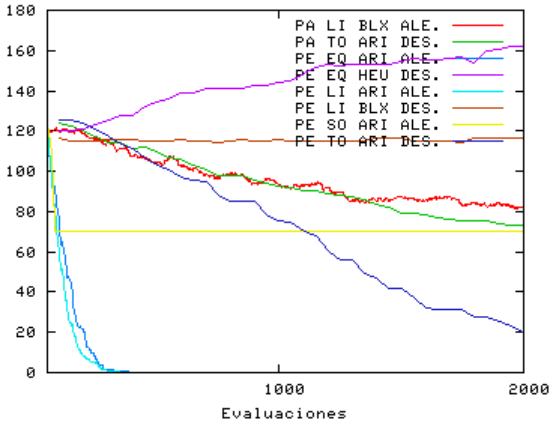

Figura 2.53. Red Completa Problema CATS5

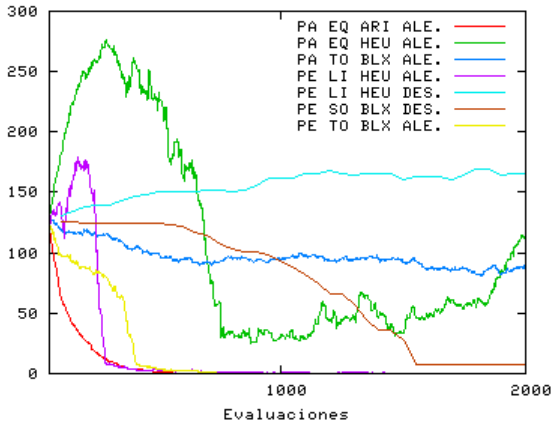

Figura 2.54. Red de Jordan Problema CATS1

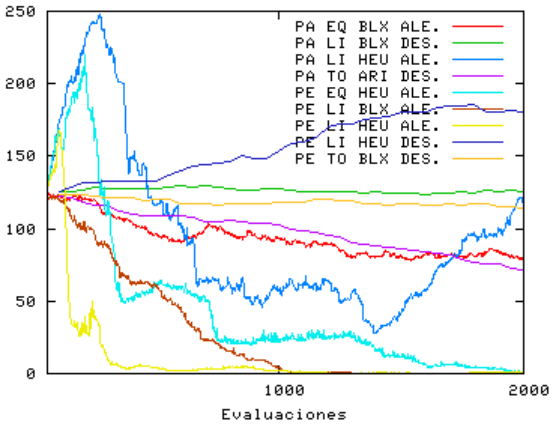

Figura 2.55. Red de Jordan Problema CATS2

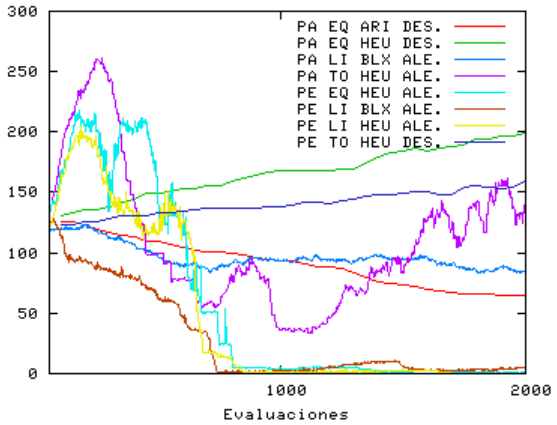

Figura 2.56. Red de Jordan Problema CATS3

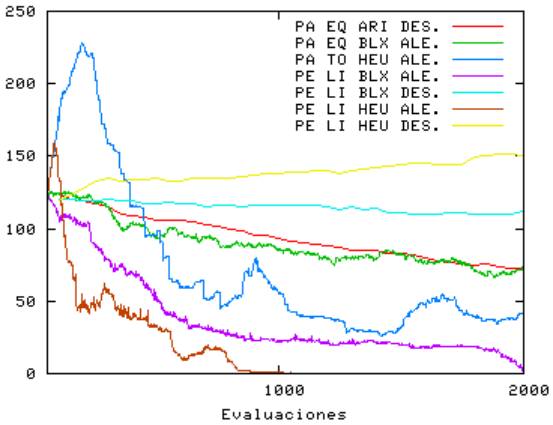

Figura 2.57. Red de Jordan Problema CATS4

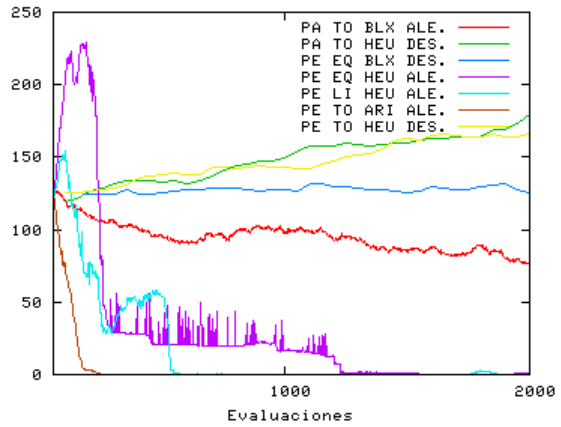

Figura 2.58. Red de Jordan Problema CATS5

3.4. Métodos de control de diversidad mediante la separación espacial de los cromosomas para entrenamiento de RNRD

Generalmente, los problemas que se abordan mediante el uso de algoritmos genéticos tienen un espacio de soluciones demasiado extenso, y muchas de los resultados de estos algoritmos pueden ser óptimos locales.

Una de las mejoras introducidas en los algoritmos genéticos, paraevitar obtener óptimos locales, ha dado lugar a los llamados algoritmos genéticos multimodales. Estos algoritmos trabajan concurrentemente en zonas diferentes del espacio de soluciones. El resultado es una mejor exploración del espacio de búsqueda, dando la posibilidad de encontrar mejores resultados.

Los algoritmos multimodales discriminan las soluciones dentro del dominio de su espacio, agrupándolas mediante técnicas similares a las de clustering (nichos) [MAH95]. Un nicho está formado por aquellos cromosomas cuya distancia sea menorque un umbral, llamado radio del nicho.

Los algoritmos genéticos multimodales trabajan sobre poblaciones que contienen soluciones optimales en varios nichos. Considerando la forma de hacer evolucionar los nichos, los algoritmos multimodales pueden clasificarse en dos categorías:

- **Espaciales**. Durante la misma ejecución, el algoritmo forma varios nichos en la población evolucionada (por ejemplo, Fitness Sharing o Clearing).
- **Temporales**. Se forman distintos nichos, uno en cada ejecución del algoritmo Multimodal (por ejemplo, Método secuencial).

En este trabajo nos centraremos en el algoritmo Clearing [PET96], por la similitud con el resto de esquemas evolutivos estudiados en este trabajo, y los resultados prometedores alcanzados en diversos problemas. El funcionamiento del algoritmo Clearing se ilustra en la figura 2.59.

El esquema principal del algoritmo Clearing es similar al de un algoritmo genético generacional clásico. La novedad introducida por este método tiene su mayor exponente en la utilización del operador de selección:

Antes de realizar la selección de individuos, el algoritmo Clearing divide la población de soluciones en nichos. En los experimentos realizados en este capítulo, un nicho estará formado por aquellas soluciones cuya distancia Euclídea al elemento central del nicho sea inferior a un valor umbral (radio del nicho). El radio del nicho, notado como R, depende de la distancia Euclídea máxima entre los cromosomas, y de un parámetro C proporcionado por el usuario del algoritmo. El parámetro C simboliza la tasa de amplitud del tamaño de un agrupamiento.

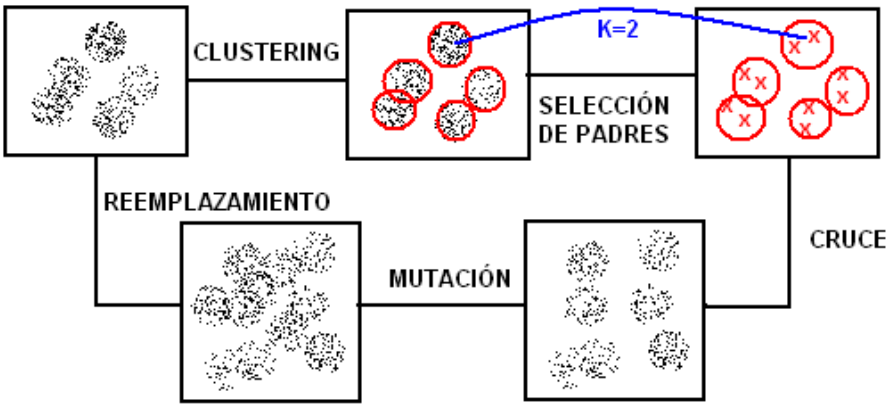

Figura 2.59. Esquema general del algoritmo Clearing

El Algoritmo 2.7 muestra el esquema principal del algoritmo Clearing. Este modelo ha sido utilizado para entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 2000 soluciones evaluadas
- **Probabilidad de cruce:** 1.0
- **Probabilidad de mutación:** 0.01
- **Factor de elitismo:** Los 2 mejores individuos permanecen en la población
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** número real en el intervalo [-5, 5]
- **Operador de selección:** Torneo binario
- **Operador de cruce:** BLX-α (BLX)/Heurístico (HEU)/Aritmético (ARI)
- **Operador de mutación:** Desplazamiento (DES)
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** 8 (Elman), 4 (FRNN) y 11 (Jordan)

Se han realizado 30 ejecuciones del algoritmo Clearing con los parámetros anteriores, para cada tipo de red recurrente estudiado. Las tablas A.10 a A.12 (Anexo I) muestran los resultados del test de normalidad realizado sobre el conjunto de resultados de test, para cada cada combinación de operadores en el algoritmo Clearing.

Los resultados de los tests de equivalencia estadística se muestran en las tablas A.34, A.35 y A.36 para cada modelo de red aplicado al problema CATS, respectivamente. Adicionalmente, la tabla 2.4 muestra un resumen de las mejores combinaciones encontradas, estadísticamente equivalentes, para cada modelo de red y problema.

1. t= 0, P(t)= inicializar población con N soluciones

2. Evaluar P(t)

3. Asignar D= Calcular distancia máxima entre cromosomas

4. R= C·D

5. Mientras (condición de parada = FALSO)

5.1. Asignar t= t+1

5.2. Calcular número de agrupamientos (nichos)

5.3. P'(t)= seleccionar las K mejores soluciones de cada agrupamiento

5.4. P''(t)= Seleccionar N soluciones de P'(t)

5.5. Cruce de soluciones en P'(t), para generar N nuevas soluciones, $\{H\}_N$

5.6. Mutación de soluciones en $\{H\}_N$, con probabilidad p_m

5.7. Evaluar soluciones en $\{H\}_N$

5.8. Reemplazamiento de P(t) con $\{H\}_N$

5.9. Elitismo, si procede

6. Devolver resultados

Algoritmo 2.7. Procedimiento general de Clearing

Tabla 2.4: Mejores combinaciones de operadores para el algoritmo Clearing, en cada tipo de red y subproblema CATS

Problema	**Elman**	**FRNN**	**Jordan**
CATS1	Clearing/BLX	Clearing/ARI	Clearing/BLX
CATS2	Clearing/BLX	Clearing/ARI	Clearing/BLX
CATS3	Clearing/BLX	Clearing/ARI	Clearing/BLX

CATS4	Clearing/BLX	Clearing/ARI	Clearing/BLX
CATS5	Clearing/BLX	Clearing/ARI	Clearing/BLX

Los tests estadísticos han concluido que el mejor operador de cruce entre los considerados, para entrenar las redes de Elman y Jordan en el problema CATS, es el cruce *BLXα*. Sin embargo, el mejor operador de cruce para entrenamiento de redes neuronales recurrentes completas mediante Clearing, es el cruce *aritmético*.

Las figuras 2.60 a 2.74 muestran la evolución de la diversidad de la mejor ejecución del algoritmo Clearing en cada tipo de red, para las soluciones no estadísticamente equivalentes de cada problema.

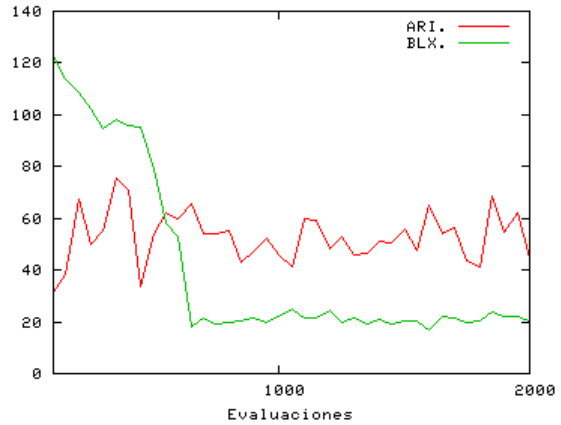

Figura 2.60. Red de Elman Problema CATS1

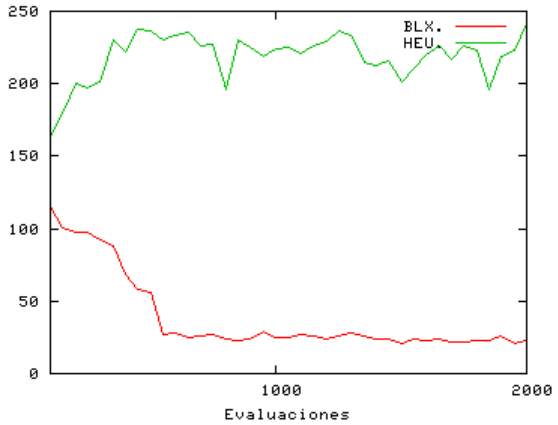

Figura 2.61. Red de Elman Problema CATS2

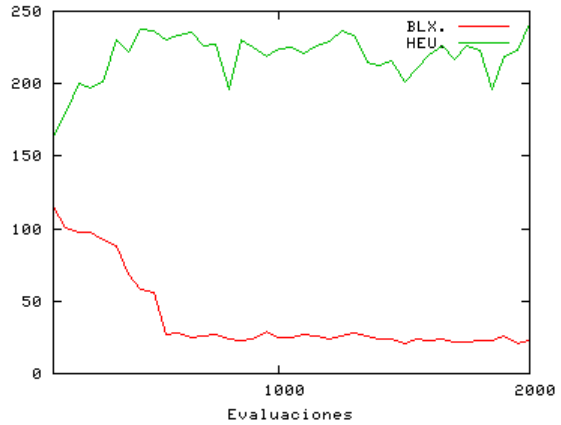

Figura 2.62. Red de Elman Problema CATS3

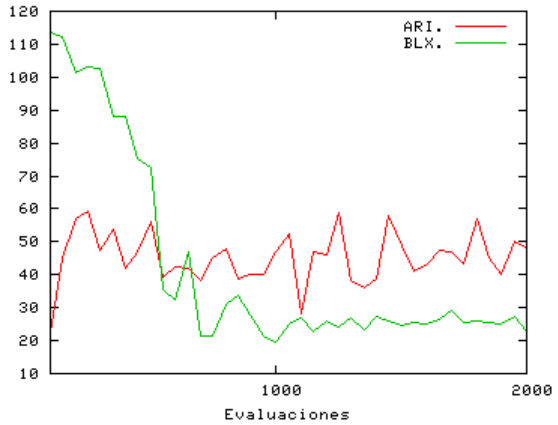

Figura 2.63. Red de Elman Problema CATS4

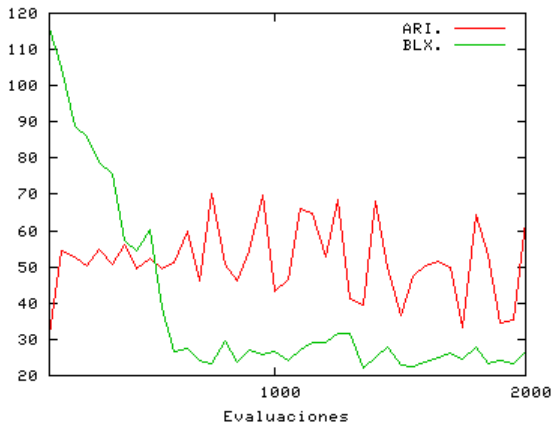

Figura 2.64. Red de Elman Problema CATS5

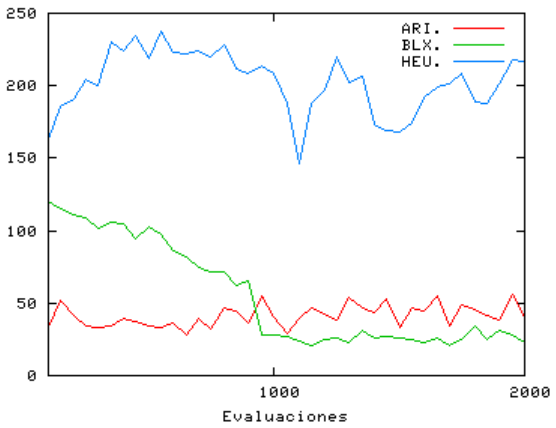

Figura 2.65. Red Completa Problema CATS1

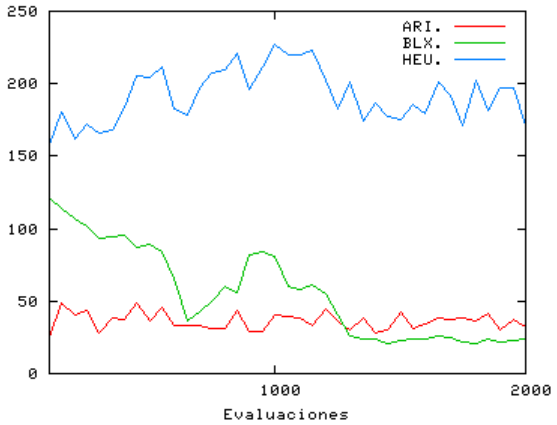

Figura 2.66. Red Completa Problema CATS2

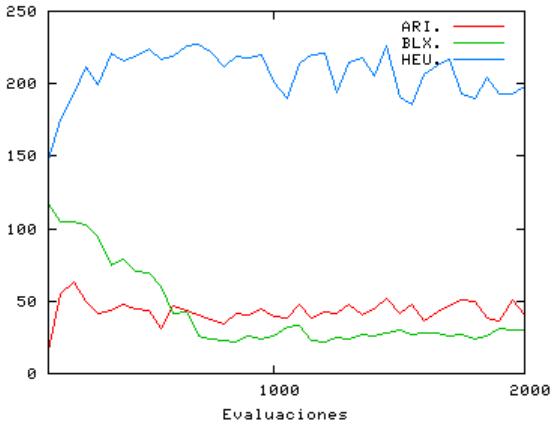

Figura 2.67. Red Completa Problema CATS3

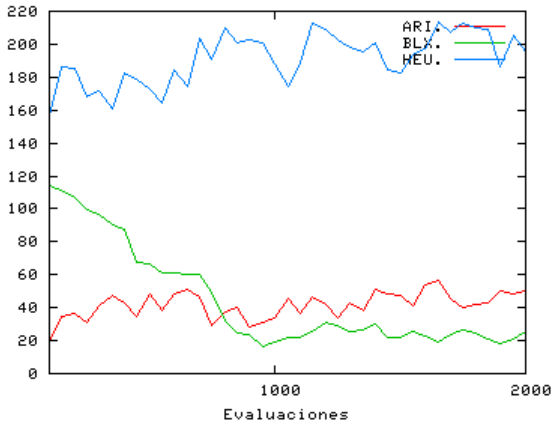

Figura 2.68. Red Completa Problema CATS4

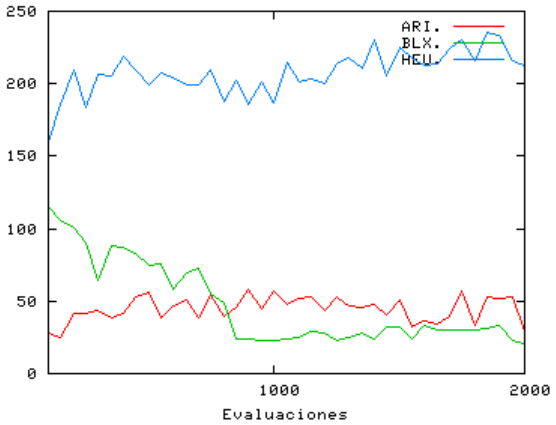

Figura 2.69. Red Completa Problema CATS5

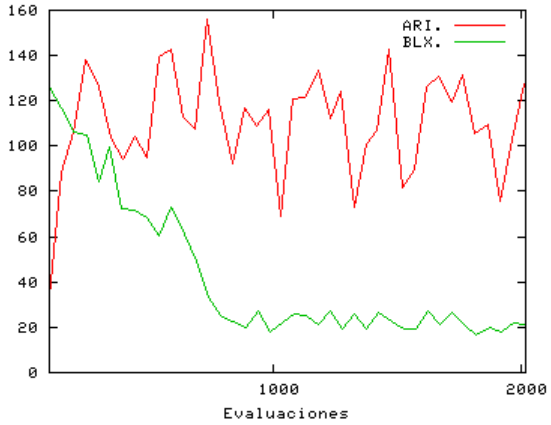

Figura 2.70. Red de Jordan Problema CATS1

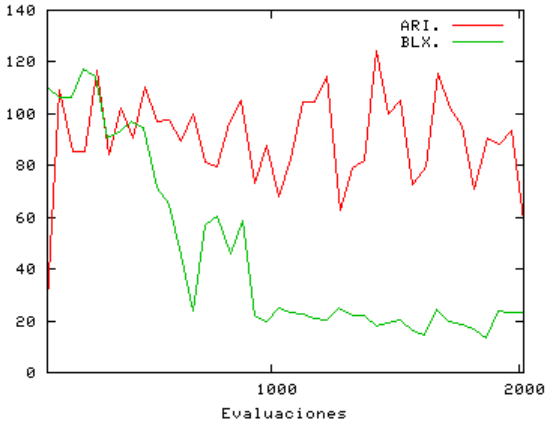

Figura 2.71. Red de Jordan Problema CATS2

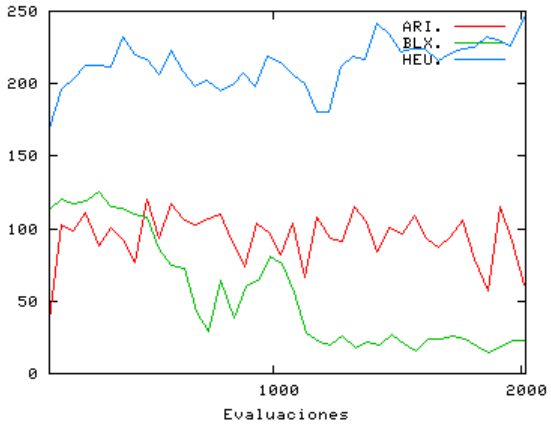

Figura 2.72. Red de Jordan Problema CATS3

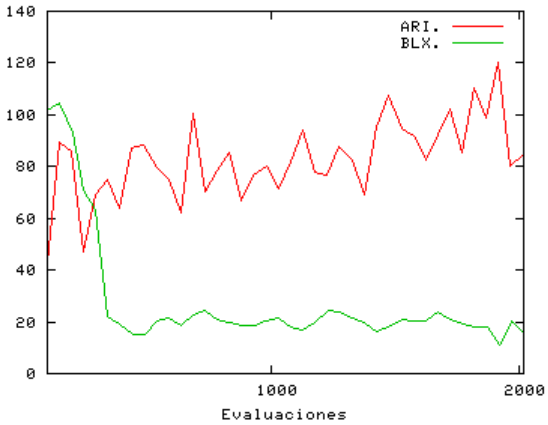

Figura 2.73. Red de Jordan Problema CATS4

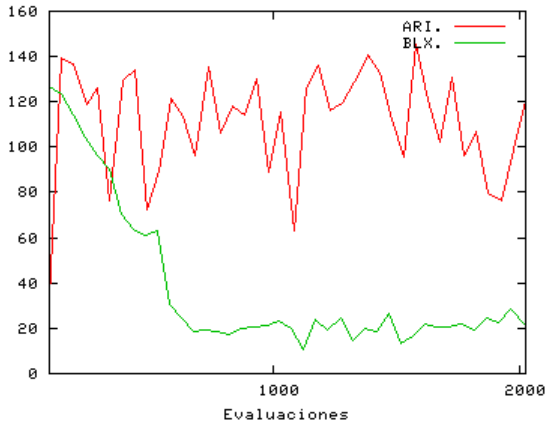

Figura 2.74. Red de Jordan Problema CATS5

3.5. Diversidad y convergencia: Algoritmo CHC para entrenamiento de RNRD

El algoritmo CHC fue una de las primeras propuestas de algoritmo genético que introducía intrínsecamente un equilibrio entre diversidad y convergencia [ESH91]. Este método realiza el control de la diversidad mediante el uso de cuatro componentes básicas:

- **Selección elitista**. La selección elitista consiste en mantener en la población los mejores individuos entre padres e hijos, para la siguiente generación.
- **Cruce uniforme HUX**. El objetivo de este cruce es producir descendencia que sea lo más diferente posible a los padres, manteniendo la información genética dentro de los cromosomas descendientes.
- **Prevención de incesto**. La prevención de incesto evita que dos padres se crucen, si son soluciones similares. Se dice que dos soluciones son similares si la distancia entre los cromosomas es inferior a un umbral.
- **Reinicialización**. La reinicialización consiste en regenerar la población de soluciones, una vez que ésta ha convergido a un espacio de soluciones. Existen diversos mecanismos para llevar a cabo la reinicialización, como por ejemplo generar todos los individuos de forma aleatoria, generar la población a partir de un factor de mutación aplicado a ciertas soluciones de poblaciones anteriores, etc. Además, la nueva población generada debe contener las mejores soluciones generadas hasta el momento, de modo que éstas sigan teniendo efecto sobre el proceso evolutivo.

Aunque la primera propuesta del algoritmo CHC estaba basada en codificación binaria [ESH91], para el desarrollo de los experimentos hemos modificado su estructura inicial de modo que podamos adaptar el algoritmo para utilizarlo con codificación real. Esta idea ha sido adoptada a partir de los prometedores resultados obtenidos al realizar cambios similares en CHC por otros autores [COR03][COR04]. La Figura 2.75 muestra el esquema principal del algoritmo CHC. Las modificaciones realizadas sobre el esquema básico de CHC tienen lugar sobre los siguiente componentes:

- El cruce uniforme HUX es sustituido por un cruce que no reduce la diversidad en la población. Concretamente, utilizamos los cruces *Heurístico* y *BLXα*.
- La medida de similitud entre cromosomas, para el mecanismo deprevención de incesto, se realizará de acuerdo a la distancia Euclídea.

El esquema planteado en el algoritmo 2.8 explica en detalle el funcionamiento del algoritmo CHC.

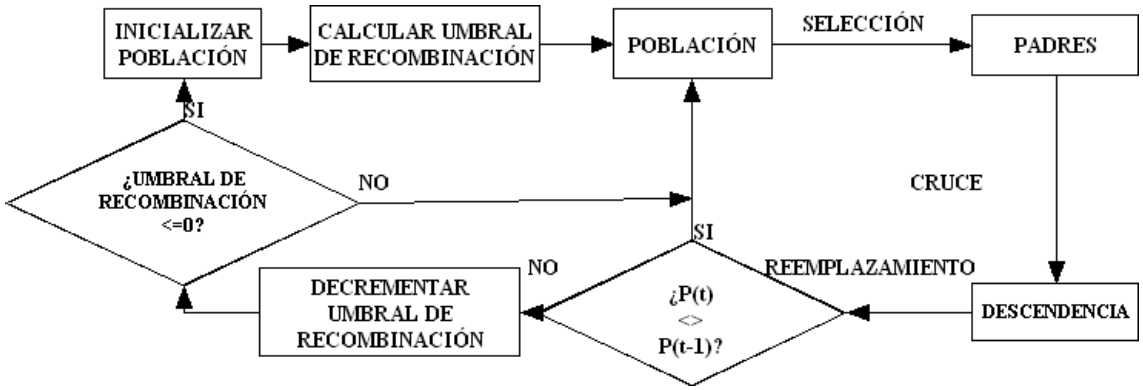

Figura 2.75. Esquema general del algoritmo CHC

El algoritmo CHC ha sido utilizado para entrenar RNRD en el benchmark CATS, utilizando los siguientes parámetros:

- **Criterio de parada:** 2000 soluciones evaluadas
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** número real en el intervalo [-5, 5]
- **Operador de selección:** Torneo binario
- **Operador de cruce:** BLX-α (BLX)/Heurístico (HEU)/Aritmético (ARI)
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** 8 (Elman), 4 (FRNN) y 11 (Jordan)

1. t=0, Inicializar P(t) con N soluciones

2. d= Distancia media entre los cromosomas en P(t)

3. Asignar dec= $\mu \cdot d$

4. Evaluar soluciones en P(t)

5. Mientras (condición de parada = FALSO),

5.1. P'(t)= selección de N padres a partir de P(t)

5.2. $\{H\}_M$ = recombinación de padres en P'(t)

5.3. Evaluar soluciones en $\{H\}_M$

5.4. Si $|\{H\}_M| = 0$, entonces

5.4.1. Asignar d= d-dec

5.5. Si $d \leq 0$

5.5.1. Asignar s= Mejor solución en P(t)

5.5.2. Reinicializar población P(t) con s y N-1 nuevas soluciones

5.5.3. Evaluar soluciones en P(t)

5.5.4.Recalcular valores d y dec

5.6. En otro caso,

5.6.1. Asignar P(t)= Mejores N soluciones entre P(t) y $\{H\}_M$

6. Devolver resultados

Algoritmo 2.8. Procedimiento de general de CHC

Se han realizado 30 ejecuciones del algoritmo CHC con los parámetros anteriores, para cada tipo de red. Las tablas A.13 a A.15 (Anexo I) muestran los resultados del test de normalidad realizado sobre el conjunto de resultados de test para cada combinación de operadores, sobre el algoritmo CHC.

Los resultados de los tests de equivalencia estadística se muestran en las tablas A.37, A.38 y A.39 para cada modelo de red aplicado al problema CATS, respectivamente. Adicionalmente, la tabla 2.5 muestra un resumen de las mejores combinaciones encontradas, estadísticamente equivalentes, para cada modelo de red y problema.

Los resultados de los tests estdísticos muestran que las mejores soluciones se obtienen igualmente utilizando un cruce *heurístico* que *BLX*α. Dado que el cruce aritmético tiende a reducir considerablemente la diversidad en la población, este tipo de recombinación de soluciones no es adecuada para ser utilizada en el algoritmo CHC.

Este hecho se ve reforzado por las gráficas de evolución de diversidad para el algoritmo CHC. Mientras que la diversidad desciende suavemente a lo largo decada iteración al utilizar un cruce *BLX*α, un cruce aritmético implica que descienda rápidamente en las primeras iteraciones del algoritmo. Acto seguido, tiene lugar el mecanismo de reinicialización.

A partir de este momento, la selección actúa excesivamente influenciada por las mejores soluciones encontradas hasta el momento, produciendo fluctuaciones en la diversidad de la población, debido al mecanismo de reinicialización.

Las figuras 2.76 a 2.90 ilustran esta situación, comparando la evolución de la diversidad sobre las mejores soluciones que utilizan un cruce BLX, y las mejores que utilizan un cruce aritmético.

Tabla 2.5: Mejores combinaciones de operadores para el algoritmo CHC,en cada tipo de red y subproblema CATS

Problema	**Elman**	**FRNN**	**Jordan**
CATS1	CHC/BLX	CHC/HEU	CHC/HEU
	CHC/HEU	CHC/BLX	CHC/BLX
CATS2	CHC/BLX	CHC/BLX	CHC/BLX
	CHC/HEU		CHC/HEU
CATS3	CHC/HEU	CHC/HEU	CHC/HEU
	CHC/BLX	CHC/BLX	CHC/BLX
CATS4	CHC/BLX	CHC/BLX	CHC/BLX
	CHC/HEU	CHC/HEU	CHC/HEU

CATS5	CHC/BLX/ CHC/HEU	CHC/BLX CHC/HEU	CHC/BLX CHC/HEU

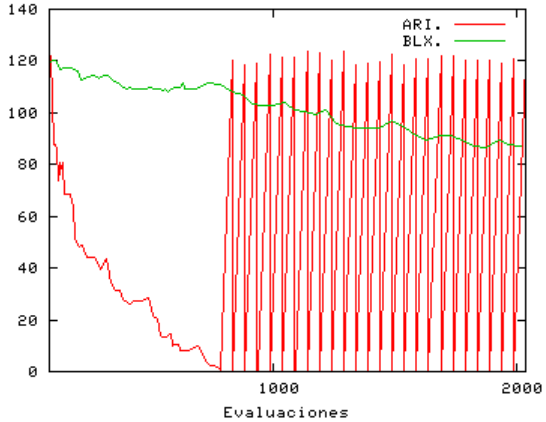

Figura 2.76. Red de Elman Problema CATS1

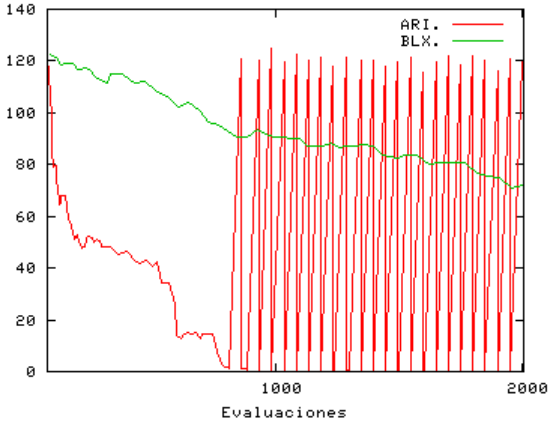

Figura 2.77. Red de Elman Problema CATS2

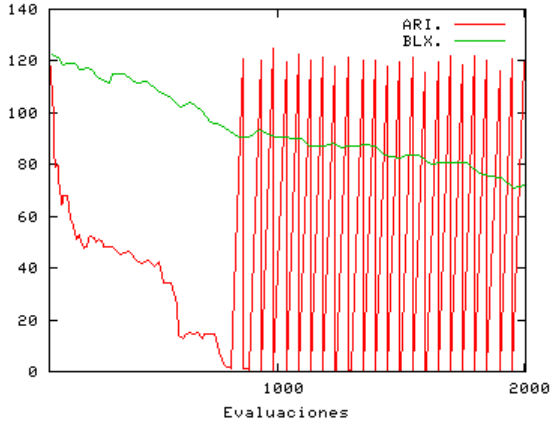

Figura 2.78. Red de Elman Problema CATS3

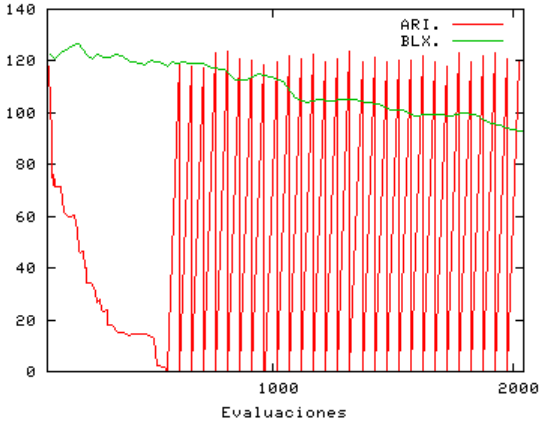

Figura 2.79. Red de Elman Problema CATS4

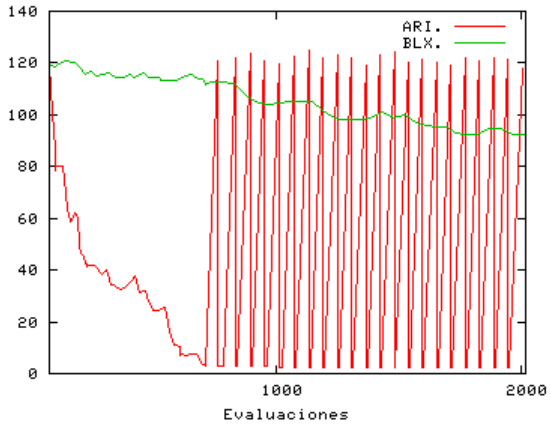

Figura 2.80. Red de Elman Problema CATS5

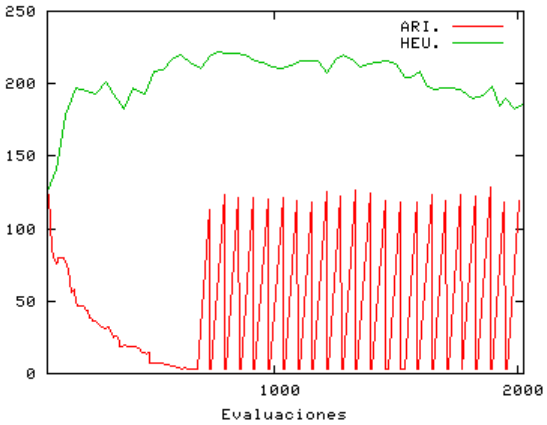

Figura 2.81. Red Completa Problema CATS1

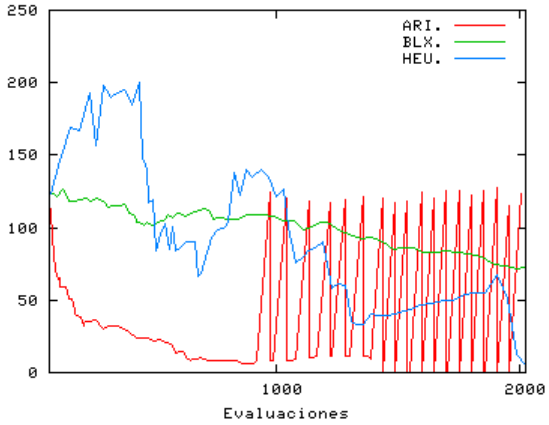

Figura 2.82. Red Completa Problema CATS2

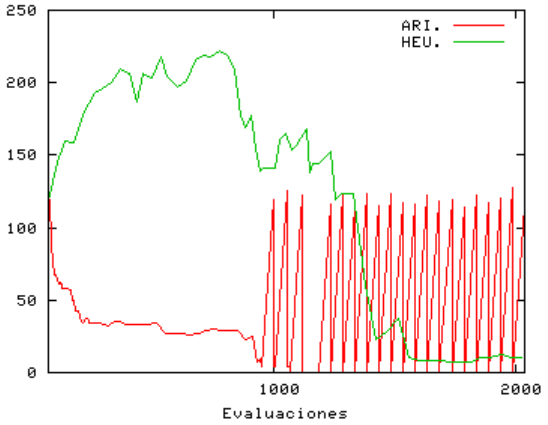

Figura 2.83. Red Completa Problema CATS3

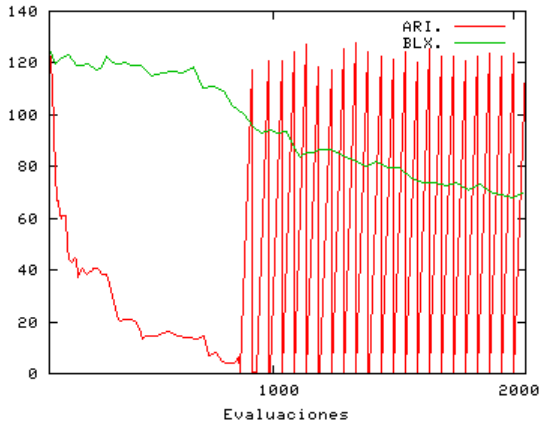

Figura 2.84. Red Completa Problema CATS4

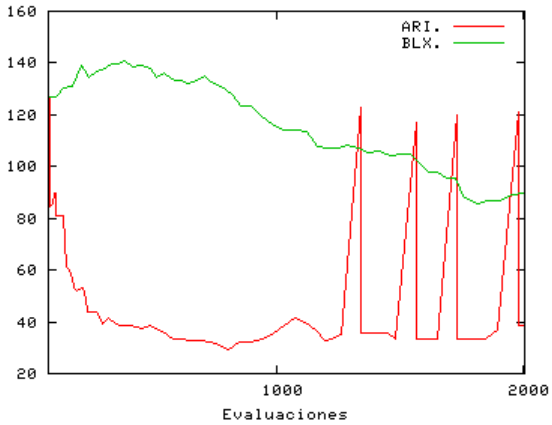

Figura 2.85. Red Completa Problema CATS5

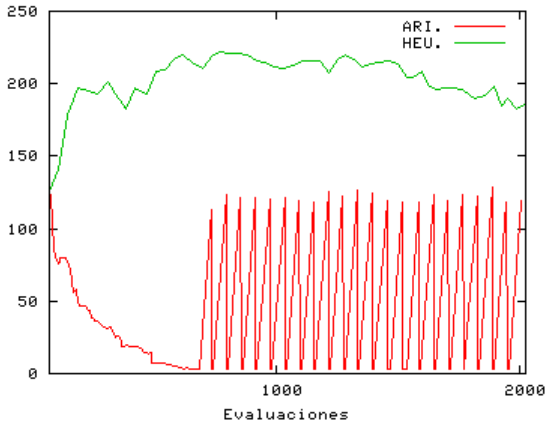

Figura 2.86. Red de Jordan Problema CATS1

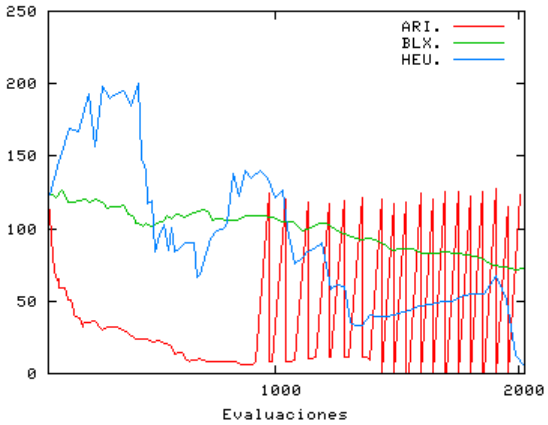

Figura 2.87. Red de Jordan Problema CATS2

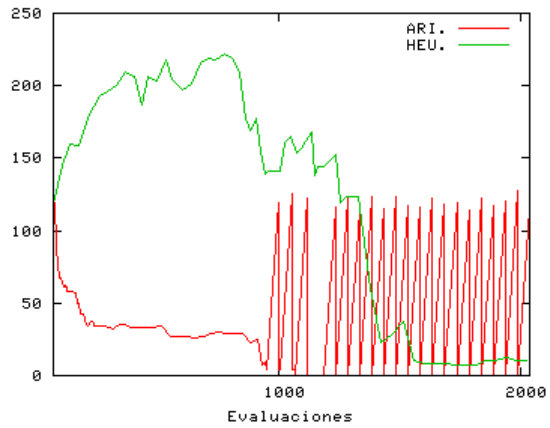

Figura 2.88. Red de Jordan Problema CATS3

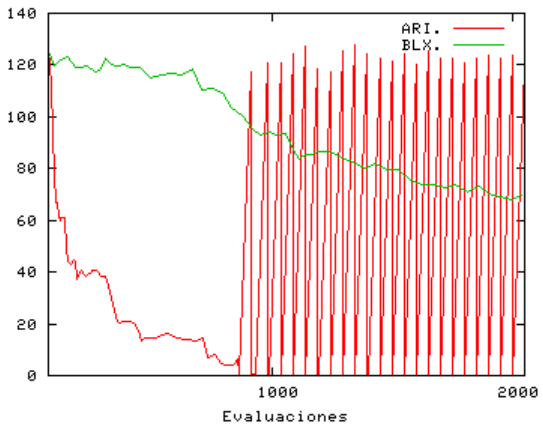

Figura 2.89. Red de Jordan Problema CATS4

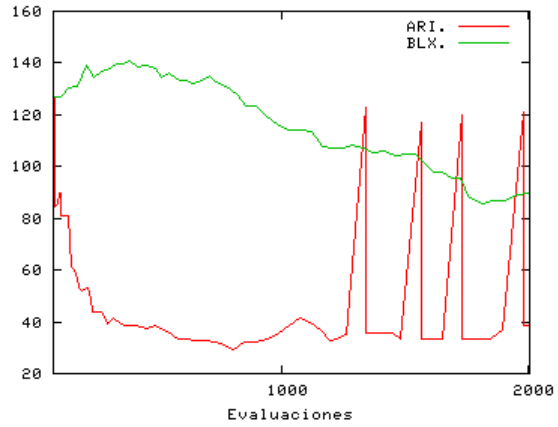

Figura 2.90. Red de Jordan Problema CATS5

4. Consideraciones finales.

Para terminar este capítulo y a modo de resumen, presentamos unanálisis general de los algoritmos que mejor comportamiento han presentado, para entrenamiento de RNRD en el problema CATS. En primer lugar, haremos un estudio estadístico sobre el conjunto de los algoritmos que mejores resultados han obtenido considerando cada modelo de red por separado. Seguidamente, contemplaremos todos los resultados en conjunto, para conocercuál es el modelo de red que ha aprendido un mejor comportamiento.

Las tablas A.40 a A.42 (Anexo I) muestran los tests de equivalencia de los algoritmos genéticos, Clearing y CHC para los diferentes modelos de redes neuronales recurrentes estudiados. Adicionalmente, la tabla 2.6 muestra un resumen de los mejores algoritmos de entrenamiento, ordenados por red y problema.

Tabla 2.6: Algoritmos que mejores resultados han proporcionado, al entrenarRNRD en el problema CATS

Problema	**Elman**	**FRNN**	**Jordan**
CATS1	MGA/PE/TO/BLX/ALE Clearing/BLX	MGA/PE/LI/BLX/ALE Clearing/BLX GGA/TO/BLX/DES	MGA/PE/LI/HEU/ALE
CATS2	MGA/PE/LI/BLX/ALE	GGA/TO/ARI/ALE MGA/PE/TO/BLX/ALE	MGA/PE/LI/BLX/ALE Clearing/BLX
CATS3	MGA/PE/TO/HEU/ALE	MGA/PE/EQ/ARI/ALE Clearing/BLX GGA/TO/ARI/ALE	MGA/PE/LI/HEU/ALE
CATS4	MGA/PE/LI/HEU/ALE Clearing/BLX	MGA/PE/LI/BLX/ALE GGA/TO/ARI/ALE	MGA/PE/LI/HEU/ALE
CATS5	MGA/PE/LI/BLX/ALE Clearing/BLX	MGA/PE/EQ/ARI/ALE Clearing/BLX	Clearing/BLX MGA/PE/LI/HEU/ALE

Según los resultados mostrados en la tabla 2.6, el esquema evolutivo que mejores resultados ha proporcionado ha sido el modelo genético mixto, juntos con los operadores comentados en el apartado 3.3. No obstante, el algoritmo Clearing también ha producido resultados muy prometedores en la mayoría de los problemas con elcruce $BLX\alpha$.

La tabla 2.7 muestra los resultados del test de equivalencia de los algoritmos de entrenamiento que mejores resultados han proporcionado, sin discriminar entre modelos de red. La tabla está ordenada de mejor a peor algoritmo y red, utilizando el test de equivalencia estadística con un nivel de confianza de 0.05. Notamos con *(x)* aquellos métodos que son estadísticamente equivalentes al algoritmo inmediatamente superior, mientras que el símbolo *(-)* diferencia a aquellos algoritmos que son estadísticamente peores que el grupo de algoritmos equivalentes inmediatamente anterior.

Tabla 2.7: Test de KruskalWallis de los mejores resultados proporcionados para entrenar redes recurrentes dinámicas, en el problema CATS

Algoritmos	CATS1	Algoritmos	CATS2
MGA/PE/LI/BLX/ALE FRNN	*1 (x)*	GGA/TO/ARI/ALE FRNN	*1 (x)*
MGA/PE/TO/BLX/ALE Elman	*1.208e-05 (-)*	MGA/PE/LI/BLX/ALE Elman	*1.415e-07 (-)*
MGA/PE/LI/HEU/ALE Jordan	*3.214e-08 (-)*	MGA/PE/LI/BLX/ALE Jordan	*2.872e-11 (-)*

Tabla 2.7: (Continuacion)

Algoritmos	CATS3	Algoritmos	CATS4
MGA/PE/EQ/ARI/ALE FRNN	*1 (x)*	MGA/PE/LI/BLX/ALE FRNN	*1 (x)*
MGA/PE/LI/HEU/ALE Elman	*0.0002759 (-)*	MGA/PE/LI/HEU/ALE Elman	*4.99e-07 (-)*
MGA/PE/LI/HEU/ALE Jordan	*1.266e-10 (-)*	MGA/PE/LI/HEU/ALE Jordan	*0.002822 (-)*

Tabla 2.7: (Continuacion)

Algoritmos	CATS5
MGA/PE/EQ/ARI/ALE FRNN	*1 (x)*
MGA/PE/LI/BLX/ALE Elman	*6.282e-07 (-)*
Clearing/BLX Jordan	*3.175e-11 (-)*

La red que mejor ha conseguido aprender los problemas planteados ha sido la red recurrente completa. Además, los tests de equivalencia estadística concluyen que hay una gran diferencia entre aplicar un modelo de red u otro, ya que los resultados obtenidos por cada modelo de red difieren entre sí en gran medida.

Así, podemos concluir que el modelo evolutivo que mejores resultados ha proporcionado es el modelo mixto, con reemplazamiento a los peores individuos de la población. Mientras que este tipo de reemplazamiento supone una fuerte convergencia a un espacio de soluciones concreto, la aplicación del operador de mutación a todas las soluciones de la población no asegura la diversidad y exploración de nuevas zonas del espacio de búsqueda.

El operador de mutación que mejores resultados ha proporcionado en este esquema es la mutación aleatoria, en combinación con un cruce *heurístico* o *BLX*α (redes de Elman y Jordan), o un cruce *aritmético* (redes recurrentes completas). En cuanto al operador de selección, podemos encontrar buenas soluciones utilizando tanto la selección por torneo binario como la lineal, dependiendo del problema que se aborde.

Capítulo 3

Métodos evolutivos híbridos para entrenamiento de Redes Neuronales Recurrentes Dinámicas.

1. Introducción.

En el capítulo anterior hemos estudiado diversas técnicas evolutivas, aplicadas al entrenamiento de redes neuronales recurrentes. Los experimentos realizados han mostrado que los algoritmos evolutivos solucionan parcialmente el problema de los algoritmos de entrenamiento clásicos, basados en el gradiente. El objetivo de este capítulo es introducir una clase de algoritmos híbridos que combine ambas técnicas, de modo que se exploten las ventajas de cada metodología y se puedan conseguir mejores soluciones.

Mientras que los algoritmos evolutivos realizan una búsqueda heurística a lo largo del espacio de soluciones, los algoritmos clásicos de entrenamiento dirigen la búsqueda basándose en la información proporcionada por el gradiente. La combinación entre algoritmos evolutivos y métodos de programación no lineal se propone como una opción prometedora para entrenar RNRD, dando como resultado algoritmos híbridos de tipo memético.

Los algoritmos meméticos son, esencialmente, la hibridación de un algoritmo genético con un operador de búsqueda local [MEL03][MOL04][MOS03][RAD04]. Mientras que un cromosoma (en algoritmos evolutivos) contiene información genética de sus antecesores, el

mismo cromosoma (en algoritmos meméticos) contiene, además, información autoadaptada por el propio individuo. Esta autoadaptación es implementada mediante un operador de búsqueda local, el cual se encarga de mejorar la solución codificada en el cromosoma dentro del vecindario de la propia solución.

Los algoritmos meméticos han sido empleados para resolver diversos problemas de optimización, obteniendo resultados muy prometedores. Dada la gran cantidad y variedad de publicaciones realizadas en los últimos años sobre algoritmos meméticos, es difícil realizar una selección de los trabajos que mayor impacto han supuesto en el área. Podemos destacar algunos trabajos relevantes ampliamente citados en la literatura o novedosos [BER04][CAO03][HER04][MER99][VIV01], aunque podemos encontrar una mayor información sobre estos métodos en la página web principal de algoritmos meméticos http://www.densis.fee.unicamp.br/~moscato/memetic_home.html En esta página web podemos encontrar una amplia información sobre este tipo de técnicas, aplicaciones a diversos problemas, enlaces a código fuente, etc.

Debido a que los algoritmos meméticos pueden incorporar las ventajas de los algoritmos evolutivos y de los métodos basados en el gradiente, en este capítulo se presentan como una alternativa para mejorar el entrenamiento de RNRD.

En estudios recientes, se han realizado propuestas del mismo tipo para entrenamiento de redes neuronales feedforward, obteniendo resultados adecuados [CHE99][CON02][KU97] [MAR03][MAR04][MAR06][PRU02][SHI99]. La propuesta que realizamos en este capítulo consiste en la hibridación de un algoritmo genético con esquema estacionario y el algoritmo CHC modificado expuesto en el capítulo anterior, junto con un operador de búsqueda local. El objetivo perseguido es encontrar una configuración de algoritmo híbrido que pueda servir como base para mejorar el entrenamiento de una RNRD. Para ello, haremos un estudio de diferentes técnicas de hibridación junto con métodos de programación no lineal [SCH95].

El capítulo está organizado de la siguiente forma: El apartado 2 expone las diferentes estrategias de hibridación consideradas. A continuación, el apartado 3 hace un estudio de algoritmos de programación no lineal para seleccionar el más adecuado para utilizarlo como operador de búsqueda local. El apartado 4 muestra las diferentes propuestas de algoritmos evolutivos híbridos que realizamos. Seguidamente, el apartado 5 hace mención a otros métodos

híbridos que hemos considerado para entrenar RNRD. Por último, el apartado 6 concluye con unas notas finales sobre el estudio realizado.

2. Estrategias de hibridación.

En la literatura puede encontrarse dos tipos de estrategias de hibridación básicas [MEL03][MOS03], similares en cuanto a la forma de mejorar unasolución dentro de un proceso evolutivo, pero que difieren en cómo afecta esta mejora al propio proceso de evolución. Estas son:

- **Lamarckiana.** Esta estrategia consiste en mejorar la función de adecuación de una solución mediante un operador de búsqueda local, generada tras los operadores evolutivos de cruce y mutación. A continuación, se evalúa la solución. El conocimiento adquirido en la búsqueda local es entonces transmitido al cromosoma y, consecuentemente, incorporado al proceso evolutivo. El valor de adecuación del individuo es también modificado con el valor de adecuación calculadotras la búsqueda local. La Figura 3.1 muestra un ejemplo de la actuación deloperador de búsqueda local sobre un cromosoma que codifica una red recurrente completa, utilizando la estrategia lamarckiana.

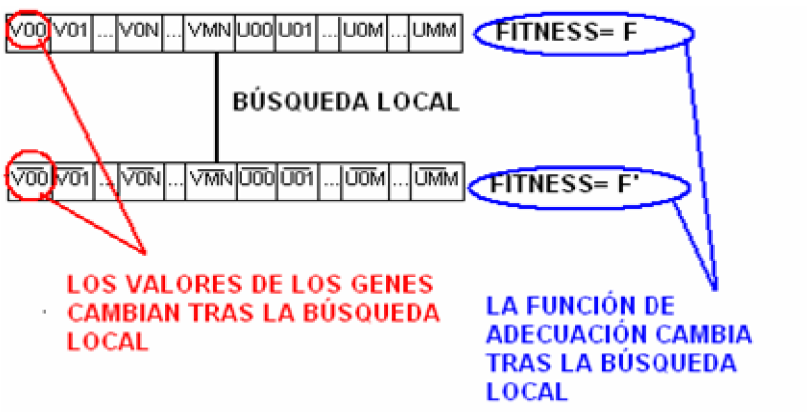

Figura 3.1. Ejemplo de uso de la estrategia Lamarckiana

- **Baldwiniana**. Esta estrategia, al igual que la *lamarckiana*, consiste en mejorar la función de adecuación de una solución mediante un operador de búsqueda local trasel proceso evolutivo, y antes de ser evaluada. Tras el proceso de mejora, elvalor de la función de adecuación de la solución original es sustituido por el de la soluciónmejorada. La Figura 3.2 muestra un ejemplo de la utilización de la estrategia de hibridación baldwiniana sobre un cromosoma que codifica a una red recurrente dinámica completa.

Como puede observarse, la diferencia entre ambas estrategias consiste en que en la estrategia *baldwiniana* el conocimiento adquirido no es transmitido a los cromosomas tras la búsqueda local. Sin embargo, el valor de adecuación del individuo sí quees modificado con el valor de adecuación calculado tras la búsqueda local.

Figura 3.2. Ejemplo de uso de la estrategia Baldwiniana

El efecto que produce la aplicación de la estrategia baldwiniano es proporcionar al proceso evolutivo información sobre lo adecuada que puede ser, potencialmente, una solución para resolver el problema en curso. Sin embargo, la aplicación del método lamarckiano incorpora directamente la solución mejorada al mecanismo de evolución.

3. Selección del operador de búsqueda local.

Puesto que un algoritmo memético es la hibridación de un algoritmo evolutivo con uno de búsqueda local, es razonable que las características y efectividad del algoritmo memético

esten fuertemente condicionadas por el algoritmo de búsqueda local que se emplee. Este apartado realiza un estudio de diversos algoritmos de programación no lineal, junto con los algoritmos clásicos para entrenamiento de RNRD. Como resultado de este estudio, seleccionaremos el algoritmo que mejores resultados ha proporcionado para construir los algoritmos híbridos.

Los métodos estudiados son los algoritmos BPTT, RTRL (capítulo 1) y los algoritmos Quasi-Newton basados en las fórmulas BFGS [BYR95][ZHU97] y de Levenberg-Marquardt [CHE99][PRU02].

3.1. Algoritmo de Levenberg-Marquardt.

El algoritmo de Levenberg-Marquadt (LM) es un método numérico de optimización no lineal, basado en la aproximación del valor de un conjunto de variables mediante un método similar a Newton-Raphson. Para entrenar una red neuronal recurrente dinámica mediante el algoritmo de LM, formularemos el funcionamiento dinámico de lared mediante un conjunto de ecuaciones no lineales de la forma:

$$D(t) = F(S(t), X(t)) \tag{3.1}$$

donde:

- *D(t)* es la respuesta deseada para las neuronas de salida, en el instante *t*.
- *F(·)* es la función de activación de las neuronas de salida.
- *S(t)* es un vector que contiene el estado de las neuronas ocultas de la red, en el instante *t*.
- *X(t)* es un vector que contiene los valores de entrada a la red, en el instante *t*.

El mecanismo de funcionamiento del algoritmo LM consiste en minimizar una función objetivo, dependiente de los pesos de la red y la salida esperada para la misma, para cada instante *t*. Esta idea se ilustra en la siguiente ecuación:

$$G(s)=\sum_{t=0}^{T}(\|D(t)-O(t)\|)^2=E^T E \tag{3.2}$$

$$E=[e(0),e(1),...,e(T)]^T$$

$$e(t)=D(t)-O(t)$$

donde:

- *D(t)* es la salida deseada en el instante t por la neurona de salida.
- *O(t)* es la salida de la red en el instante t.
- *E* es una matriz de *T*x1 valores, que contiene el error de las neuronas de salida en cada instante *t*.
- *s* es el vector de variables a optimizar.

Dependiendo del modelo de red a entrenar, el vector *s* tendrá la estructura que se muestra en las ecuaciones 3.3 (Elman), 3.4 (completa) y 3.5 (Jordan).

$$s=\left(V_{11},\ldots,V_{1n},V_{21}\ldots,V_{2n},\ldots V_{hn},U_{11},\ldots,U_{hh},W_{11},\ldots,W_{oh}\right)_Q \tag{3.3}$$

$$s=\left(V_{11},\ldots,V_{1n},V_{21}\ldots,V_{2n},\ldots V_{hn},U_{11},\ldots,U_{hh}\right)_Q \tag{3.4}$$

$$s=\left(V_{11},\ldots,V_{1n},V_{21}\ldots,V_{2n},\ldots V_{hn},U_{11},\ldots,U_{ho},W_{11},\ldots,W_{oh}\right)_Q \tag{3.5}$$

Para entrenar una red neuronal recurrente dinámica mediante estemétodo, asumimos una representación matricial que iguala cada salida de la red alas salidas deseadas, en cada instante *t*:

$$\begin{bmatrix} Y(0) \\ \ldots \\ Y(t) \\ \ldots \\ Y(T) \end{bmatrix}=\begin{bmatrix} F(S(0),X(0)) \\ \ldots \\ F(S(t),X(t)) \\ \ldots \\ F(S(T),X(T)) \end{bmatrix} \tag{3.6}$$

El algoritmo LM actualiza cada variable a optimizar en la iteración *i*, de acuerdo con el siguiente esquema:

$$s(i+1)=s(i)-(J_i^T J_i + \mu I)^{-1} J_i^T E_i \tag{3.7}$$

donde *J* es la matriz Jacobiana de *G(s)*:

$$J=\begin{bmatrix} \frac{\partial e(0)}{\partial s_0}, \frac{\partial e(0)}{\partial s_1}, \dots \frac{\partial e(0)}{\partial s_Q} \\ \frac{\partial e(1)}{\partial s_0}, \frac{\partial e(1)}{\partial s_1}, \dots \frac{\partial e(1)}{\partial s_Q} \\ \dots \\ \frac{\partial e(T)}{\partial s_0}, \frac{\partial e(T)}{\partial s_1}, \dots \frac{\partial e(T)}{\partial s_Q} \end{bmatrix} \tag{3.8}$$

Cuando el valor $\mu = 0$, la fórmula es equivalente al método de Newton-Raphson, mientras que para valores altos de μ la fórmula llega a asemejarse a una variante del algoritmo *BackPropagation*.

3.2. Algoritmo BFGS.

El algoritmo BFGS es un método clásico de optimización no lineal que realiza una actualización iterativa de la solución al problema utilizando un método Quasi-Newton. El algoritmo inicial fue propuesto en 1970 por Broyden, Fletcher, Goldfarb y Shanno. Desde entonces han surgido diversas variantes del mismo, adaptadas para resolución de problemas con restricciones, incorporando tratamiento de memoria limitada, etc. El funcionamiento básico del método se ilustra en el esquema del Algoritmo 3.1, donde:

- s_k representa la solución al problema en la iteración *k*.
- g_k se corresponde con el gradiente de la función a optimizar evaluado en el punto s_k, $\nabla f(s_k)$.
- La matriz H_k representa la inversa de la matriz Hessiana, de tamaño (n,n).

- Los valores d_k y A representan la dirección de búsqueda y el tamaño del paso en la actualización, respectivamente.

El algoritmo BFGS se caracteriza por la actualización que realiza sobre H_k. Este cálculo se realiza de acuerdo a la siguiente ecuación:

$$H_{k+1} = H_k + \frac{(s_{k+1} - s_k)(s_{k+1} - s_k)^T}{(s_{k+1} - s_k)^T(\nabla f(s_{k+1}) - \nabla f(s_k))} - \frac{H_k(\nabla f(s_{k+1}) - \nabla f(s_k))(\nabla f(s_{k+1}) - \nabla f(s_k))^T H_k}{(\nabla f(s_{k+1}) - \nabla f(s_k))^T H_k(\nabla f(s_{k+1}) - \nabla f(s_k))} \quad (3.9)$$

De acuerdo con la notación introducida en el apartado anterior, el gradiente de la función de error se representa en forma matricial siguiendo el mismo esquema utilizado para la solución s. Se calcula según muestran las ecuaciones 3.10 a 3.12 para redes recurrentes de Elman, completas y de Jordan, respectivamente.

$$\nabla G = \left(\frac{\partial}{\partial V_{ij}} G, \frac{\partial}{\partial U_{ir}} G, \frac{\partial}{\partial W_{ki}} G \right) \quad (3.10)$$

$$\nabla G = \left(\frac{\partial}{\partial V_{ij}} G, \frac{\partial}{\partial U_{ir}} G \right) \quad (3.11)$$

$$\nabla G = \left(\frac{\partial}{\partial V_{ij}} G, \frac{\partial}{\partial U_{ir}} G, \frac{\partial}{\partial W_{ki}} G \right) \quad (3.12)$$

Cada componente del vector que representa el gradiente de la solución se calcula de la siguiente forma, para cada tipo de red:

$$\frac{\partial}{V_{ij}} G = \sum_{t=1}^{T} \partial S_i(t) \cdot X_j(t)$$

$$\frac{\partial}{W_{ki}} G = \frac{2}{T} \sum_{t=1}^{T} (O_k(t) - d_k(t)) \cdot S_j(t); \textit{ red de Elman y Jordan}$$

$$\frac{\partial}{U_{ir}} G = \begin{cases} \sum_{t=1}^{T} (\partial S_i(t) S_r(t)) \cdot S_r(t-1); \textit{ red de Elman y Completa} \\ \sum_{t=1}^{T} (\partial S_i(t) S_r(t)) \cdot O_r(t-1); \textit{ red de Jordan} \end{cases}$$

0. Entradas al algoritmo:

s= vector de N variables, conteniendo una solución inicial

F= Función a optimizar

1. Asignar K=0, $s_k = s$

2. Mientras (condición de parada = FALSO)

2.1. g_k *= Gradiente de F evaluado en* s_k

2.2. Calcular matriz H_k

2.3. Calcular dirección de búsqueda, $d_k = -H_k g_k$

2.4. Asignar $\alpha = \arg\min_{\alpha>0} \{f(s_k + \alpha d_k)\}$

2.5. Asignar $s_{k+1} = s_k + \alpha d_k$

2.6. $k = k+1$

3. Devolver resultados

Algoritmo 3.1. Procedimiento de optimización mediante la fórmula BFGS

3.3. Experimentación.

Los algoritmos BPTT, RTRL, BFGS y LM han sido utilizados para entrenar los modelos de RNRD en el benchmark CATS. Se han realizado 30 ejecuciones multiarranque de cada algoritmo por cada problema y red. El criterio de parada utilizado para cada algoritmo es el tiempo de ejecución, habiendo fijado este a 10 segundos.

Las tablas A.43 a A.45 (Anexo I) muestran los resultados de la comparación estadística entre los algoritmos propuestos. La tabla 3.1 muestra un resumen del test estadístico, discriminando por tipo de problema y red.

Tabla 3.1: Clasificación de los algoritmos de entrenamiento que mejores resultados han obtenido estadísticamente, para cada problema y red.

Problema	Elman	FRNN	Jordan
CATS1	*BFGS* *LM*	*BFGS* *LM*	*BFGS*
CATS2	*BFGS*	*LM* *BFGS*	*BFGS*
CATS3	*LM* *BFGS*	*BFGS* *LM*	*BFGS*
CATS4	*BFGS*	*LM*	*BFGS*
CATS5	*BFGS*	*LM* *BFGS*	*BFGS*

Podemos observar que, por regla general, el algoritmo BFGS ha obtenido los mejores resultados al entrenar las redes de Elman y Jordan, aunque el método LM lo ha tenido para las redes recurrentes completas.

Debido a que el método BFGS ha presentado un buen comportamiento (en comparación con los demás algoritmos), para entrenar los diferentes modelos deredes, en la construcción de algoritmos híbridos haremos uso de este algoritmo como operador de búsqueda local.

4. Entrenamiento evolutivo híbrido de RNRD.

Este apartado muestra las diferentes propuestas de algoritmos híbridos que hemos desarrollado para entrenar RNRD. Considerando las estrategias dehibridación presentadas en el apartado 4.2, proponemos cuatro posibles variantes de algoritmos híbridos:

1. *SGALAMBFGS:* hibridación de un algoritmo genético con búsqueda local mediante BFGS, utilizando estrategia lamarckiana.

2. *SGABALBFGS:* hibridación de un algoritmo genético con búsqueda local mediante BFGS, utilizando estrategia baldwiniana.
3. *CHCLAMBFGS:* hibridación del algoritmo CHC con búsqueda local mediante BFGS, utilizando estrategia lamarckiana.
4. *CHCBALBFGS:* hibridación del algoritmo CHC con búsqueda local mediante BFGS, utilizando estrategia baldwiniana.

A continuación, mostramos los esquemas de los algoritmos meméticos propuestos.

4.1. Hibridaciones genéticas estacionarias con métodos de programación no lineal (SGABFGS)

Las hibridaciones SGABFGS se generan al combinar un algoritmo genético y el método de optimización BFGS. Entre las tres estrategias básicas consideradas en algoritmos genéticos (generacional, estacionaria y mixta), en la sección experimental de este trabajo hemos utilizado la estrategia estacionaria. Esta modalidad de hibridación ha sido la que mejores resultados ha obtenido; no obstante, las secciones 5 y 6 de este capítulo profundizan en otros esquemas híbridos considerados y ayudan a justificar la selección de este esquema evolutivo como propuesta de hibridación. El funcionamiento general del algoritmo híbrido se muestra en la figura 3.3.

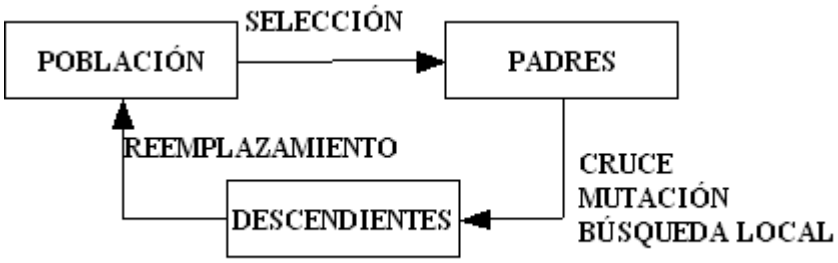

Figura 3.3. Esquema memético general

Partiendo de la estructura mostrada en la figura 3.3, podemos construir el algoritmo híbrido de acuerdo con el esquema mostrado en el Algoritmo 3.2, utilizando la siguiente notación:

- P es una población que contiene N soluciones.
- H es una solución descendiente,
- H' es la solución resultante de aplicar el algoritmo de búsqueda local sobre H.

El Algoritmo 3.2 puede ser utilizado tanto con la estrategia lamarckiana como con la baldwiniana. Comienza inicializando la población P con N soluciones, generadas aleatoriamente a partir de una distribución uniforme en el intervalo $[L_m, L_M]$. El paso 2 evalúa las soluciones generadas, para calcular el valor de adecuación de cada solución. El paso 3 engloba el cuerpo principal del algoritmo. Este paso se itera hasta que se cumpla la condición de parada requerida.

El paso 3.1 selecciona un total de K soluciones en P con base a la función objetivo de cada solución, para ser padres de las nuevas soluciones a generar. Aquí hemos utilizado $K=2$. El paso 3.2 realiza la recombinación de los padres seleccionados mediante un operador de cruce, para generar K nuevas soluciones (modelo genético estacionario).

Tras el cruce, el paso 3.3 aplica un operador de mutación sobre los nuevos individuos generados, $\{H\}_K$, teniendo en cuenta que un *mem* es modificado con probabilidad p_m. Al finalizar el proceso evolutivo de selección, cruce y mutación, el algoritmo memético realiza una búsqueda local sobre cada nueva solución generada, un total de I_l iteraciones, generando un nuevo conjunto de soluciones, $\{H'\}_K$. Estas se evalúan en los pasos 3.4.2-3.4.3 para calcular su valor de adecuación.

En este punto, se llevan a cabo los pasos clave de la hibridación. Si sólo se ejecuta el paso 3.4.3, entonces se aplica la estrategia baldwiniana. El valor de adecuación de una solución H es modificado con el valor de adecuación de la solución H'. Por otra parte, si se aplica la estrategia lamarckiana, el paso 3.4.5 modifica el cromosoma de la solución H reemplazándolo por el cromosoma de la solución H'.

Para concluir el paso 3, los individuos descendientes, $\{H\}_K$, reemplazan a las soluciones de la población P para la siguiente iteración del algoritmo, manteniendo una estrategia elitista en

caso de ser requerida. Una vez que ha finalizado el proceso evolutivo, el algoritmo devuelve la mejor solución encontrada.

1. Inicializar población P con N soluciones

2. Evaluar soluciones en P

3. Mientras (condición de parada = FALSO), hacer:

3.1. Seleccionar K padres

3.2. Recombinar padres para generar K nuevas soluciones, $\{H\}_K$

3.3. Mutar las nuevas soluciones, con probabilidad p_m *por mem*

3.4. Para cada nuevo cromosoma H, hacer:

3.4.1. $H' = BL(H, I_l)$

3.4.2. Evaluar H'

3.4.3. Asignar Fitness(H)= Fitness(H')

3.4.4. Si (R= Estrategia Lamarckiana), entonces

3.4.5. Asignar H= H'

3.5. Reemplazamiento

3.6. Estretegia elitista, si es necesario

4. s= mejor solución encontrada

5. Devolver s

Algoritmo 3.2. Procedimiento de hibridación genético con esquema estacionaria

Los resultados de aplicar la estrategia baldwiniana sólo modifican el valor de adecuación de las soluciones. Por tanto, el efecto de esta estrategia tiene lugar únicamente en la selección de individuos en futuras iteraciones del algoritmo y, por tanto, guiando la búsqueda hacia zonas prometedoras del espacio de soluciones. En cambio, los resultados de aplicar la estrategia lamarckiana también pueden influir sobre el equilibrio entre diversidad y convergencia del

algoritmo evolutivo, debido a que los cromosomas de las soluciones son modificados y trasladados a otras zonas prometedoras del espacio de soluciones.

El algoritmo genético estacionario híbrido ha sido utilizado para entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 2000 soluciones evaluadas
- **Probabilidad de cruce:** 1.0
- **Probabilidad de mutación:** 0.08
- **Factor de elitismo:** Los 2 mejores individuos permanecen en la población
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** número real en el intervalo [-5, 5]
- **Operador de selección:** Torneo binario (TO)/Sorteo (SO)/Equitativo (EQ)/Lineal (LI)
- **Operador de cruce:** BLX-α (BLX)/Heurístico (HEU)/Aritmético (ARI)
- **Operador de mutación:** Desplazamiento (DES)/Aleatorio (ALE)
- **Reemplazamiento:** Padres (PA)/Peores (PE)
- **Búsqueda local:** BFGS
- **Iteraciones de búsqueda local:** 20
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** 8 (Elman), 4 (FRNN) y 11 (Jordan)

Para cada tipo de red se han realizado 30 ejecuciones del algoritmo genético estacionario híbrido con los parámetros anteriores. Las tablas A.19 a A.21 y A.46 a A.48 muestran los resultados de los tests de normalidad y de equivalencia estadística realizados, respectivamente. La tabla 3.1 muestra un resumen de los algoritmos, estadísticamente equivalentes, que mejores resultados han proporcionado.

Tabla 3.1: Mejores combinaciones de operadores para el algoritmo genético estacionario híbrido, en cada tipo de red y subproblema CATS

Problema	**Elman**	**FRNN**	**Jordan**
CATS1	SGABAL/PA/SO/BLX/ALE SGABAL/PE/LI/HEU/DES SGABAL/PA/SO/HEU/ALE SGABAL/PA/LI/HEU/ALE SGABAL/PA/EQ/HEU/ALE SGABAL/PA/TO/BLX/ALE SGABAL/PE/TO/BLX/DES SGABAL/PE/TO/HEU/ALE SGABAL/PA/TO/HEU/DES SGABAL/PA/TO/HEU/ALE SGABAL/PA/LI/HEU/DES SGABAL/PE/LI/BLX/ALE SGABAL/PA/EQ/HEU/DES SGABAL/PE/EQ/BLX/DES SGABAL/PE/LI/HEU/ALE SGABAL/PE/TO/ARI/ALE SGABAL/PE/EQ/HEU/ALE SGABAL/PE/TO/ARI/DES SGABAL/PA/TO/ARI/DES SGABAL/PE/EQ/HEU/DES SGABAL/PE/TO/HEU/DES SGABAL/PA/SO/HEU/DES SGABAL/PA/TO/BLX/DES SGABAL/PE/EQ/BLX/ALE SGABAL/PA/SO/ARI/ALE SGABAL/PA/SO/ARI/DES SGABAL/PA/TO/ARI/ALE	SGABAL/PA/LI/BLX/DES SGABAL/PA/LI/BLX/ALE SGABAL/PE/EQ/BLX/DES SGABAL/PA/TO/HEU/DES SGABAL/PE/LI/BLX/ALE SGABAL/PE/TO/BLX/DES SGABAL/PA/TO/BLX/DES SGABAL/PA/TO/HEU/ALE SGABAL/PA/TO/BLX/ALE SGABAL/PA/EQ/BLX/DES SGABAL/PE/EQ/HEU/DES SGABAL/PA/EQ/HEU/DES SGABAL/PA/SO/BLX/ALE SGABAL/PA/SO/HEU/ALE SGABAL/PE/EQ/BLX/ALE SGABAL/PA/EQ/ARI/ALE SGABAL/PA/SO/ARI/ALE SGABAL/PA/EQ/BLX/ALE SGABAL/PE/TO/HEU/ALE SGABAL/PA/EQ/HEU/ALE SGABAL/PE/TO/BLX/ALE SGABAL/PE/EQ/HEU/ALE	SGABAL/PA/LI/ARI/ALE SGABAL/PE/TO/ARI/ALE SGABAL/PE/TO/ARI/DES SGABAL/PA/EQ/ARI/ALE SGABAL/PA/TO/BLX/DES SGABAL/PA/TO/ARI/DES SGABAL/PE/EQ/ARI/ALE SGABAL/PE/EQ/ARI/DES SGABAL/PE/EQ/BLX/DES SGABAL/PA/TO/ARI/ALE SGABAL/PE/LI/ARI/DES SGABAL/PA/TO/BLX/ALE SGABAL/PE/TO/BLX/ALE SGABAL/PE/TO/BLX/DES SGABAL/PA/EQ/BLX/DES SGABAL/PE/LI/ARI/ALE SGABAL/PE/SO/HEU/DES SGABAL/PE/SO/ARI/ALE SGABAL/PA/EQ/ARI/DES SGABAL/PE/SO/HEU/ALE SGABAL/PE/SO/BLX/DES SGABAL/PA/LI/ARI/DES SGABAL/PE/SO/BLX/ALE SGABAL/PE/SO/ARI/DES SGABAL/PE/LI/HEU/ALE SGABAL/PA/LI/BLX/ALE SGABAL/PE/TO/HEU/DES SGABAL/PA/EQ/BLX/ALE SGABAL/PE/EQ/BLX/ALE SGABAL/PA/LI/BLX/DES
CATS2	SGABAL/PA/SO/ARI/DES SGALAM/PA/SO/ARI/DES SGABAL/PA/SO/ARI/ALE SGABAL/PA/SO/BLX/ALE SGALAM/PA/SO/BLX/DES SGABAL/PA/EQ/ARI/ALE SGABAL/PE/EQ/ARI/ALE SGABAL/PA/SO/HEU/DES SGABAL/PE/TO/ARI/ALE SGABAL/PA/SO/HEU/ALE SGABAL/PE/LI/BLX/DES SGABAL/PE/TO/BLX/DES	SGABAL/PE/EQ/BLX/DES SGABAL/PA/SO/HEU/ALE SGABAL/PA/SO/BLX/ALE SGABAL/PE/LI/BLX/ALE SGALAM/PA/SO/HEU/ALE SGABAL/PA/TO/BLX/DES SGABAL/PE/LI/HEU/ALE SGABAL/PA/LI/BLX/ALE SGABAL/PE/LI/BLX/DES SGABAL/PE/TO/BLX/DES SGALAM/PA/SO/BLX/ALE SGABAL/PA/LI/HEU/ALE SGABAL/PA/SO/BLX/DES SGABAL/PA/LI/BLX/DES SGABAL/PA/EQ/BLX/ALE SGABAL/PE/EQ/BLX/ALE SGABAL/PA/SO/ARI/DES SGABAL/PE/TO/HEU/DES SGABAL/PA/SO/HEU/DES	SGABAL/PE/SO/BLX/ALE SGABAL/PE/TO/BLX/DES SGABAL/PE/LI/BLX/DES SGABAL/PA/EQ/ARI/ALE SGABAL/PA/TO/ARI/ALE SGABAL/PA/EQ/ARI/DES SGABAL/PE/LI/ARI/ALE SGABAL/PE/SO/HEU/DES SGABAL/PE/LI/ARI/DES SGABAL/PE/TO/ARI/DES SGABAL/PE/TO/ARI/ALE SGABAL/PA/LI/ARI/ALE SGABAL/PE/SO/ARI/DES SGABAL/PE/TO/BLX/ALE SGABAL/PE/LI/BLX/ALE SGABAL/PE/SO/BLX/DES SGABAL/PE/EQ/BLX/DES SGABAL/PA/LI/BLX/ALE

CATS3	SGABAL/PA/TO/BLX/DES SGABAL/PA/SO/BLX/DES SGABAL/PA/SO/ARI/DES SGABAL/PA/SO/BLX/ALE SGALAM/PA/SO/BLX/DES SGABAL/PA/SO/HEU/DES SGABAL/PA/SO/ARI/ALE SGABAL/PE/TO/HEU/ALE SGALAM/PA/SO/ARI/DES SGALAM/PA/SO/BLX/ALE SGABAL/PE/TO/BLX/DES SGABAL/PE/LI/BLX/ALE SGABAL/PA/EQ/BLX/ALE SGABAL/PA/SO/HEU/ALE SGABAL/PA/EQ/BLX/DES SGABAL/PA/TO/HEU/DES SGABAL/PA/TO/BLX/ALE SGABAL/PA/EQ/HEU/DES SGABAL/PA/LI/HEU/DES SGABAL/PE/EQ/BLX/ALE SGABAL/PE/LI/BLX/DES SGABAL/PA/LI/BLX/ALE SGALAM/PA/SO/HEU/DES SGALAM/PA/SO/HEU/ALE SGABAL/PA/LI/HEU/ALE SGABAL/PE/TO/BLX/ALE SGABAL/PE/LI/ARI/ALE SGABAL/PA/TO/HEU/ALE SGABAL/PA/TO/ARI/ALE SGALAM/PA/SO/ARI/ALE SGABAL/PE/EQ/ARI/ALE SGABAL/PE/EQ/HEU/DES SGABAL/PE/LI/HEU/DES SGABAL/PE/EQ/BLX/DES SGABAL/PA/EQ/ARI/DES SGABAL/PE/LI/ARI/DES SGABAL/PA/LI/ARI/ALE SGABAL/PA/EQ/ARI/ALE SGABAL/PA/LI/BLX/DES SGABAL/PA/TO/ARI/DES SGABAL/PE/EQ/ARI/DES	SGABAL/PE/LI/HEU/ALE SGABAL/PA/LI/BLX/DES SGABAL/PE/TO/BLX/DES SGABAL/PA/LI/HEU/ALE SGABAL/PE/EQ/BLX/ALE SGABAL/PE/TO/HEU/ALE SGABAL/PA/TO/BLX/DES SGABAL/PE/TO/HEU/DES SGABAL/PE/EQ/BLX/DES SGABAL/PE/TO/BLX/ALE SGABAL/PA/EQ/BLX/ALE SGABAL/PA/LI/HEU/DES SGABAL/PA/TO/HEU/ALE SGABAL/PA/TO/BLX/ALE SGABAL/PE/EQ/HEU/DES SGABAL/PA/LI/BLX/ALE SGABAL/PA/SO/ARI/ALE SGABAL/PA/EQ/BLX/DES SGABAL/PA/TO/HEU/DES SGABAL/PA/SO/HEU/ALE SGABAL/PA/EQ/HEU/DES SGABAL/PA/SO/HEU/DES SGABAL/PE/LI/BLX/DES SGABAL/PA/SO/BLX/ALE SGABAL/PE/TO/ARI/DES SGABAL/PA/TO/ARI/ALE SGABAL/PA/EQ/HEU/ALE SGABAL/PA/TO/ARI/DES SGABAL/PE/LI/BLX/ALE SGABAL/PE/TO/ARI/ALE SGABAL/PA/LI/ARI/DES SGABAL/PA/EQ/ARI/DES SGABAL/PA/SO/BLX/DES SGABAL/PA/LI/ARI/ALE SGABAL/PE/LI/ARI/DES SGALAM/PE/TO/BLX/ALE SGALAM/PE/TO/BLX/DES	SGABAL/PA/LI/ARI/ALE SGABAL/PA/EQ/ARI/ALE SGABAL/PA/TO/ARI/DES SGABAL/PA/TO/ARI/ALE SGABAL/PE/TO/ARI/DES SGABAL/PE/LI/ARI/DES SGABAL/PA/LI/ARI/DES SGABAL/PA/EQ/ARI/DES SGABAL/PA/EQ/HEU/DES SGABAL/PE/TO/BLX/ALE SGABAL/PE/EQ/ARI/ALE SGABAL/PE/LI/HEU/DES SGABAL/PA/LI/BLX/DES SGABAL/PE/SO/HEU/ALE SGABAL/PE/EQ/ARI/DES SGABAL/PE/TO/ARI/ALE SGABAL/PE/SO/HEU/DES SGABAL/PE/SO/ARI/ALE SGABAL/PE/SO/BLX/DES

CATS4	SGABAL/PA/SO/HEU/ALE SGABAL/PA/TO/HEU/DES SGABAL/PA/SO/BLX/ALE SGALAM/PA/SO/BLX/ALE SGABAL/PA/SO/ARI/ALE SGABAL/PE/LI/HEU/ALE	<todos con Baldwiniana>	SGABAL/PA/SO/BLX/DES SGALAM/PA/SO/BLX/DES SGABAL/PA/SO/ARI/ALE SGABAL/PA/SO/BLX/ALE SGALAM/PA/SO/HEU/DES SGABAL/PA/SO/ARI/DES SGABAL/PA/SO/HEU/DES SGALAM/PA/SO/ARI/DES SGALAM/PA/SO/BLX/ALE SGABAL/PA/SO/HEU/ALE SGABAL/PA/TO/HEU/ALE SGALAM/PA/SO/ARI/ALE SGALAM/PA/SO/HEU/ALE SGABAL/PA/TO/HEU/DES SGABAL/PE/EQ/HEU/DES SGABAL/PA/EQ/HEU/DES SGABAL/PE/LI/HEU/ALE SGABAL/PA/EQ/HEU/ALE SGABAL/PE/TO/HEU/ALE SGABAL/PE/TO/HEU/DES SGABAL/PA/LI/HEU/ALE SGABAL/PA/LI/HEU/DES SGABAL/PA/LI/BLX/DES SGALAM/PA/TO/HEU/DES SGABAL/PA/TO/BLX/DES SGABAL/PE/LI/HEU/DES SGABAL/PE/EQ/HEU/ALE SGABAL/PA/EQ/BLX/ALE SGALAM/PA/LI/HEU/DES SGALAM/PA/LI/HEU/ALE SGABAL/PA/LI/BLX/ALE SGABAL/PE/TO/BLX/DES SGALAM/PE/LI/HEU/ALE SGALAM/PA/TO/HEU/ALE SGALAM/PA/EQ/HEU/ALE SGALAM/PA/EQ/HEU/DES SGABAL/PE/LI/BLX/DES SGABAL/PE/TO/BLX/ALE SGABAL/PE/LI/BLX/ALE SGABAL/PE/LI/ARI/ALE SGABAL/PA/LI/ARI/DES SGABAL/PE/TO/ARI/ALE SGABAL/PA/EQ/ARI/ALE SGABAL/PE/EQ/BLX/ALE SGABAL/PA/LI/ARI/ALE SGABAL/PE/EQ/ARI/ALE SGABAL/PA/EQ/ARI/DES SGABAL/PE/EQ/BLX/DES SGABAL/PA/EQ/BLX/DES SGABAL/PE/SO/ARI/ALE SGABAL/PA/TO/BLX/ALE SGABAL/PE/EQ/ARI/DES SGABAL/PE/TO/ARI/DES SGABAL/PE/LI/ARI/DES SGALAM/PE/TO/HEU/DES SGABAL/PA/TO/ARI/DES SGABAL/PA/TO/ARI/ALE SGALAM/PE/LI/HEU/DES SGALAM/PA/TO/BLX/DES SGALAM/PE/TO/HEU/ALE SGALAM/PA/TO/BLX/ALE SGALAM/PE/EQ/HEU/DES SGALAM/PA/LI/BLX/DES SGABAL/PE/SO/HEU/DES

CATS5	<todos con Baldwiniana y reemplazamiento a padres>	<todos con Baldwiniana y reemplazamiento a padres>	SGABAL/PA/TO/BLX/ALE SGABAL/PA/TO/ARI/ALE SGABAL/PA/TO/BLX/DES SGABAL/PA/LI/ARI/DES SGABAL/PA/TO/ARI/DES SGABAL/PE/LI/BLX/DES SGABAL/PA/LI/ARI/ALE SGABAL/PE/LI/BLX/ALE SGABAL/PE/LI/ARI/DES SGABAL/PA/EQ/BLX/DES SGABAL/PA/EQ/ARI/ALE SGABAL/PA/EQ/ARI/DES SGABAL/PE/TO/BLX/ALE SGABAL/PE/EQ/ARI/ALE SGABAL/PA/SO/HEU/ALE SGABAL/PA/LI/BLX/ALE SGABAL/PE/EQ/ARI/DES SGABAL/PE/TO/BLX/DES SGABAL/PE/TO/ARI/DES SGABAL/PA/SO/ARI/ALE SGABAL/PA/EQ/BLX/ALE SGABAL/PE/TO/ARI/ALE SGABAL/PE/LI/ARI/ALE SGABAL/PA/EQ/HEU/ALE SGABAL/PA/SO/HEU/DES SGABAL/PA/EQ/HEU/DES SGABAL/PE/EQ/BLX/DES SGABAL/PA/SO/ARI/DES SGABAL/PE/EQ/BLX/ALE SGABAL/PA/SO/BLX/DES SGABAL/PA/LI/BLX/DES

La tabla 3.1 muestra que la hibridación que proporciona mejores resultados, independientemente de los operadores evolutivos utilizados, es la que utiliza la estrategia baldwiniana. Además, dada la cantidad de algoritmos que proporcionan una solución óptima estadísticamente equivalente, podemos decir que la selección de los operadores evolutivos no es un aspecto clave para entrenar RNRD.

El operador de búsqueda local, por sí solo, consigue dirigir la búsqueda hacia zonas prometedoras del espacio de soluciones. No obstante, una mala combinación de operadores evolutivos puede desembocar en una disminución drástica de la diversidad del algoritmo, evitando así que se puedan explorar diferentes zonas del espacio de soluciones. Esta situación se observa en la Figuras 3.5 a 3.19, donde las soluciones que presentan una disminución de diversidad producen peores soluciones.

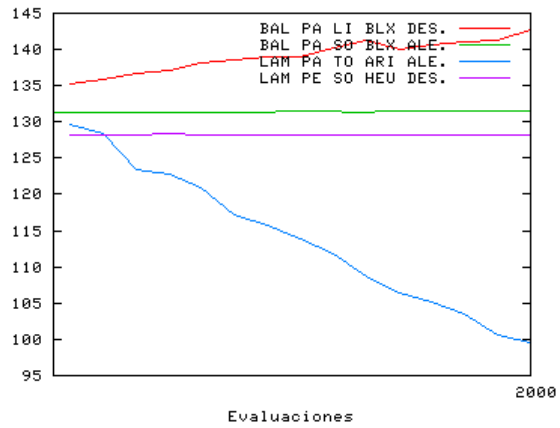

Figura 3.5. Evolución de diversidad en red de Elman, problema CATS1

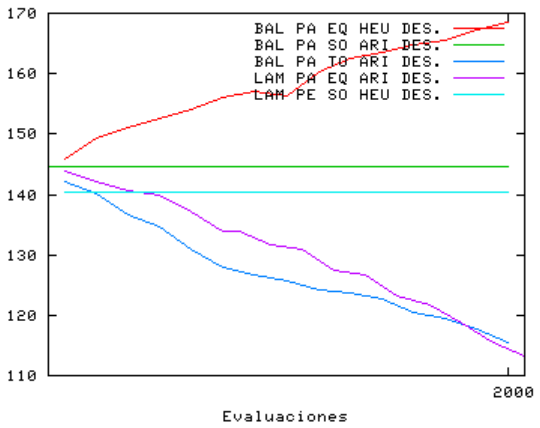

Figura 3.6. Evolución de diversidad en red de Elman, problema CATS2

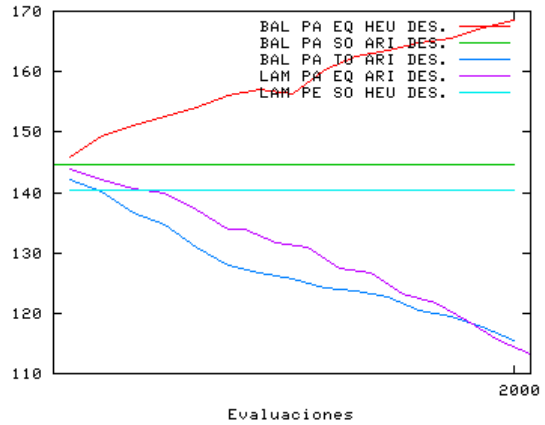

Figura 3.7. Evolución de diversidad en red de Elman, problema CATS3

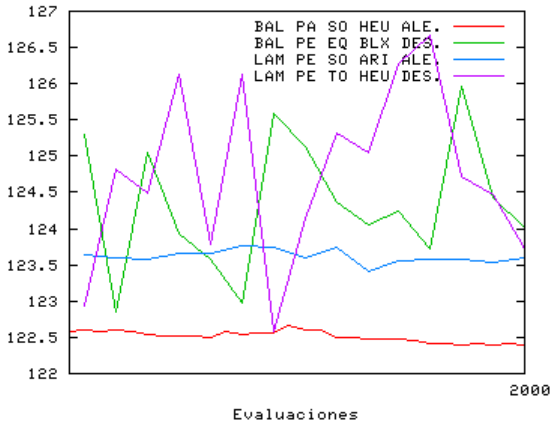

Figura 3.8. Evolución de diversidad en red de Elman, problema CATS4

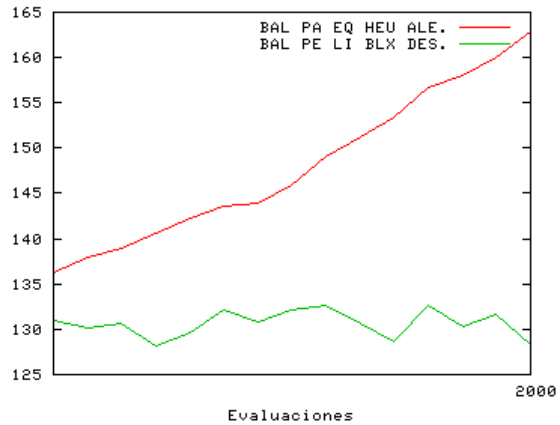

Figura 3.9. Evolución de diversidad en red de Elman, problema CATS5

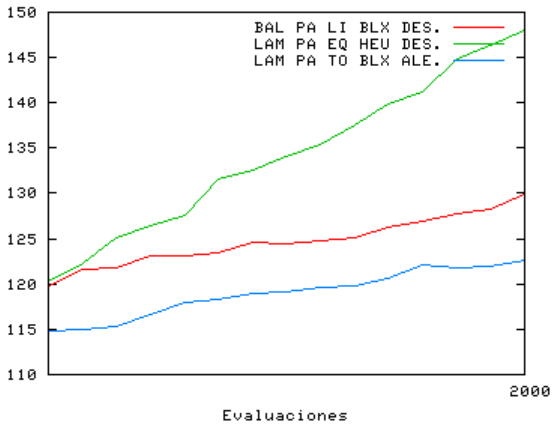

Figura 3.10. Evolución de diversidad en red completa, problema CATS1

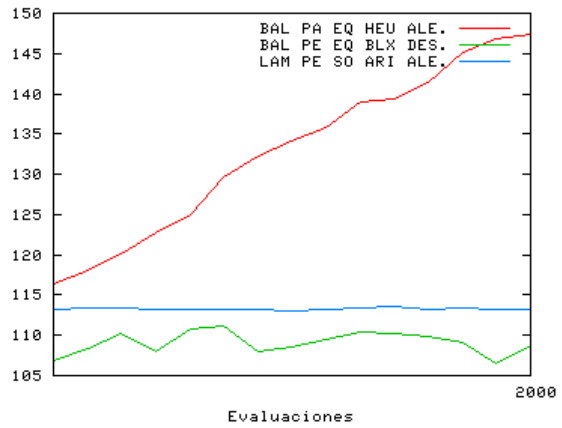

Figura 3.11. Evolución de diversidad en red completa, problema CATS2

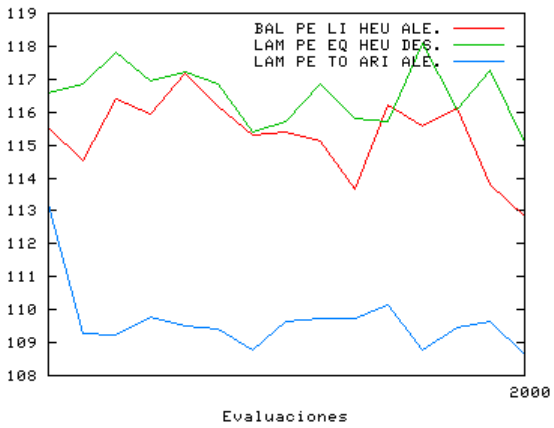

Figura 3.12. Evolución de diversidad en red completa, problema CATS3

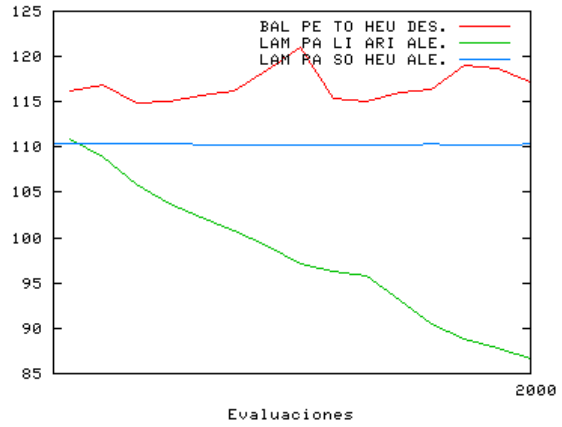

Figura 3.13. Evolución de diversidad en red completa, problema CATS4

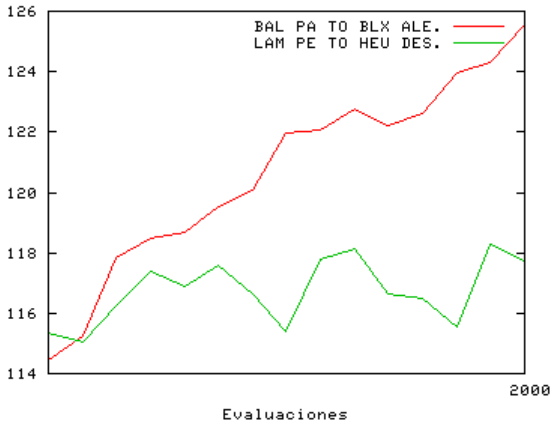

Figura 3.14. Evolución de diversidad en red completa, problema CATS5

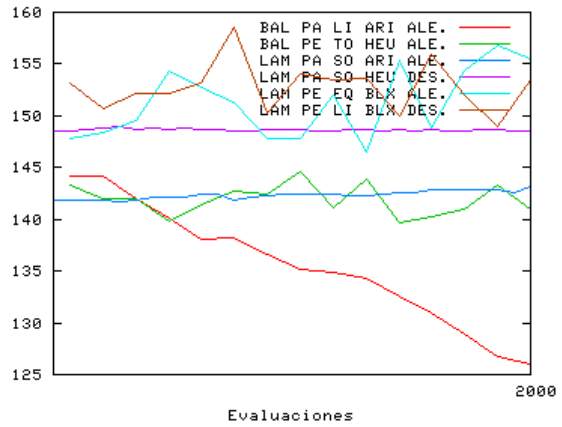

Figura 3.15. Evolución de diversidad en red de Jordan, problema CATS1

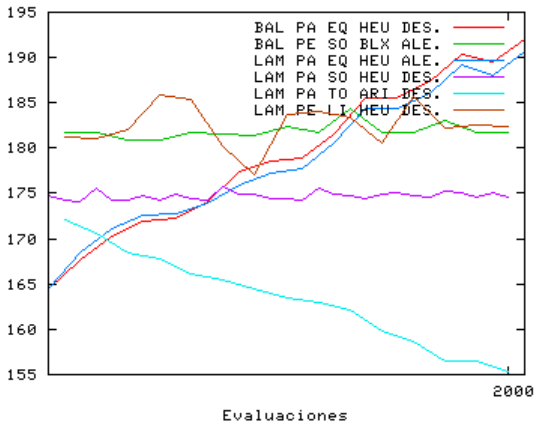

Figura 3.16. Evolución de diversidad en red de Jordan, problema CATS2

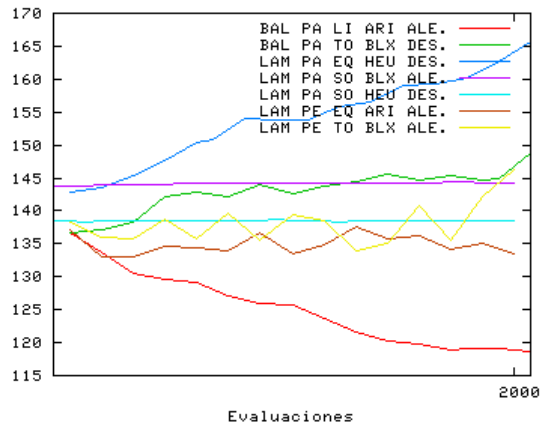

Figura 3.17. Evolución de diversidad en red de Jordan, problema CATS3

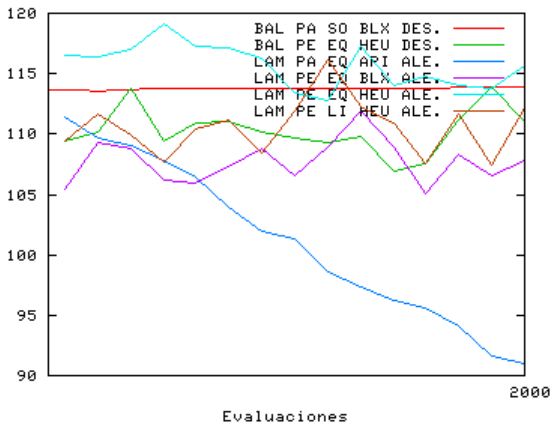

Figura 3.18. Evolución de diversidad en red de Jordan, problema CATS4

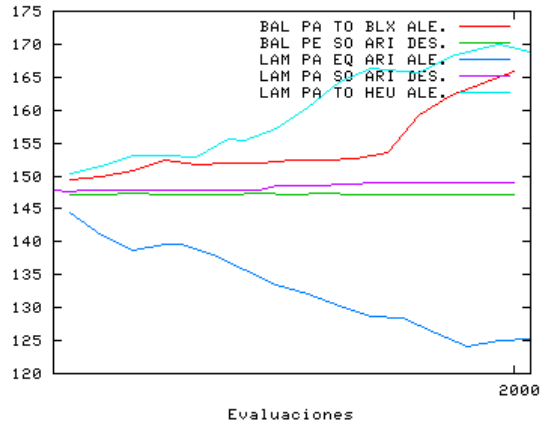

Figura 3.19. Evolución de diversidad en red de Jordan, problema CATS5

4.2. Hibridaciones del algoritmo CHC con métodos de programación no lineal (CHCBFGS)

Las hibridaciones CHCBFGS surgen de combinar el algoritmo CHC, adaptado para codificación real, y el método de optimización BFGS. El funcionamiento del procedimiento híbrido CHCBFGS se ilustra en el Algoritmo 3.3, con la siguiente notación:

- t indica la iteración actual del algoritmo.
- $P(t)$ es una población con N soluciones.
- d es el valor que indica la distancia Euclídea media entre los cromosomas de la población inicial.
- dec indica el decremento a realizar en el valor d cuando no se produzca descendencia, dependiente del parámetro μ ($0<\mu<1$).
- $\{H\}_M$ contiene las nuevas M soluciones generadas mediante el operador de cruce.

El algoritmo comienza inicializando la población $P(t)$ (t=0), con N soluciones generadas aleatoriamente por medio de una distribución uniforme. Seguidamente, los pasos 2 y 3 calculan la distancia media d entre los cromosomas de la población, y el valor de decremento de la misma, dec. El paso 4 evalúa las soluciones de $P(t)$ en el problema a resolver.

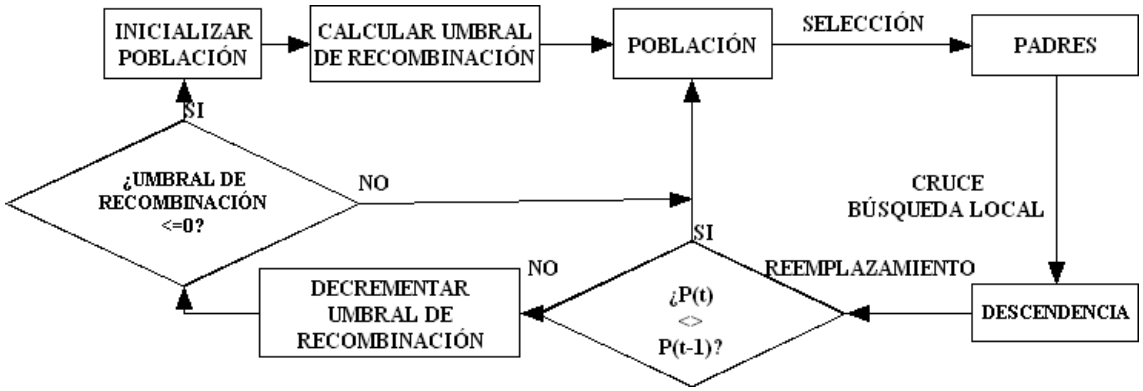

Figura 3.20. Esquema CHC memético

El paso 5 constituye el cuerpo principal del algoritmo CHC. En primer lugar, en el paso 5.1 se realiza una selección de N individuos a partir de la población $P(t)$. A continuación, en el paso 5.2 tiene lugar la recombinación de los individuos seleccionados para generar un total de M ($M \leq N$) descendientes, los cuales son evaluados en el paso 5.3. La recombinación se realiza aplicando una estrategia de prevención de incesto. Si la distancia Euclídea entre dos cromosomas a cruzar es inferior al umbral d, entonces se asume que las soluciones son similares, y no se combinan. En este punto, se llevan a cabo los pasos clave de la hibridación. A cada nueva solución H obtenida mediante el cruce se le aplica el operador de búsqueda local BFGS un total de I_1 iteraciones (paso 5.4.1), generando una nueva solución H'. Seguidamente, los pasos 5.4.2 y 5.4.3 evaluan las soluciones mejoradas tras el paso 5.4.1, y modifican el valor

de adecuación de cada solución *H* con el valor de adecuación de *H'* (sólo estrategia baldwiniana). Si se desea aplicar la estrategia lamarckiana entonces el paso 5.4.5 modifica el cromosoma de la solución *H*, reemplazándolo por el cromosoma de la solución *H'*.

Al introducir la estrategia de prevención de incesto, es posible que no se produzca ninguna recombinación entre los individuos de la población. En este caso, la población tiende a converger hacia una zona concreta dentro del espacio de soluciones. Para explotar adecuadamente dicha zona, el valor *d* decrementa en el paso 5.5. Llegado el caso en que la zona del espacio se haya explotado lo suficiente, y siendo el umbral *d* menor o igual que cero, la población *P(t)* es reinicializada (paso 5.6) para poder explorar nuevas zonas del espacio de búsqueda. Tras la reinicialización, *P(t)* contendrá la mejor solución encontrada hasta el momento, y *N-1* nuevas soluciones generadas aleatoriamente.

Los valores *d* y *dec* deben ser recalculados (paso 5.5.4), para volver a aplicar el ciclo evolutivo hasta que se cumpla una cierta condición de parada, que puede ser llegar a un número dado de iteraciones del algoritmo, haber evaluado un número dado de individuos, llegar a obtener una solución suficientemente adecuada, etc. Cuando finaliza el proceso evolutivo, el algoritmo devuelve la mejor solución encontrada.

El algoritmo CHC híbrido ha sido utilizado para entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 2000 soluciones evaluadas
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** número real en el intervalo [-5, 5]
- **Operador de selección:** Torneo binario
- **Operador de cruce:** BLX-α (BLX)/Heurístico (HEU)/Aritmético (ARI)
- **Búsqueda local:** BFGS
- **Iteraciones de búsqueda local:** 20
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)

- **Neuronas ocultas:** 8 (Elman), 4 (FRNN) y 11 (Jordan)

1. *t=0, Inicializar P(t) con N soluciones*
2. *d= Distancia media entre los cromosomas en P(t)*
3. *Asignar dec= $\mu \cdot d$*
4. *Evaluar soluciones en P(t)*
5. *Mientras (condición de parada = FALSO),*
   - *5.1. P'(t)= selección de N padres a partir de P(t)*
   - *5.3. $\{H\}_M$ = recombinación de padres en P'(t)*
   - *5.4.. Para cada nuevo cromosoma H, hacer:*
     - *5.4.1. H'= BL(H, I_l)*
     - *5.4.2. Evaluar H'*
     - *5.4.3. Asignar Fitness(H)= Fitness(H')*
     - *5.4.4. SI (R= Estrategia Lamarckiana), entonces Asignar H= H'*
   - *5.5. Si $|\{H\}_M| = 0$, entonces*
     - *5.5.1. Asignar d= d-dec*
   - *5.6. Si $d \leq 0$*
     - *5.6.1. Asignar s= Mejor solución en P(t)*
     - *5.6.2. Reinicializar población P(t) con s y N-1 nuevas soluciones*
     - *5.5.3. Evaluar soluciones en P(t)*
     - *5.5.4.Recalcular valores d y dec*
6. *Devolver s*

Algoritmo 3.3. Procedimiento de hibridación con esquema base CHC

Se han realizado 30 ejecuciones del algoritmo CHC híbrido con los parámetros anteriores, para cada tipo de red. Las tablas A.22 a A.24 y A49 a A.51 muestran los resultados de los tests de normalidad y de equivalencia estadística realizados sobre las diferentes combinaciones de operadores evolutivos y técnicas de hibridación estudiados. No obstante, la tabla 3.2 muestra un resumen de los algoritmos, estadísticamente equivalentes, que mejores resultados han proporcionado.

Tabla 3.2: Mejores combinaciones para el algoritmo CHC híbrido, en cada tipo de red y subproblema CATS

Problema	**Elman**	**FRNN**	**Jordan**
CATS1	CHC-BAL	CHC-BAL	CHC-BAL
CATS2	CHC-BAL	CHC-BAL	CHC-BAL
CATS3	CHC-BAL	CHC-BAL	CHC-BAL
CATS4	CHC-BAL	CHC-BAL	CHC-BAL
CATS5	CHC-BAL	CHC-BAL	CHC-BAL

La tabla 3.2 muestra que la mejor estrategia de hibridación, para el algoritmo CHC, es la estrategia baldwiniana. La utilización de la estrategia lamarckiana altera considerablemente la diversidad en la población, anulando de esta forma las componentes de *reinicialización* y de *prevención de incesto* del algoritmo.

De este modo el equilibrio entre diversidad y convergencia, implementado mediante la combinación de las diversas componentes del método CHC, se vuelve inestable, produciendo soluciones poco adecuadas. Las Figuras 3.21 a 3.35 corroboran este análisis, mostrando la evolución de la diversidad de las soluciones de los algoritmos CHC híbridos, al entrenar redes neuronales recurrentes de Elman, completas y Jordan, en el benchmark CATS.

Podemos observar que la diversidad es mayor en las gráficas que utlizan el método de hibridación lamarckiano.

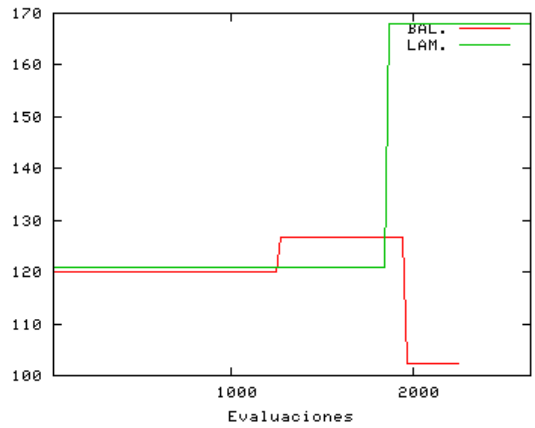

Figura 3.21. Evolución de diversidad en red de Elman, problema CATS1

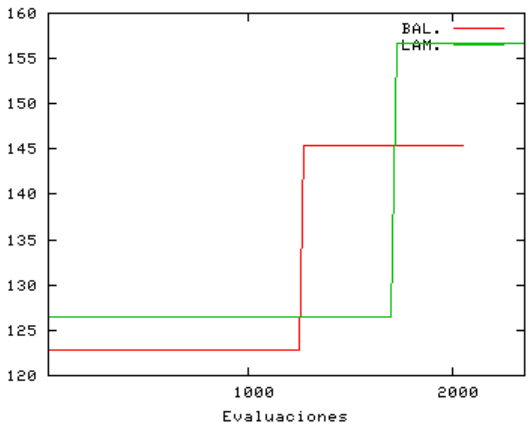

Figura 3.22. Evolución de diversidad en red de Elman, problema CATS2

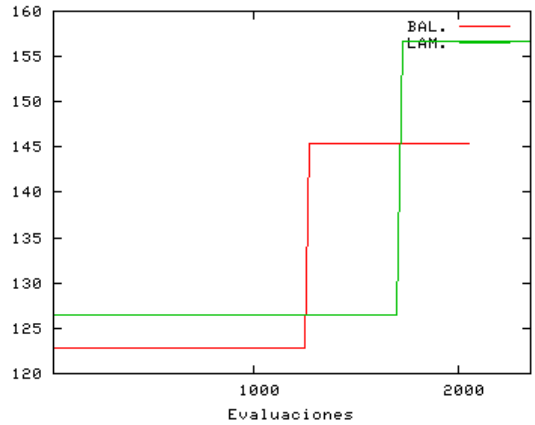

Figura 3.23. Evolución de diversidad en red de Elman, problema CATS3

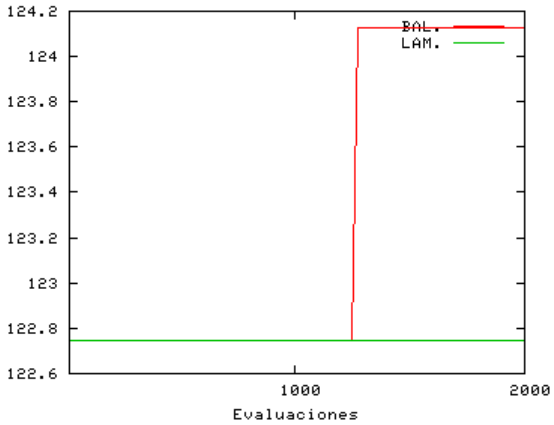

Figura 3.24. Evolución de diversidad en red de Elman, problema CATS4

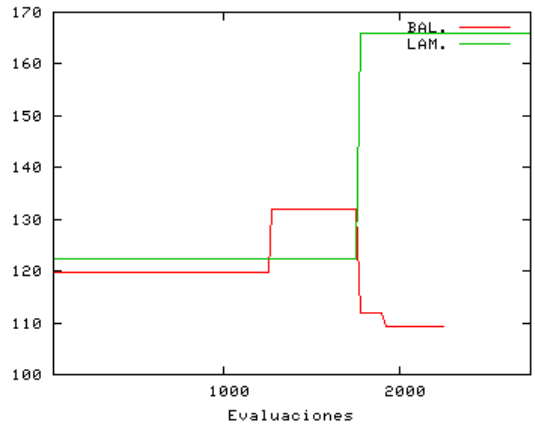

Figura 3.25. Evolución de diversidad en red de Elman, problema CATS5

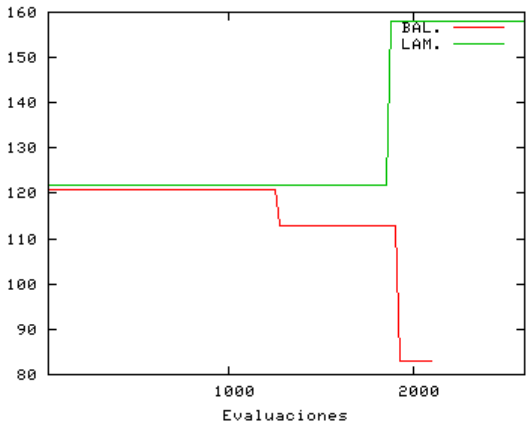

Figura 3.26. Evolución de diversidad en red completa, problema CATS1

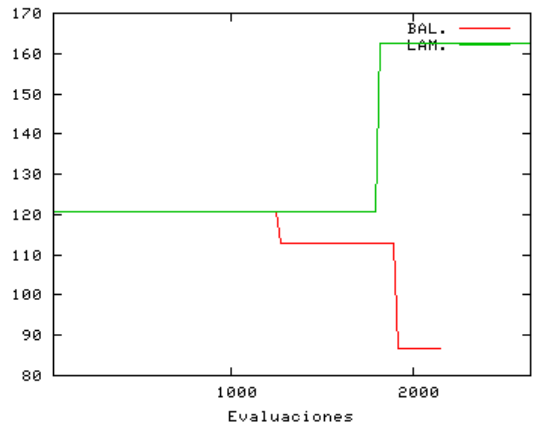

Figura 3.27. Evolución de diversidad en red completa, problema CATS2

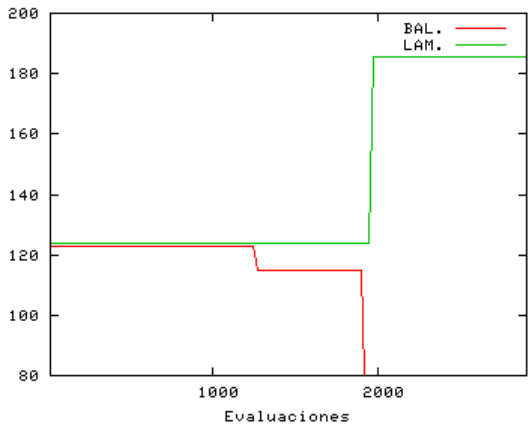

Figura 3.28. Evolución de diversidad en red completa, problema CATS3

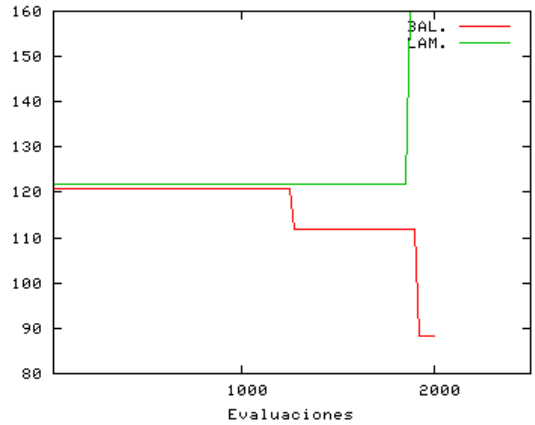

Figura 3.29. Evolución de diversidad en red completa, problema CATS4

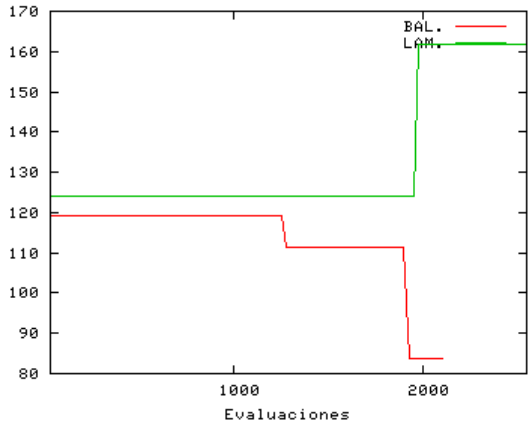

Figura 3.30. Evolución de diversidad en red completa, problema CATS5

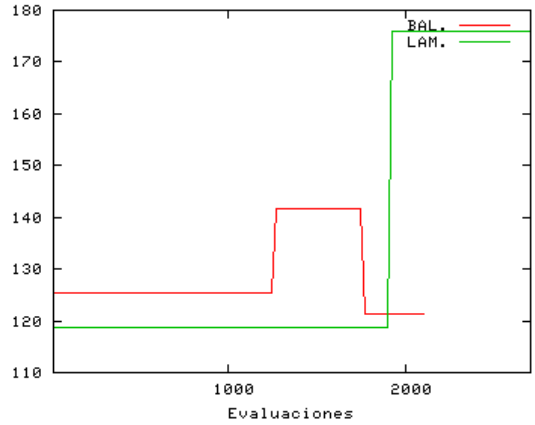

Figura 3.31. Evolución de diversidad en red de Jordan, problema CATS1

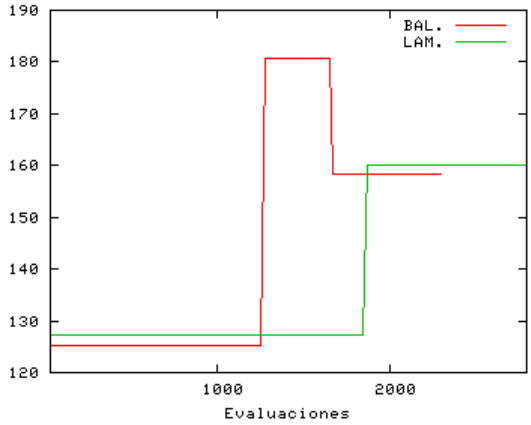

Figura 3.32. Evolución de diversidad en red de Jordan, problema CATS2

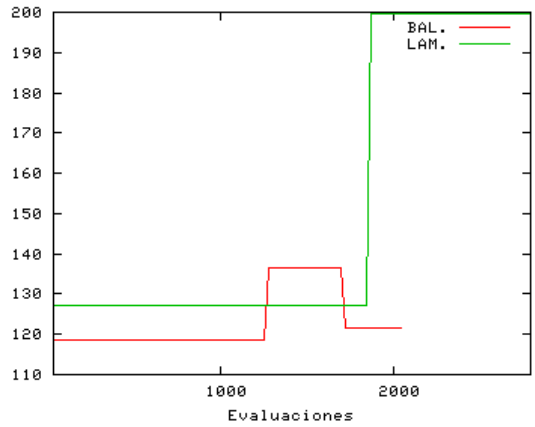

Figura 3.33. Evolución de diversidad en red de Jordan, problema CATS3

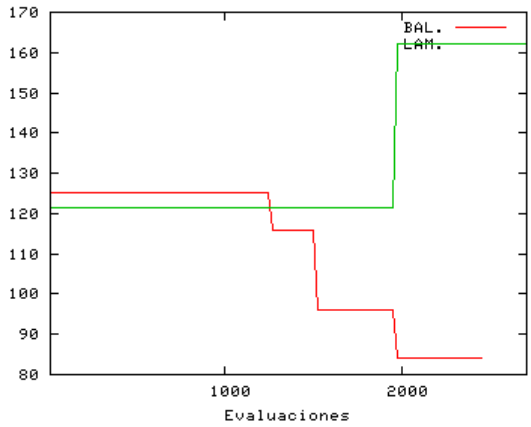

Figura 3.34. Evolución de diversidad en red de Jordan, problema CATS4

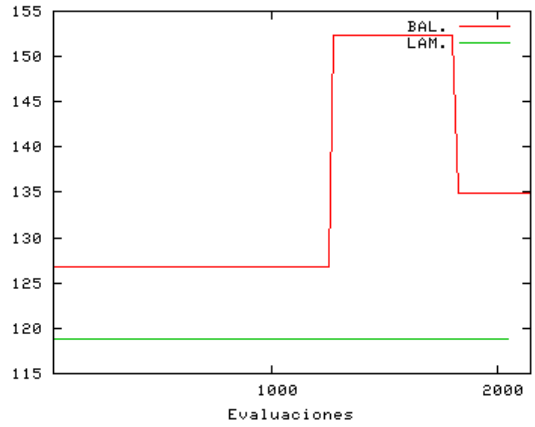

Figura 3.35. Evolución de diversidad en red de Jordan, problema CATS5

5. Otros métodos híbridos.

En este apartado introducimos otros algoritmos híbridos considerados durante nuestra investigación. La sección 5.1 estudia los resultados de la hibridación de algoritmos genéticos con estrategia generacional y mixta, para entrenar RNRD en el problema CATS1. La sección 5.2 introduce el método híbrido de *búsqueda dispersa* (Scatter Search), y los experimentos realizados sobre los problemas considerados.

5.1. Entrenamiento de RNRD mediante algoritmos genéticos híbridos con esquemas evolutivos generacional y mixto.

Durante el desarrollo de nuestra investigación, hemos realizado experimentos con algoritmos genéticos híbridos utilizando estrategias de evolución generacional y mixta. Los parámetros utilizados son similares a los del esquema híbrido SGABFGS, expuesto en el apartado 4 de este capítulo. No obstante, nuestro interés se centra en los parámetros del tamaño de la población, las iteraciones del operador de búsqueda local, y el número de evaluaciones. Estos valores son 50, 25 y 2000, respectivamente.

Los algoritmos híbridos que utilizan los esquemas evolutivos con esquema generacional y mixto han dado resultados estadísticamente equivalentes a una ejecución multiarranque del algoritmo de búsqueda local BFGS. Por este motivo, los hemos separado de los algoritmos híbridos SGABFGS y CHCBFGS de modo que no eclipsen ni dificulten la lectura de los resultados obtenidos por estos últimos. Los motivos de este comportamiento pueden atribuirse al siguiente razonamiento:

La inicialización de las soluciones y su evaluación (junto con el operador de búsqueda local), consumen un total de 50·25= 1250 evaluaciones. Esto deja un total de 750 evaluaciones para el proceso evolutivo. Sin embargo, tanto en el esquema generacional como en el mixto, en cada iteración se evalúa un total de soluciones igual al tamaño de la población. Junto con el operador de búsqueda local, el resultado son 1250 evaluaciones más, superando el límite de las 2000 evaluaciones del criterio de parada.

De este modo, durante una misma ejecución de un algoritmo genético con esquema generacional o mixto, realmente no ha tenido lugar más de una generación evolutiva. Los resultados esperados son, por tanto, estadísticamente equivalentes a la ejecución multiarranque del operador de búsqueda local BFGS, con criterio de parada igual al tiempo de ejecución medio de un algoritmo evolutivo híbrido de los modelos considerados.

La tabla 3.3 muestra la veracidad de la afirmación realizada, mediante la comparación estadística de los modelos genéticos híbridos (generacional y mixto) con el algoritmo multiarranque BFGS, en el entrenamiento de RNRD para el problema CATS1.

Tabla 3.3: Comparación de los modelos evolutivos híbridos con estrategia generacional y mixta, frente al algoritmo multiarranque BFGS, en el problema CATS1

Elman	FRNN	Jordan
BFGS (1) X	BFGS (1) X	BFGS (1) X
MGALAM_PE_LI_HEU_DES (0.05844) X	GGABAL_PA_EQ_BLX_DES (0.3219) X	MGABAL_PE_TO_ARI_ALE 0.0001211 (-)
GGABAL_PA_EQ_ARI_DES (3.496e-08) -	MGALAM_PA_LI_HEU_DES (0.1188) X	GGABAL_PA_EQ_ARI_DES 0.764000 (X)

La tabla 3.3 denota con el símbolo *"X"* a los algoritmos estadísticamente equivalentes al inmediatamente anterior, y con el símbolo "-" a los peores.

5.2. Algoritmo de Búsqueda Dispersa.

El modelo híbrido denominado búsqueda dispersa (*Scatter Search*) fue propuesto en la última mitad de la década de los 90 por Fred Glover [GLO94][GLO00].Fue utilizado por Rafael Martí *et al.* para resolver el problema del entrenamiento de redes neuronales de tipo feedforward, obteniendo resultados prometedores [MAR03][MAR04]. En la actualidad, es un método metaheurístico que está teniendo éxito en la resolución de multitud de problemas, y se encuentra en proceso de mejora y expansión [GLO03][GLO04][MAR03][MAR05].

En nuestra investigación, hemos usado este método para comparar los resultados obtenidos por los métodos desarrollados en el apartado 4. El Algoritmo3.4 resume en detalle el funcionamiento básico del método. Para mayor información sobre el método de búsqueda Dispersa, puede consultarse la bibliografía [GLO94][GLO00].

1. t= 0. Construir P(t) inicial mediante el método de generación de soluciones
2. Construir el conjunto de referencia R, con las b/2 soluciones mejores de P(t) y las b/2 soluciones más diversas
3. Evaluar las soluciones en R
4. Ordenar R de de mejor a peor solución.
5. Asignar NuevaSolución = TRUE
6. Mientras (NuevaSolución)
7. NuevaSolucion = FALSE
8. Generar subconjuntos de R en los que haya al menos una nueva solución.
9. Para cara subconjunto generado, hacer:
10. Recombinar soluciones del subconjunto
11. Aplicar el método de mejora a cada solución obtenida x
12. Si ((fitness (x)< fitness (mejor)) hacer
13. mejor= x
14. reordenar R
15. Hacer NuevaSolucion = TRUE
16. Devolver mejor

Algoritmo 3.4. Procedimiento general del método Scatter Search

6. Consideraciones finales.

En los apartados anteriores hemos estudiado las diferentes propuestas híbridas, y hemos realizado una experimentación para calcular los parámetros óptimos de cada método por separado, para el problema de series temporales benchmark *CATS*. Para terminar este capítulo y a modo de resumen, en este apartado realizamos un estudio estadístico general entre las configuraciones de los métodos híbridos que mejor comportamiento han presentado. En primer lugar, analizaremos estadísticamente el conjunto de los resultados obtenidos en cada modelo de red por separado. Seguidamente, analizaremos todos los resultados en conjunto, para conocer cuál es el modelo de red que ha conseguido aprender un mejor comportamiento.

Las tablas A.40 a A.43 (Anexo I) muestran los tests de equivalencia de los algoritmos meméticos para los diferentes modelos de redes neuronales recurrentes estudiados, junto con los algoritmos genéticos híbridos con esquemas generacional y mixto (sólo problema CATS1), el algoritmo Scatter Search, y un algoritmo multiarranque BFGS. Adicionalmente, la tabla 3.4 muestra un resumen de los mejores algoritmos de entrenamiento, estadísticamente equivalentes, ordenados por red y problema.

Tabla 3.4: Algoritmos híbridos que mejores resultados han proporcionado, alentrenar RNRD en el problema CATS

Problema	**Elman**	**FRNN**	**Jordan**
CATS1	*SGABAL/PA/SO/BLX/ALE* *SS* *CHCBAL* *MGALAM/PE/LI/HEU/DES*	*SGABAL/PA/LI/BLX/DES* *CHCBAL* *MGALAM/PA/LI/HEU/DES*	*BFGS*
CATS2	*SGABAL/PA/SO/ARI/DES* *BFGS*	*SGABAL/PE/EQ/BLX/DES* *CHCBAL* *SS*	*BFGS*
CATS3	*SGABAL/PA/TO/BLX/DES* *SS* *CHCBAL*	*SGABAL/PE/LI/HEU/ALE* *CHCBAL*	*BFGS*
CATS4	*SGABAL/PA/SO/HEU/ALE* *SS* *CHCBAL*	*CHCBAL* *SGABAL/PE/TO/HEU/DES* *SS*	*BFGS*
CATS5	*BFGS* *SGABAL/PE/LI/BLX/DES*	*SGABAL/TO/BLX/ALE* *BFGS* *CHCBAL*	*BFGS*

A partir de los resultados mostrados en la tabla 3.4, podemos extraer las conclusiones siguientes:

- El algoritmo BFGS sin hibridación es el que mejores resultadosha obtenido en las redes de Jordan.
- El esquema evolutivo híbrido que mejores resultados ha proporcionado (independientemente del modelo de red entrenada), ha sido el modelo genético estacionario, hibridado con el algoritmo BFGS con la estrategia baldwiniana.

- Este modelo ha sido el que mejores resultados ha proporcionado, en conjunto, para las redes neuronales recurrentes de Elman y completas.
- Por último, cabe también mencionar los buenos resultados obtenidos porlos algoritmos de entrenamiento híbridos CHC con estrategia baldwiniana y Scatter Search, en algunos problemas.

La tabla 3.5 muestra los resultados del test de equivalencia de los algoritmos de entrenamiento que mejores resultados han obtenido, sin discriminar entre modelos de red.

Tabla 3.5: Test de KruskalWallis de los mejores resultados proporcionados para entrenar redes recurrentes dinámicas, en el problema CATS

Algoritmos	**CATS1**	**Algoritmos**	**CATS2**
SGABAL/PA/LI/BLX/DES FRNN	*1 (x)*	SGABAL/PE/EQ/BLX/DES FRNN	*1 (x)*
SGABAL/PA/SO/BLX/ALE Elman	*0.001406 (-)*	BFGS Jordan	*5.228e-11 (-)*
BFGS Jordan	*0.007129 (-)*	SGABAL/PA/SO/ARI/DES Elman	*0.007129 (-)*

Tabla 3.5: (Continuacion)

Algoritmos	**CATS3**	**Algoritmos**	**CATS4**
SGABAL/PE/LI/HEU/ALE FRNN	*1 (x)*	CHCBAL FRNN	*1 (x)*
SGABAL/PA/TO/BLX/DES Elman	*0.008498 (-)*	SGABAL/PA/SO/HEU/ALE Elman	*2.331e-09 (-)*
BFGS Jordan	*0.0228 (-)*	BFGS Jordan	*0.004128 (-)*

Tabla 3.5: (Continuacion)

Algoritmos	**CATS5**
SGABAL/PA/TO/BLX/ALE FRNN	*1 (x)*
BFGS Jordan	*0.4161 (x)*
BFGS Elman	*0.03089 (-)*

La red que mejor ha conseguido aprender los problemas planteados ha sido la red recurrente completa. Además, el test de equivalencia estadística concluye que hay una gran importancia a la hora de seleccionar el modelo de red a entrenar, ya que los resultados de los mejores algoritmos para entrenar cada red difieren estadísticamente en gran medida.

Capítulo 4

Métodos híbridos para optimización de Redes Neuronales Recurrentes Dinámicas.

1. Introducción.

En este capítulo diseñamos un procedimiento que simultáneamente determina la topología óptima de la red y realiza el entrenamiento de la misma. Para ello, utilizaremos modelos de algoritmos evolutivos multiobjetivo [COE02][COE00][COE99][FON95] [FON98][LU03][NAY99][PAR01]. El objetivo perseguido es doble:

- Reducir el número de experimentos necesarios para encontrar el número de neuronas ocultas de una RNRD, usualmente realizado mediante un método de ensayo y error.
- Generalizar la idea de entrenamiento híbrido expuesta en el capítulo 3 a propuestas de entrenamiento y optimización multiobjetivo.

La idea de utilizar métodos multiobjetivo para obtener una red neuronal con estructura topológica óptima no es nueva. Algunos estudios recientes han abordado este problema obteniendo resultados muy prometedores [LIU98][LIU99][DEL00][GAR01][GAR03][HUS02] [HUS03][PEG05][ZUR01]. Concretamente, el aporte realizado por este capítulo se centra en:

- La propuesta de codificación de RNRD a cromosomas para solucionar los problemas de entrenamiento y optimización de estos modelos de red.

- La propuesta de métodos de cruce y mutación multiobjetivo específicos para los modelos de redes considerados.
- La propuesta de hibridación de algoritmos multiobjetivo con métodos clásicos de entrenamiento, basándonos en las conclusiones obtenidas en el capítulo 3.

Los operadores genéticos clásicos, tanto de cruce como de mutación, actúan sobre cromosomas homogéneos de igual tamaño, para resolver el problema del entrenamiento de una red neuronal recurrente dinámica cuya estructura topológica es predefinida y fija (capítulos 2 y 3). En este capítulo, necesitamos desarrollar una nueva representación que permita codificar las estructura interna de la red junto con los pesos de las conexiones, de modo que podamos utilizarla para entrenamiento y optimización de la misma. Esta representación necesita tener una estructura heterogénea (mezcla de codificación entera para el número de neuronas ocultas, y real para los pesos de las conexiones), y su tamaño puede variar en función del número de neuronas ocultas de la red.

Por estos motivos, resulta necesario el desarrollo de nuevos operadores de cruce y mutación de soluciones, para operar sobre cromosomas que codifiquen RNRD con una estructura del tipo anteriormente comentado. El capítulo está organizado según el esquema siguiente:

El apartado 2 introduce la representación a utilizar, para entrenamiento y optimización de RNRD mediante algoritmos evolutivos multiobjetivo. El apartado 3 explica las funciones objetivo que guiarán el proceso de búsqueda en los algoritmos, para resolver el problema planteado. Seguidamente, los apartados 4 y 5 detallan los operadores de recombinación y mutación multiobjetivo propuestos, respectivamente. Estos operadores serán utilizados por los algoritmos expuestos en los apartados 6 y 7 para entrenar y optimizar RNRD, utilizando los métodos SPEA-II y NSGA-II clásicos y nuevas propuestas híbridas. Por último, el apartado 8 muestra los resultados experimentales obtenidos al comparar los métodos propuestos entre sí.

2. Representación de RNRD para problemas multiobjetivo.

En el capítulo 2 ya hemos discutido la representación de los distintos modelos de RNRD, considerando únicamente el problema del entrenamiento de las mismas. Partiendo de tal representación, este apartado presenta una mejora para permitir, además, la optimización topológica de los modelos de redes recurrentes estudiados.

Considerando los modelos de red de Elman, Jordan y completamente recurrente, podemos diferenciar los tipos de neuronas cuyo número es fijo o es susceptible de ser optimizado. En los tres tipos de red, tanto el número de neuronas de entrada como el de salida son parámetros que vienen definidos y fijos por el problema que se trata de resolver. Así, la optimización de la topología de estos modelos consiste en encontrar el número óptimo de neuronas ocultas para un adecuado aprendizaje de la red. La representación de RNRD desarrollada en este apartado debe considerar, por tanto, el número de neuronas ocultas de la red como parámetro de optimización de la estructura topológica.

Considerando el planteamiento anterior, un cromosoma que represente una red neuronal recurrente dinámica, para su entrenamiento y optimización, tendrá una parte que utilice codificación entera (números de neuronas ocultas de la red), y otra parte que utilice codificación real (valor del peso de las conexiones). A continuación, se expone la representación utilizada para cada modelo de red recurrente dinámica estudiado.

Representación del modelo de Elman

Como acabamos de comentar, la nueva codificación está formada por un gen de codificación entera, cuyo valor es el número de neuronas de la capa oculta, seguido de los genes que representan a las conexiones de la red. La figura 4.1 muestra la estructura del cromosoma resultante.

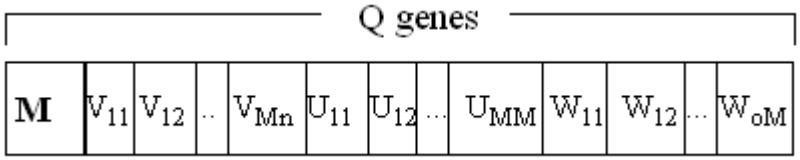

Figura 4.1. Esquema de codificación multiobjetivo general para redes recurrentes de Elman

En la figura 4.1, el valor *M* representa el número de neuronas ocultas de la capa oculta de la red de Elman. El resto de la estructura del cromosoma puede obtenerse a partir de la introducida en el capítulo 2.

Podemos observar que un cromosoma con la configuración anterior puede tener tamaños diferentes, dependiendo del valor del primer gen *M*. El número de genes *Q* del cromosoma viene dado por la siguiente expresión:

$$Q = n \cdot M + M \cdot M + o \cdot M + 1 \tag{4.1}$$

La figura 4.2 muestra un ejemplo de la codificación de una redrecurrente de Elman con una entrada, una salida y dos neuronas ocultas, con un total de 9 genes.

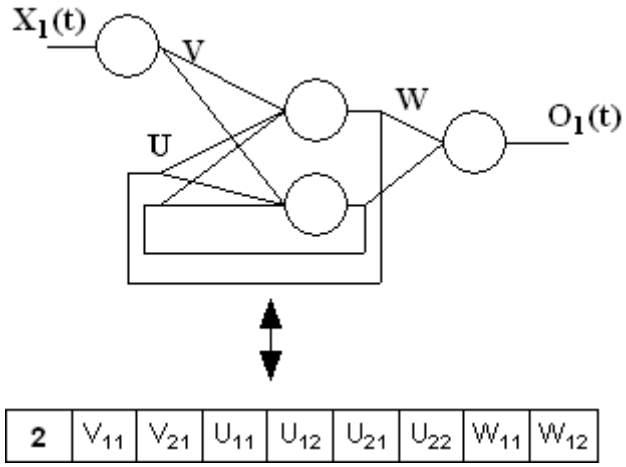

Figura 4.2. Ejemplo de codificación multiobjetivo de una red de Elman con una entrada, una salida y dos neuronas ocultas

Dado un cromosoma $C=(c_1, c_2, \ldots, c_Q)$, el algoritmo 4.1 muestra el esquema de asignación de variables a optimizar a cada gen.

1. K= 1

2. Asignar valor M al gen c_K; Asignar K= K+1

3. Para cada neurona de entrada i=1..n
3.1. para cada neurona oculta j=1..M
Asignar peso V_{ij} al gen c_K; Asignar K=K+1

4. Para cada neurona oculta i=1..M
4.1. para cada neurona oculta j=1..M
Asignar peso U_{ij} al gen c_K; Asignar K= K+1

5. Para cada neurona de salida i=1..o
5.1. para cada neurona oculta j=1..M
Asignar peso W_{ij} al gen c_K; Asignar K= K+1

Algoritmo 4.1. Procedimiento de asignación genética de una red de Elman,para problemas multiobjetivo

Representación del modelo recurrente completo

Con las mismas ideas de base del apartado anterior, la figura 4.3muestra la estructura del cromosoma que codifica una red FRNN ampliado para incluir, además,el número de neuronas ocultas de la red.

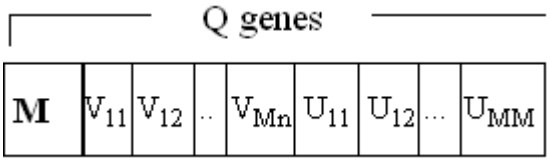

Figura 4.3. Esquema de codificación multiobjetivo general para redes recurrentes completas

En la figura 4.3, el gen que contiene el valor *M* representa el número de neuronas ocultas de la red recurrente completa. El resto de la estructura delcromosoma es idéntica a la utilizada en el capítulo 2.

El número de genes del cromosoma que codifica una red neuronal recurrente completa, en el problema multiobjetivo planteado, se obtiene a partir de la siguiente expresión:

$$Q = n \cdot M + M \cdot M + 1 \tag{4.2}$$

La figura 4.4 muestra un ejemplo de la codificación de una red recurrente completa con una entrada, una salida y dos neuronas ocultas, con un total de 7 genes. El algoritmo 4.2 muestra el esquema de asignación de variables a cada gen en un cromosoma $C=(c_1, c_2, \ldots, c_Q)$.

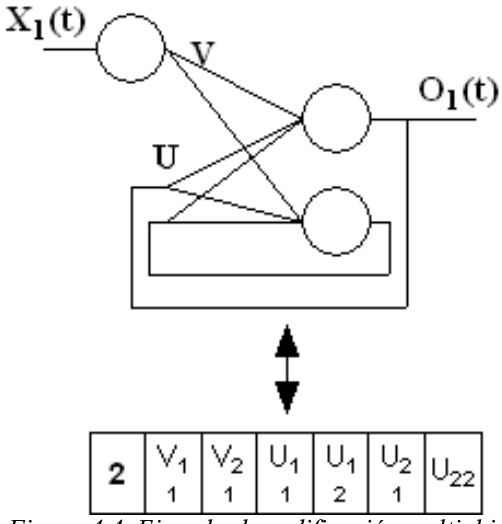

Figura 4.4. Ejemplo de codificación multiobjetivo de una red recurrente completa con una entrada, una salida y dos neuronas ocultas

1. *K= 1*
2. *Asignar valor M al gen c_K; Asignar K= K+1*
3. *Para cada neurona de entrada i=1..n*
   - *3.1. para cada neurona oculta j=1..M*
     - *Asignar peso V_{ij} al gen c_K; Asignar K=K+1*
4. *Para cada neurona oculta i=1..M*
   - *4.1. para cada neurona oculta j=1..M*
     - *Asignar peso U_{ij} al gen c_K; Asignar K= K+1*

Algoritmo 4.2. Procedimiento de asignación genética de una red recurrente completa para problemas multiobjetivo

Representación del modelo de Jordan

Al igual que ocurre en los modelos anteriores, la representación delmodelo recurrente de Jordan (para el problema de optimización y entrenamiento) es como una extensión de la representación estudiada en el capítulo 2. La figura 4.5 muestra la estructura del cromosoma resultante.

El valor M representa el número de neuronas ocultas de la capa oculta de la red recurrente de Jordan. El resto de la estructura del cromosoma puede obtenerse a partir de la codificación estudiada en el capítulo 2.

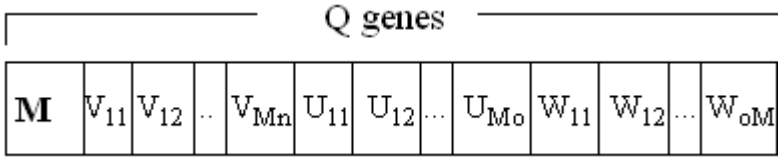

Figura 4.5. Esquema de codificación multiobjetivo general para redes recurrentes de Jordan

El número de genes del cromosoma que codifica una red neuronal recurrente completa, en el problema multiobjetivo planteado, se obtiene a partir de la siguiente expresión:

$$Q = n \cdot M + 2 \cdot o \cdot M + 1 \tag{4.3}$$

La figura 4.6 muestra un ejemplo de la codificación de una red recurrente de Jordan con una entrada, una salida y dos neuronas ocultas, con un total de 7 genes. El Algoritmo 4.3 muestra el esquema de asignación de variables a optimizar a cada gen en un cromosoma $C=(c_1, c_2, \ldots, c_Q)$.

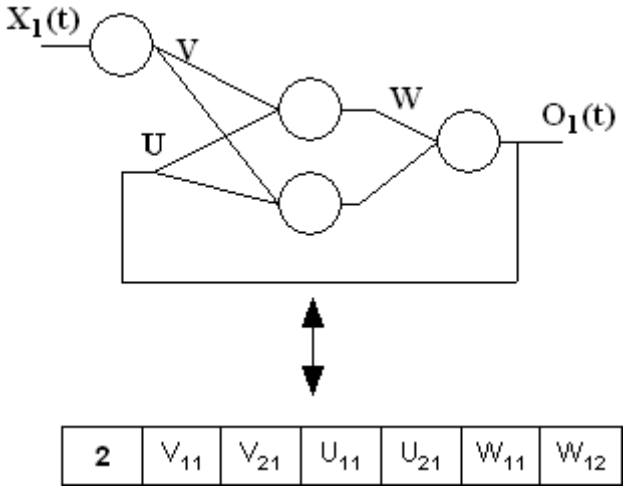

Figura 4.6. Ejemplo de codificación multiobjetivo de una red de Jordan con una entrada, una salida y dos neuronas ocultas

1. K= 1

2. Asignar valor M al gen c_K; Asignar K= K+1

3. Para cada neurona de entrada i=1..n

3.1. para cada neurona oculta j=1..M

Asignar peso V_{ij} al gen c_K; Asignar K=K+1

4. Para cada neurona oculta i=1..M

4.1. para cada neurona de salida j=1..o

Asignar peso U_{ij} al gen c_K; Asignar K= K+1

5. Para cada neurona de salida i=1..o

5.1. para cada neurona oculta j=1..M

Asignar peso W_{ij} al gen c_K; Asignar K= K+1

Algoritmo 4.2. Procedimiento de asignación genética de una red recurrente de Jordan para problemas multiobjetivo

3. Las funciones objetivo.

Una vez fijada la forma de codificación de las redes mediante cromosomas, hay que establecer los criterios a optimizar por los algoritmos multiobjetivo, con el fin de:

a) entrenar una red neuronal recurrente dinámica.

b) Determinar su topología óptima.

Con respecto al criterio para resolver a), consideraremos de nuevo el empleado en los capítulos 2 y 3. Trataremos de minimizar el error cuadrático medio producido entre las salidas de la red en un instante *t*, y el valor de la serie temporal en el instante *t+1*.

Por otra parte, para resolver el problema planteado en b) partiremos de las siguientes observaciones, ampliamente contrastadas:

- Una red neuronal con un número pequeño de neuronas ocultas puede no aprender correctamente los datos deseados.
- Una red neuronal con un número de neuronas ocultas excesivo puede sobreaprender los datos de entrenamiento, produciendo resultados poco adecuados al aplicarla sobre el conjunto de test.

Teniendo en cuenta ambas observaciones, el objetivo es encontrar la red neuronal que proporcione resultados adecuados, con el mínimo número de neuronas ocultas posible.

De este modo, podemos plantear los criterios objetivo para el problema de la optimización y entrenamiento de una red neuronal recurrente dinámica del siguiente modo:

$$f_1(C^*) = Min\, f_1(C) = Min\{\sum_{i=1}^{T} (O(t) - Y(t+1))^2\} \tag{4.4}$$

$$f_2(C^*) = Min\, f_2(C) = Min\{M\} \tag{4.5}$$

De acuerdo con la notación empleada hasta el momento, C^* representa una solución óptima, $O(t)$ es la salida de la red en el instante t, $Y(t+1)$ es el valor de la serie temporal en el instante $t+1$, T es el número de patrones de entrenamiento utilizados, y M es el número de neuronas ocultas de la red neuronal.

Los algoritmos multiobjetivo planteados tratarán de encontrar las soluciones que minimicen las funciones objetivo f_1 y f_2, produciendo como resultado un conjunto de soluciones optimales, según el criterio de optimalidad de Pareto. Finalmente, el usuario deberá escoger la solución que considere más adecuada entre las presentadas, para solucionar el problema considerado.

4. El operador de cruce.

Tradicionalmente, podemos clasificar los diferentes operadores de cruce evolutivos según el número de descendientes que se generan tras la recombinación:

- Recombinación de dos o más soluciones para generar un único descendiente.
- Recombinación de dos o más soluciones para generar dos o más descendientes.

Para desarrollar el operador de cruce multiobjetivo expuesto en este capítulo, hemos analizado las opciones anteriores para escoger la más efectiva. En el apartado 3.1 mostramos las conclusiones obtenidas tras la experimentación utilizando métodos de cruce que generan un único individuo. Seguidamente, el apartado 3.2 muestra el operador de cruce desarrollado para generar dos descendientes a partir de dos padres.

4.1. Operadores de cruce que generan un único descendiente.

Los operadores de cruce desarrollados para optimización y entrenamiento simultáneos de RNRD, discriminan entre la sección del cromosoma que codifica la estructura topológica de la red y la sección que representa los pesos de las conexiones entre neuronas. En primer lugar, este apartado estudiará el problema del cruce del gen que codifica al número de neuronas ocultas de la red.

Dados dos cromosomas C_1 y C_2 de tamaños respectivos *Q1* y *Q2*, que codifican dos RNRD del mismo modelo (Elman, Jordan o completa), hemos considerado los siguientes métodos para la recombinación de la estructura topológica representada por $C_1=(c^1_1,c^1_2,...,c^1_{Q1})$ y $C_2=(c^2_1,c^2_2,...,c^2_{Q2})$, y generar un único descendiente *H*:

1. El descendiente *H* tendrá un número de neuronas ocultas igual al número de neuronas ocultas de alguno de los padres, seleccionado aleatoriamente.
2. El descendiente *H* tendrá un número de neuronas ocultas igual al número medio de neuronas ocultas de los padres.
3. El descendiente *H* tendrá un número de neuronas ocultas aleatorio, escogido en el intervalo $[\min\{c^1_1,c^2_1\}, \max\{c^1_1,c^2_1\}]$.

Dependiendo de la estrategia de recombinación seleccionada, la diversidad de la población de redes puede disminuir, provocando así una convergencia prematura [BOS03][THI02][TOF03]. Los tres modelos de recombinación anteriores, y en especial el segundo, tienen una gran tendencia a disminuir la diversidad de las soluciones (considerando únicamente el número de neuronas ocultas de las redes representadas en la población). Cuando el número de iteraciones del algoritmo genético es elevado, el número de soluciones con menor número de neuronas ocultas tiende a aumentar.

Esta situación no es deseable, ya que evita la exploración del espacio de búsqueda (redes con un mayor número de neuronas ocultas). Para paliar el efecto del sesgo producido por el operador de cruce, consideramos dos opciones:

1. Aumentar la probabilidad de mutación de una solución, para favorecer la exploración del espacio de búsqueda.
2. Introducir en el algoritmo un mecanismo de reinicialización, de modo que la población sea regenerada de nuevo, cuando al menos un porcentaje prefijado de las soluciones que la forman tengan igual número de neuronas ocultas.

Los resultados de los experimentos realizados, utilizando la primera opción, han dado lugar a una gran diversidad de soluciones con número de neuronas ocultas diferente, pero con un

factor de convergencia pobre con respecto al entrenamiento. No obstante, la segunda opción ha producido resultados adecuados en cuanto a diversidad de soluciones y convergencia, siendo estos comparables a los resultados obtenidos mediante el operador de cruce que estudiamos en el apartado siguiente.

El segundo factor a considerar por el operador de cruce multiobjetivo es la forma en que se recombinan los genes correspondientes a los pesos de cada red progenitora. Dado que dos cromosomas C_1 y C_2 pueden tener diferente tamaño, el cruce de los pesos de la red debe hacerse siguiendo una estructura adecuada. Estudiaremos este modelo de cruce partiendo de la estructura de recombinación estudiada en los capítulos 2 y 3:

En los capítulos anteriores hemos utilizado operadores de cruce básicos de codificación real, para recombinar dos cromosomas A y B que codifican a RNRD, con estructura interna predefinida y fija. En estos casos, el cruce se realiza gen a gen, ya que cada componente a_i y b_i representan una misma conexión, aunque en redes diferentes. Sin embargo, considerando dos cromosomas C_1 y C_2 que codifican dos redes con la estructura expuesta en este capítulo (y cuyo tamaño puede ser diferente), los genes asociados a una posición dada *i* (c^1_i y c^2_i) pueden no corresponderse a la misma conexión en las dos redes codificadas. Ilustraremos esta idea mediante un ejemplo.

Ejemplo 1:

Correspondencia entre conexiones de red y cromosomas.

Sean C_1 y C_2 dos cromosomas que representan a dos redes de Elman con una entrada, una salida, y dos y tres neuronas ocultas, respectivamente.

$$C_1 = (2, 1.2, 3.7, 0.4, 7.6, 6.6, 3.3, 2.3, 1.1)$$

$$C_2 = (3, 1.1, 2.2, 3.3, 0.1, 0.2, 0.3, 2.2, 3.3, 1.7, 6.7, 7.6, 6.5, 8.7, 0.3, 02)$$

En ambos casos, el gen c^1_2 y c^2_2 se corresponde con la conexión V_{11} de la red. Sin embargo, los genes c^1_4 y c^2_4 se corresponden con las conexiones de la red U_{11} y V_{31}, respectivamente. Suponiendo un caso extremo, consideremos la posición del cromosoma i=13. En C_2, dicho gen se corresponde con la conexión de la red U_{33}. Sin embargo, en C_1 ni siquiera existe ese gen.

Para recombinar los genes correspondientes a los pesos de la red,hemos considerado dos opciones principales:

1. Realizar la recombinación gen a gen, mientras esto sea posible. En caso contrario, copiar al hijo el gen del padre que lo posee.
2. Realizar la recombinación conexión a conexión, mientras esto sea posible. En caso contrario, copiar al hijo el gen del padre que lo posee.

Ilustraremos ambos métodos mediante sendos ejemplos. Consideraremos un cruce aritmético, en el que dos genes *a* y *b* se recombinan para generar un nuevo gen *h* de la forma *h=(a+b)/2.*

Ejemplo 2:

Recombinación de C^1 y C^2 (ejemplo 1) para generar un descendiente *H* con 3 neuronas ocultas, mediante el método 1.

H= (3, 1.15, 2.95, 1.85, 3.85, 3.40, 1.8, 2.25, 2.2, 1.7, 6.7, 7.6, 6.5, 8.7,0.3, 0.2)

Ejemplo 3:

Recombinación de C^1 y C^2 (ejemplo 1) para generar un descendiente *H* con 3 neuronas ocultas, mediante el método 2.

H= (3, 1.15, 2.95, **3.3**, 0.25, 3.9, **0.3**, 3.9, 3.3, **1.7, 6.7, 7.6, 6.5**, 5.5, 0.7, **0.2**)

En el ejemplo 2 resaltamos en negrita aquellos genes que, a falta de poder ser recombinados, han sido heredados por el hijo a partir del padre que los posee.

El método 2 produce una mejor convergencia a soluciones óptimas, debidoque explota la información de las conexiones de las redes a recombinar. Sin embargo, el método 1 se

asemeja en mayor grado a un operador de mutación, ya que trata de combinar genes de diferentes conexiones de la red que, en un principio, no tienen relación alguna entre sí.

Para implementar el método 2, resulta necesario conocer una aplicación que nos permita hallar el gen correspondiente a una conexión concreta de la red codificada. Esta aplicación se muestra en la ecuación 4.7.

$$
\begin{aligned}
V_{ij} &\Rightarrow C_{M\cdot(i-1)+j} \\
U_{ij} &\Rightarrow \begin{cases} C_{M\cdot I + M\cdot(i-1)+j} & ; \text{redes Completa y de Elman} \\ C_{M\cdot I + O\cdot(i-1)+j} & ; \text{red de Jordan} \end{cases} \\
W_{ij} &\Rightarrow \begin{cases} C_{M\cdot I + M\cdot M + O\cdot(i-1)+j} & ; \text{red de Elman} \\ C_{M\cdot I + M\cdot O + O\cdot(i-1)+j} & ; \text{red de Jordan} \end{cases}
\end{aligned}
\tag{4.7}
$$

El Algoritmo 4.4 muestra el esquema principal del operador de cruce multiobjetivo, el cual genera un único descendiente $H=(h_1,h_2,...,h_{QH})$ a partir de dos padres $C_1=(c^1_1,c^1_2,...,c^1_{Q1})$ y $C_2=(c^2_1,c^2_2,...,c^2_{Q2})$, donde C_1 codifica un red neuronal recurrente dinámica del mismo modelo que C_2, y C_1 tiene un tamaño igual o menor a C_2.

1. Asignar n= Selección número de neuronas ocultas a partir de c^1_1 y c^2_1

2. H= generar cromosoma para codificar una red con n neuronas ocultas, con un total de Q_H genes

3. Asignar h_1= n

4. Para cada gen K=2..Q_H de H, hacer

- *4.1. L= conexión asociada al gen h_k*
- *4.2. Asignar P^1= valor del gen de C^1 asociado a la conexión L*
- *4.3. Asignar P^2= valor del gen de C^2 asociado a la conexión L*
- *4.4. Si existen P^1 y P^2,*

  asignar h_K= COMBINAR(P^1, P^2)
- *4.5. en otro caso, asignar h_K= P^2*

Algorimo 4.4. Procedimiento de cruce multiobjetivo para generar un único descendiente

La selección del número de neuronas ocultas en el algoritmo 4.4 (paso 1), se realiza mediante alguno de los tres criterios expuestos anteriormente. De este modo, logramos la recombinación de la estructura de los cromosomas padres que codifica la topología interna de las redes.

Por otra parte, la estructura de los cromosomas correspondiente a los pesos de la red se recombina en el paso 4. El método *COMBINAR* utiliza un cruce genético clásico de codificación real, para generar un nuevo gen a partir de dos progenitores (por ejemplo: cruce aritmético, heurístico de Wright, BLXα, etc.).

4.2. Operadores de cruce que generan dos descendientes.

El operador de cruce desarrollado en este apartado opera sobre dos padres C_1 y C_2 para generar dos nuevas soluciones descendientes, H_1 y H_2. Este operador actúa según se indica a continuación:

- La recombinación de la sección correspondiente a la estructura de la red neuronal asigna a cada uno de los dos descendientes el número de neuronas ocultas de uno de los padres, respectivamente.
- El cruce de los pesos de la red combina genes asociados a una misma conexión, que es común en las dos redes progenitoras. Dicha combinación se lleva a cabo utilizando un operador de cruce típico de algoritmos genéticos mono-objetivo.

El algoritmo 4.5 explica en detalle el procedimiento de recombinación multiobjetivo de soluciones, suponiendo que C_1 tiene un tamaño igual o inferior a C_2.

1. H_1, H_2= generar cromosoma para codificar una red con c^1_1 y c^2_1 neuronas ocultas, respectivamente, con un total de Q_{H1} y Q_{H2} genes

2. Para cada descendiente H_j, j=1..2, hacer

3. Para cada gen K=2..Q_{Hj} de H_j, hacer

3.1. L= conexión asociada al gen h_k

3.2. Asignar P^1= valor del gen de C^1 asociado a la conexión L

3.3. Asignar P^2= valor del gen de C^2 asociado a la conexión L

3.4. Si existen P^1 y P^2, asignar h_K= COMBINAR(P^1, P^2)

3.5. en otro caso, asignar h_K= P^2

Algorimo 4.5. Procedimiento de cruce multiobjetivo para generar dosdescendientes

Mediante la política de asignación de estructura topológica entrepadres y descendientes (paso 1 del algoritmo 4.5), conseguimos que el operador de cruce no altere la diversidad de soluciones. Esta medida evita una convergencia prematura y una diversidad excesiva, ya que la diversidad de estructuras topológicas no varía dentro de la población, salvo por la acción de los operadores de selección y reemplazamiento.

Por otra parte, el cruce de la estructura que codifica los pesos de las redes se realiza conexión a conexión, en lugar de ser gen a gen. Este hecho permite lacombinación de redes de diferente tamaño.

La figura 4.7 ilustra un ejemplo de la recombinación de dos redesneuronales recurrentes completas con una neurona de entrada, una neurona de salida, y dos y tres neuronas ocultas, respectivamente. Del mismo modo, se obtienen dos nuevas redes recurrentes completas descendientes, con dos y tres neuronas ocultas.

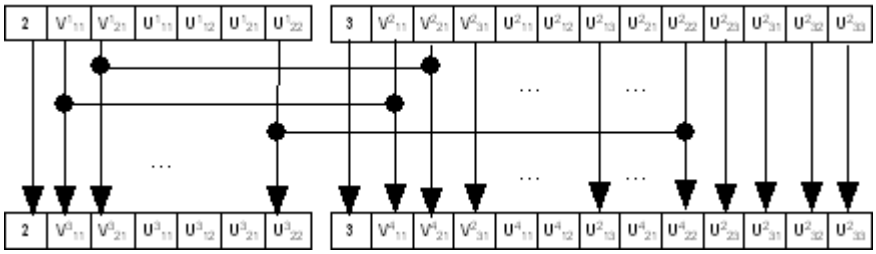

Figura 4.7. Ejemplo de cruce multiobjetivo de dos redes recurrentes completas, para generar dos descendientes

En el ejemplo de la figura 4.7, las dos nuevas soluciones generadas mantienen el número de neuronas ocultas de las dos soluciones progenitoras, respectivamente. Los genes de los descendientes contienen la combinación de los genes padres asociados a la misma conexión, si estos la comparten. En caso contrario, el valor de los genes descendientes será el valor del gen padre que comparta dicha conexión con el hijo.

5. El operador de mutación.

El operador de mutación altera el valor de los genes de un cromosoma, de acuerdo a un factor de mutación por gen p_m. El valor p_m es un parámetro del algoritmo, que toma un valor de probabilidad en el intervalo [0, 1]. Considerando la representación de una red neuronal recurrente dinámica introducida en el apartado 2, podemos plantear dos tipos de mutaciones que pueden tener lugar en un cromosoma:

- **Mutación estructural.** Este tipo de mutación altera los genes que definen la estructura interna de una red. Al aplicar este tipo de mutación, se pueden generar nuevos genes, asociados a nuevas conexiones en la red, o eliminar genes ya existentes. Por otra parte, la mutación de los genes correspondiente a la codificación de la estructura de la red neuronal conlleva la reorganización interna del mismo cromosoma, que puede crecer o decrecer en tamaño.
- **Mutación genética.** Este tipo de mutación altera los genes correspondientes a los pesos de las conexiones de la red. Puede establecerse, por tanto, una analogía con el operador

de mutación presentado para el problema del entrenamiento, estudiadoen los capítulos 2 y 3.

La mutación estructural conlleva una alteración del cromosoma mucho más importante que la mutación genética, ya que su aplicación implica la eliminación o generación de un nuevo conjunto de genes. Es por ello que resulta necesario introducir dosfactores de mutación, p^1_m y p^2_m, para controlar el efecto de la mutación estructural:

- Mediante p^1_m se seleccionará el tipo de mutación a realizar. Se escogerá la mutación estructural con probabilidad p^1_m, y la mutación genética con probabilidad 1-p^1_m.
- Mediante p^2_m se controla la mutación de los genes susceptibles de ser mutados, una vez aplicado el primer factor de mutación. El valor de un gen será alterado con probabilidad p^2_m.

El algoritmo 4.6 muestra el procedimiento a seguir para aplicar el operador de mutación multiobjetivo. El paso 2.6 ilustra cómo el valor de los genes correspondientes a las nuevas conexiones generadas son asignados a un valor aleatorio perteneciente a un intervalo válido, proporcionado por el usuario. De este modo, la solución generada contieneinformación de la solución inicial, y también nuevos valores que ayudan a explorar el espacio de soluciones, dentro del esquema evolutivo del algoritmo.

Los procedimientos *MUTACIONENTERA* y *MUTACIONREAL* alteran el valor de un gen mediante un operador genético clásico de mutación, cuando el gen a mutar está codificado mediante codificación entera o real, respectivamente.

Ilustraremos el funcionamiento del operador de cruce mediante un ejemplo. La figura 4.8 muestra la mutación de una red recurrente completa con una entrada, una salida y dos neuronas ocultas. Tras la mutación estructural, la nueva red contiene tresneuronas ocultas.

1. *Asignar p_1= número aleatorio en [0, 1]*
2. *Si $p_1 < p^1_m$, entonces hacer*
   - *2.1. Asignar p_2= número aleatorio en [0, 1]*
   - *2.2. Si $p_2 < p^2_m$, entonces*
   - *2.3. Asignar c'_1= valor del gen c_1*
   - *2.4. Asignar c_1= MUTACIONENTERA(c_1)*
   - *2.5. Reestructurar cromosoma C, de modo que cada gen c_K contenga el valor de la conexión asociada, previo a la mutación*
   - *2.6. Para cada gen c_K asociado a una nueva conexión generada, Asignar c_K= valor aleatorio válido*
3. *Si $p_1 \geq p^1_m$, hacer*
   - *3.1. Para cada gen c_K, K=2..Q*
     - *Asignar p_2= número aleatorio en [0, 1]*
     - *Si $p_2 < p^2_m$, asignar c_K= MUTACIONREAL(c_k)*

Algoritmo 4.6. Procedimiento de mutación multiobjetivo

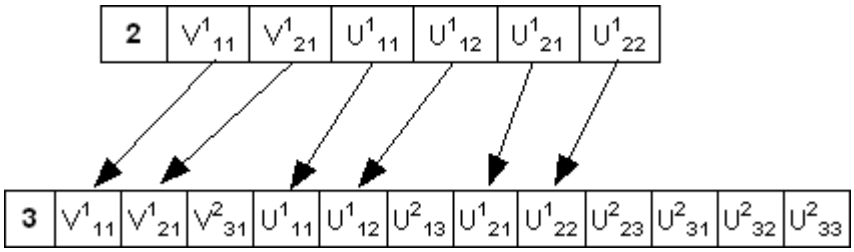

Figura 4.8. Ejemplo de mutación estructural de un cromosoma quecodifica una red recurrente completa con una entrada, una salida y dos neuronas ocultas

6. Entrenamiento y optimización de RNRD mediante algoritmos evolutivos multiobjetivo.

En este apartado, utilizamos los conceptos y métodos introducidos en las secciones anteriores para encontrar una red neuronal recurrente dinámica conestructura topológica óptima

y entrenada, mediante el uso de algoritmos de optimización evolutivos multiobjetivo. Del mismo modo, nos basaremos en los algoritmos evolutivos híbridos expuestos en el capítulo 3 para construir modelos híbridos de optimización evolutiva multiobjetivo, y los aplicaremos al problema del entrenamiento y optimización de los modelos de redes recurrentes considerados.

Las secciones 5.1 y 5.2 muestran la utilización de los algoritmos SPEA-II [ZIT01] y NSGA-II [DEB02], para resolver el problema planteado. Seguidamente, el apartado 6 expone el desarrollo de los algoritmos híbridos multiobjetivo, partiendo de los esquemas de los modelos fundamentales.

6.1. El algoritmo SPEA-II.

El algoritmo SPEA-II [ZIT01] hace evolucionar una población de soluciones mediante un esquema genético. No obstamte, la comparación y selección de soluciones la realiza considerando el criterio de optimalidad de Pareto. Los resultados proporcionados por el algoritmo son un conjunto de soluciones que forman la frontera de Pareto que se ha encontrado durante el proceso de evolución.

Durante el proceso de evolución, el algoritmo discrimina entre dos tipos de soluciones: dominadas y no dominadas. El conjunto de soluciones no dominadas es llamado *archivo Pareto*, mientras que el resto de soluciones forman la población común.

Pese a la discriminación entre los conjuntos de archivo Pareto y población, el algoritmo aplica el proceso evolutivo a la unión de ambos conjuntos, con el fin de poder profundizar en la búsqueda sobre zonas prometedoras del espacio de soluciones. La Figura 4.9 muestra el esquema principal del algoritmo SPEA-II.

El Algoritmo 4.7 muestra el esquema básico de SPEA-II, adaptado para utilizar los diferentes métodos de cruce estudiados en el apartado 3.

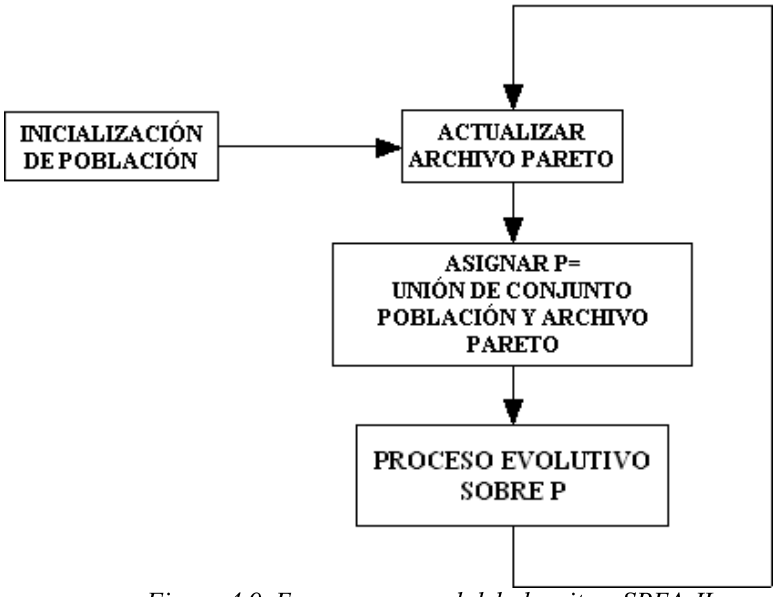

Figura 4.9. Esquema general del algoritmo SPEA-II

1. Asignar t= 0; P(t)= Inicializar población
2. Evaluar soluciones en P(t). Calcular Fitness Multiobjetivo
3. Asignar archivo Pareto A= soluciones no dominadas de P(t)
4. Mientras (no se cumpla condición de parada), hacer
4.1. Asignar T= P(t)+A
4.2. Asignar T'= selección de K soluciones sobre T
4.3. Asignar H= cruce de soluciones en T'
4.4. Asignar H'= mutación de soluciones en H
4.5. Evaluación de soluciones en H'.
4.6. Reemplazamiento de soluciones en P(t) y T por H'
4.7. Asignar S'= Soluciones no dominadas de T
[4.8. Si el número de soluciones en P(t) con igual número de neuronas ocultas es mayor que un umbral, reinicializar P(t)]
4.9. Asignar A= A+S' y reestructurar A
5. Devolver A

Algoritmo 4.7. Procedimiento del método de optimización multiobjetivo SPEA-II

El algoritmo comienza inicializando la población con un total de N soluciones generadas aleatoriamente mediante una distribución uniforme. Seguidamente, lassoluciones son evaluadas en el paso 2. Las soluciones recién evaluadas que son no dominadas constituyen el archivo

Pareto inicial. El paso 3 realiza esta operación, y reestructura el archivo Pareto en caso de que el número de soluciones no dominadas sea superior al máximo permitido.

Seguidamente, en el paso 4 da comienzo el proceso evolutivo. En primer lugar, las población *P* y el archivo Pareto *A* son fusionados en una misma población, *T*. Sobre esta población se aplican los operadores evolutivos de selección, cruce y mutación, siguiendo un esquema estacionario con reemplazamiento elitista a los padres (pasos 4.1 a 4.6). Por último, una generación evolutiva finaliza mediante el cómputo de las nuevas soluciones no dominadas en los descendientes generados, y la actualización del archivo Pareto.

Las operaciones más relevantes del algoritmo 4.7 son el cálculo del fitness multiobjetivo (realizado en el paso 4.5 del algoritmo, junto a la evaluación de las soluciones) y la reestructuración del archivo Pareto. A continuación, detallamos cadauna de ellas.

El cálculo del fitness multiobjetivo proporciona un mecanismo para poder ordenar el conjunto de soluciones, de acuerdo con el criterio de optimalidad de Pareto. Está basado en el cómputo de dos valores, *S(i)* y *R(i)*, para cada solución *i* de la población conjunta *T,* según se indica a continuación:

$$S(i) = |\{ j :\ j \in T \wedge i > j \}| \tag{4.8}$$

$$R(i) = \sum_{j \in T \wedge j > i} S(j) \tag{4.9}$$

El valor *S(i)* informa del número de soluciones dominadas por la solución *i*, mientras que *R(i)* proporciona el número de soluciones dominadas por todas las soluciones que dominan a *i*.

De acuerdo con los valores *S(i)* y *R(i)*, una solución i perteneciente a *T* será mejor cuanto menor sea el valor *R(i)*. El cálculo del fitness multiobjetivo utiliza el valor *R(i)* y una medida *D(i)* para estimar la bondad de una solución. El valor *D(i)* proporciona una medida sobre la densidad de la solución *i*; es decir, el número de soluciones similares a *i* existentes en *T:*

$$D(i) = \frac{1}{\sigma_i^k + 2} \tag{4.10}$$

El valor σ_i^k denota la distancia Euclídea del cromosoma *i* al *k*-ésimo cromosoma más cercano, donde *k* toma, por norma general, el siguiente valor:

$$k = \sqrt{|T|}$$

Una vez calculados los valores *R(i)* y *D(i)*, para toda solución *i* de *T*, el fitness multiobjetivo de una solución se calcula según la siguiente ecuación:

$$F(i) = R(i) + D(i) \tag{4.11}$$

Atendiendo a la notación anterior, una solución *i* será mejor que otra solución *j* si *F(i)<F(j)*. De este modo, los operadores genéticos clásicos de selección y reemplazamiento pueden operar sobre el conjunto de soluciones, considerando el valor del fitness multiobjetivo *F(·)*, en lugar de cada función objetivo por separado.

Otro aspecto importante a tener en cuenta, dentro del esquema del algoritmo 4.7, es la reestructuración del archivo Pareto *A*. Cada vez que el algoritmo ejecuta el paso 4.9, el conjunto *A* se modifica añadiendo las nuevas soluciones no dominadas de *T*. Una vez finalizado, podemos encontrar diversas situaciones:

- **Soluciones repetidas en *A***. En este caso, eliminamos las copias redundantes de cada solución repetida, dentro del archivo Pareto.
- **Soluciones dominadas en *A*.** En caso de que al añadir a *A* las nuevas soluciones de *T*, exista dominancia ente dos soluciones del archivo Pareto, se eliminan las soluciones dominadas del mismo.
- **Tamaño de *A* pequeño.** En este caso, copiamos las mejores soluciones dominadas de *P* al archivo Pareto, hasta obtener el tamaño requerido para *A*.
- **Tamaño de *A* excesivo.** En este caso, las peores soluciones de *A* son eliminadas iterativamente hasta obtener el tamaño requerido para *A*.

El algoritmo multiobjetivo SPEA-II ha sido utilizado para optimizar y entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 10000 soluciones evaluadas
- **Probabilidad de cruce:** 1.0
- **Probabilidad de mutación:** p_1=0.5 , p_2= 0.1
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** entero en [3, 12] para el número de neuronas ocultas, y número real en el intervalo [-5, 5] para los pesos
- **Operador de selección:** Torneo binario
- **Operador de cruce:** Multiobjetivo que genera 2 descencientes. Heurístico para codificación de los pesos
- **Operador de mutación:** Multiobjetivo por Desplazamiento
- **Reemplazamiento:** Padres (PA)
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** mínimo=3; máximo=12

Se han realizado 30 ejecuciones del algoritmo SPEA-II con los parámetros anteriores, para cada tipo de red.

Las tablas A.56 a A.70 muestran los resultados obtenidos por las ejecuciones del algoritmo SPEA-II, para entrenamiento y optimización de redes neuronales recurrentes de Elman, Jordan y Completas. No obstante, las tablas 4.1 a 43 muestran un resumen de las mismas, calculando la frontera de Pareto obtenida tras las 30 ejecuciones de cada algoritmo.

Tabla 4.1: Frontera de Pareto encontrada por el algoritmo SPEAII, tras el entrenamiento y optimización de redes neuronales recurrentes de Elman, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	3	1211.612357	1357.596682
	7	279.216653	324.081897

	6	917.618774	1028.553181
CATS2			
	9	3185.013174	3198.089904
	6	4073.335891	4118.717967
	10	985.128645	971.627420
	4	13066.302659	12840.313898
	3	13157.956142	12901.080781
	5	11616.150021	11424.216732
CATS3			
	9	209.864348	217.305246
	4	943.771588	958.299420
	5	646.864610	655.071419
	6	247.193060	257.394699
	3	1624.835756	1626.445702
CATS4			
	3	699.934260	780.703637
	4	344.302658	460.126916
	5	175.415155	208.344637
	10	174.623901	206.408560
CATS5			
	9	373.749381	472.980527
	5	2192.547194	2382.741134
	3	3579.048103	3567.663295
	4	3047.877848	3054.993284
	7	1163.179374	1204.122908
	6	2117.565614	2151.689177
	8	678.842696	772.955494
	11	234.666596	317.777498

Tabla 4.2: Frontera de Pareto encontrada por el algoritmo SPEAII,tras el entrenamiento y optimización de redes neuronales recurrentes Completas, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			

	3	212.566270	216.460296
	4	172.693376	172.668078
CATS2			
	4	172.984605	172.568321
	3	236.324966	238.171396
CATS3			
	4	146.886895	154.455691
	3	174.824110	182.665624
CATS4			
	5	150.658370	152.505309
	3	162.957225	162.868958
CATS5			
	3	159.249976	231.534720

Tabla 4.3: Frontera de Pareto encontrada por el algoritmo SPEAII,tras el entrenamiento y optimización de redes neuronales recurrentes de Jordan, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	9	549.935760	546.384543
	8	1271.615559	1257.192738
	6	1551.634700	1530.153504
	7	1270.749749	1310.487888
	3	2598.531852	2663.874007
CATS2			
	3	12987.146700	12731.411532
	10	2425.145858	2406.997527
	6	3406.697142	3355.808254
	8	2860.284100	2807.934567
CATS3			
	6	725.402342	743.381530
	3	2420.567463	2440.730999
	9	545.781910	581.391425
	4	1964.810843	1999.952363

	5	876.749611	901.818003
	11	223.680171	231.881293
CATS4			
	3	579.995947	994.278836
	7	174.158869	247.743374
	8	184.946959	222.703235
	4	563.969354	718.405183
CATS5			
	4	4144.105186	4315.418355
	5	3805.927843	3934.585487
	10	775.611814	904.770673
	7	1296.801232	1431.832614
	6	2530.984468	2770.303455
	3	4981.554822	5188.117762

Los resultados obtenidos por SPEA-II serán contrastados con los proporcionados por NSGA-II, en el apartado 5.3. Del mismo modo, analizaremos qué algoritmo es el que mejores resultados ha proporcionado para resolver el problema del entrenamiento y optimización de una red neuronal recurrente dinámica.

6.2. El algoritmo NSGA-II.

En contraposición al algoritmo SPEA-II, NSGA-II [DEB02] no discrimina la población en dos conjuntos de soluciones diferentes. En su lugar, realiza una ordenación de la población por rangos, basándose en el criterio de optimalidad de Pareto.

El método de ordenación de soluciones asegura que las mejores soluciones formarán parte de la población en la siguiente iteración, por lo que este algoritmo no necesita la incorporación de un mecanismo de elitismo externo. La figura 4.10 muestra la estructura de funcionamiento básica del algoritmo NSGA-II, mientras que el Algoritmo 4.8 detalla cada paso a seguir.

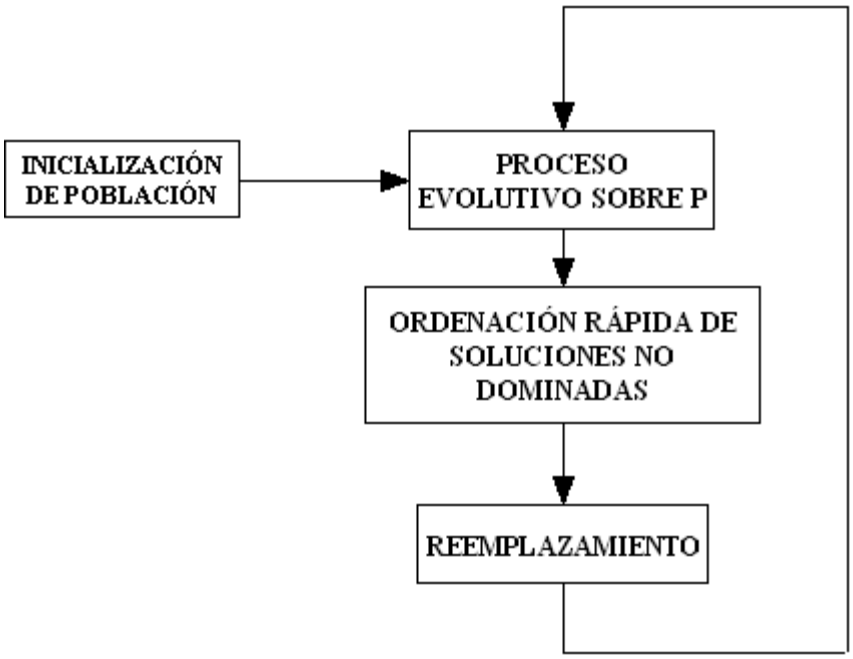

Figura 4.10. Esquema general del algoritmo NSGA-II

El algoritmo NSGA-II comienza inicializando un total de N soluciones a partir de una distribución uniforme. Seguidamente, estas soluciones son evaluadas en cada objetivo en el paso 2. El paso 3 realiza el proceso evolutivo de selección, cruce y mutación de soluciones, y realiza una ordenación de los padres y descendientes considerando el criterio de optimalidad de Pareto. Las N mejores y más diversas soluciones entre padres e hijos serán utilizadas para reemplazar a la población en el paso 3.8. Al finalizar el proceso evolutivo, el algoritmo NSGA-II devuelve la población de individuos, entre los cuales seleccionamos sólo los no dominados. Una descripción más detallada de este método puede consultarse en [DEB02].

El algoritmo multiobjetivo NSGA-II ha sido utilizado para optimizar y entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 10000 soluciones evaluadas
- **Probabilidad de cruce:** 1.0
- **Probabilidad de mutación:** p_1=0.5 , p_2= 0.1
- **Tamaño de la población**: 50 individuos

- **Alelos de un gen:** entero en [3, 12] para el número de neuronas ocultas, y número real en el intervalo [-5, 5] para los pesos
- **Operador de selección:** Torneo binario
- **Operador de cruce:** Multiobjetivo que genera 2 descencientes. Heurístico para codificación de los pesos
- **Operador de mutación:** Multiobjetivo por Desplazamiento
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** mínimo=3; máximo=12

1. Asignar t= 0; P(t)= Inicializar población con N soluciones
2. Evaluar soluciones en P(t).
3. Mientras (no se cumpla condición de parada), hacer
3.2. Asignar T'= selección de N soluciones sobre T
3.3. Asignar H= cruce de soluciones en T'
3.4. Asignar H'= mutación de soluciones en H
3.5. Evaluación de soluciones en H'.
3.6. T= {H'}+{P(t)}
3.7. Ordenación rápida multiobjetivo elitista de T
3.8. Reemplazamiento de soluciones en P(t) por T
4. Devolver P

Algoritmo 4.8. Procedimiento general del método de optimización multiobjetivo NSGA-II

Se han realizado 30 ejecuciones del algoritmo NSGA-II con los parámetros anteriores, para cada tipo de red. Las tablas A.71 a A.85 muestran los resultados obtenidos por las ejecuciones del algoritmo, para entrenamiento y optimización de redes neuronales recurrentes de Elman, Jordan y Completas. No obstante, las tablas 4.4 a 46 muestran un resumen de las mismas, calculando la frontera de Pareto obtenida tras las 30 ejecuciones de cada algoritmo.

Tabla 4.4: Frontera de Pareto encontrada por el algoritmo NSGAII,tras el entrenamiento y optimización de redes neuronales recurrentes de Elman, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	3	371.602600	377.238200
	10	183.317600	183.155000
	6	193.325500	193.169600
	4	214.056400	218.516400
CATS2			
	5	322.054500	319.154500
	3	3247.698000	3279.650000
	4	1406.578000	1439.226000
	6	213.379500	214.408500
CATS3			
	5	159.443200	169.396600
	9	156.156200	164.169000
	3	413.954700	417.789500
	4	171.018300	178.976500
CATS4			
	4	154.388000	161.203300
	8	147.990600	148.288000
	3	261.304700	323.518600
CATS5			
	3	412.863100	506.518500
	5	197.520500	261.557800
	4	266.493000	354.188100
	6	164.999600	259.077300

Tabla 4.5: Frontera de Pareto encontrada por el algoritmo NSGAII,tras el entrenamiento y optimización de redes neuronales recurrentes Completas, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	3	157.000200	156.547600
CATS2			
	3	151.287900	151.630900
CATS3			
	3	136.504900	143.740000
CATS4			
	3	140.125200	141.298000
	4	133.966500	139.489900
CATS5			
	3	134.803900	204.923800

Tabla 4.6: Frontera de Pareto encontrada por el algoritmo NSGAII,tras el entrenamiento y optimización de redes neuronales recurrentes de Jordan, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	7	400.372400	399.408500
	4	659.944900	665.647100
	3	1199.863000	1179.431000
	5	518.473200	512.147800
CATS2			
	4	4396.847000	4342.308000
	8	1992.760000	1975.852000
	7	2220.280000	2211.965000
	5	4125.985000	4050.250000
	11	1644.863000	1636.244000
	9	1980.865000	1947.950000
	3	7511.521000	7374.704000

CATS3			
	8	366.096900	370.168800
	5	551.113700	562.324100
	7	369.910200	380.318100
	4	708.160200	724.454300
	3	803.128100	820.454400
CATS4			
	3	218.885900	359.432600
	6	168.260400	204.963900
	4	176.627600	252.827700
CATS5			
	9	573.748100	712.817200
	7	751.912400	899.385100
	3	1564.132000	1768.381000
	4	1370.481000	1445.530000
	5	1326.665000	1417.282000
	8	632.104500	754.302600

Los resultados de los algoritmos clásicos multiobjetivo, aplicados al problema del entrenamiento y optimización de una red neuronal recurrente dinámica, son analizados en el apartado 8, junto con los resultados proporcionados por los algoritmos multiobjetivo híbridos propuestos. Los métodos híbridos se presentan a continuación, en el apartado 7.

7. Entrenamiento y optimización de RNRD mediante algoritmos multiobjetivo híbridos.

La idea de combinar algoritmos multiobjetivo junto con otras técnicas, de forma similar a cómo lo hace un algoritmo memético como los estudiados en el capítulo 3, surge a finales de la década de los 90. Desde entonces, se han desarrollado diversos trabajos tanto de índole teórica como práctica [ISH98][ISH03][JAS02][LO00], obteniendo resultados muy prometedores. En el

caso de la optimización de redes neuronales mediante métodos multiobjetivo híbridos, en la bibliografía podemos encontrar algunas propuestas, como por ejemplo [HER02][HUS02] [HUS03].

La mayoría de los trabajos realizados están orientados hacia la optimización de la topología de una red neuronal feedforward. No obstante, el trabajo que hay sobre aplicación de algoritmos multiobjetivo híbridos para optimizar RNRD es muy escaso. De hecho, las referencias encontradas en la bibliografía acerca de estos modelos de red son de los autores de este documento [PEG05]. La motivación de este apartado surge a partir de los buenos resultados alcanzados por los algoritmos híbridos desarrollados en el capítulo 3, con el fin de mejorar el proceso evolutivo multiobjetivo de entrenamiento y optimización de RNRD.

El objetivo de esta sección es combinar los algoritmos evolutivos multiobjetivo, estudiados en el apartado anterior, junto con el operador de búsqueda local BFGS para mejorar el entrenamiento de cada red. Partiendo de los resultados obtenidos en el capítulo 3, utilizaremos la estrategia de hibridación baldwiniana para construir los diferentes modelos híbridos multiobjetivo. La sección 6.1 estudia la hibridación basada en el esquema del algoritmo SPEA-II, mientras que la sección 6.2 se centra en la hibridación basada en NSGA-II. Por último, la sección 6.3 compara los resultados obtenidos por ambos métodos.

7.1. El algoritmo híbrido SPEA2H.

El algoritmo híbrido SPEA2H surge a partir de la combinación del esquema principal del algoritmo evolutivo multiobjetivo SPEA-II, utilizando un modelo evolutivo genético estacionario, con un operador de búsqueda local.

El operador de búsqueda local se aplica tras el proceso de evolución, y antes de la evaluación de cada solución. Puesto que los mejores resultados, en entrenamiento de RNRD (capítulo 3), han sido proporcionados por la estrategia baldwiniana, implementaremos esta técnica de hibridación en el algoritmo desarrollado. La figura 4.11 muestra el esquema básico del algoritmo SPEA2H, el cual puede obtenerse modificando el ciclo general del algoritmo SPEAII de la figura 4.9.

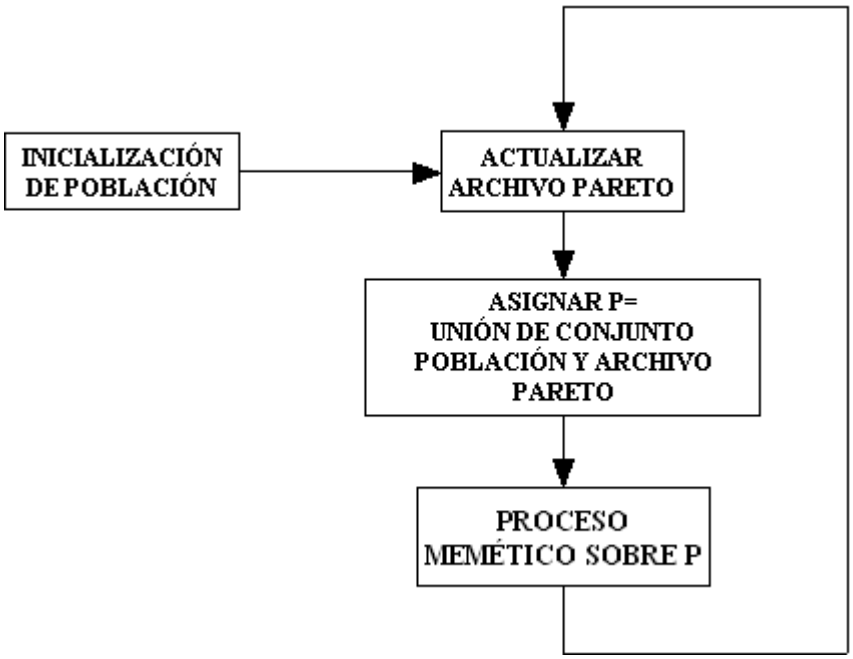

Figura 4.11. Esquema general del algoritmo híbrido basado en el método SPEA-II

El Algoritmo 4.9 muestra la modificación realizada al esquema básico de SPEA-II, para obtener el algoritmo híbrido HSPEA-II. La hibridación tiene lugar en el paso 4.5, donde se aplica un operador de búsqueda local sobre cada nueva solución generada por los operadores evolutivos de cruce y mutación. La estrategia de hibridación utilizadaes el modelo baldwiniano, debido a que este ha sido el que mejores resultados ha proporcionado trasel estudio realizado en el capítulo 3. El algoritmo multiobjetivo SPEA-II ha sido utilizado para optimizar y entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 10000 soluciones evaluadas
- **Probabilidad de cruce:** 1.0
- **Probabilidad de mutación:** p_1=0.5 , p_2= 0.1
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** enterno en [3, 12] para el número de neuronas ocultas, y número real en el intervalo [-5, 5] para los pesos
- **Operador de selección:** Torneo binario

- **Operador de cruce:** Multiobjetivo que genera 2 descencientes. Heurístico para codificación de los pesos
- **Operador de mutación:** Multiobjetivo por Desplazamiento
- **Reemplazamiento:** Padres (PA)
- **Búsqueda local:** BFGS
- **Iteraciones de búsqueda local:** 20
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** mínimo=3; máximo=12

1. Asignar t= 0; P(t)= Inicializar población
2. Para cada solución s_i de P(t), i=1..N, hacer
2.1. $s_i' = BL(s_i, I_1)$
2.2. Si ($f_1(s_i') < f_1(s_i)$),Asignar $f_1(s_i) = f_1(s_i')$
3. Asignar archivo Pareto A= soluciones no dominadas de P(t)
4. Mientras (no se cumpla condición de parada), hacer
4.1. Asignar T= P(t)+A
4.2. Asignar T'= selección de K soluciones sobre T
4.3. Asignar H= cruce de soluciones en T'
4.4. Asignar H'= mutación de soluciones en H
4.5. Para cada solución s_i de H', i=1..K, hacer
4.5.1. $s_i' = BL(s_i, I_1)$
4.5.2. Si ($f_1(s_i') < f_1(s_i)$),Asignar $f_1(s_i) = f_1(s_i')$
4.6. Reemplazamiento de soluciones en P(t) y T por H'
4.7. Asignar S'= Soluciones no dominadas de T
[4.8. Si el número de soluciones en P(t) con igual número de neuronas ocultas es mayor que un umbral, reinicializar P(t)]
4.9. Asignar A= A+S' y reestructurar A
5. Devolver A

Algoritmo 4.9. Procedimiento geneeral del método híbrido HSPEA2

Se han realizado 30 ejecuciones del algoritmo SPEA-II híbrido con los parámetros anteriores, para cada tipo de red. Las tablas A.86 a A.100 muestran los resultados obtenidos por las ejecuciones del algoritmo SPEA-II híbrido, para entrenamiento y optimización de redes neuronales recurrentes de Elman, Jordan y Completas. No obstante, las tablas 4.7 a 4.9 muestran un resumen de las mismas, calculando la frontera de Pareto obtenida tras las 30 ejecuciones de cada algoritmo.

Tabla 4.7: Frontera de Pareto encontrada por el algoritmo SPEA2H, tras el entrenamiento y optimización de redes neuronales recurrentes de Elman, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	5	156.526400	158.448900
	3	167.746300	167.404000
	8	153.260800	154.556100
CATS2			
	9	160.903500	161.639300
	11	153.757900	154.203800
	3	182.446500	182.748400
	4	165.405100	165.219400
	10	156.779900	156.662900
CATS3			
	7	143.289600	149.907200
	11	132.595100	140.889100
	3	145.853600	153.552900
	6	142.758600	151.261600
	10	140.264900	147.755100
	4	144.800400	152.732900
	9	138.869500	148.268200
CATS4			
	8	131.877000	132.234900
	9	128.621200	130.321000

	7	132.386400	134.299800
	3	146.054400	146.025800
	4	138.197000	139.519400
CATS5			
	9	134.833500	195.937300
	11	141.258900	189.820500
	7	138.378700	196.775800
	3	155.300800	207.125400

Tabla 4.8: Frontera de Pareto encontrada por el algoritmo SPEA2H, tras el entrenamiento y optimización de redes neuronales recurrentes Completas, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	7	143.005800	143.731000
	3	150.136700	150.001000
CATS2			
	8	143.630700	144.293100
	4	146.634400	146.978700
	3	149.954600	150.123500
CATS3			
	7	131.950100	139.353600
	4	132.124300	140.671500
	3	135.938800	144.212600
CATS4			
	6	123.654600	128.810000
	7	124.704500	126.636800
	3	134.563200	136.100800
	5	125.418600	129.451500
	8	123.945000	125.848500
	4	129.378500	130.480800
	11	123.536200	123.823500
CATS5			
	3	132.940800	204.010700

	5	131.214800	200.139400
	4	134.142000	201.286000

Tabla 4.9: Frontera de Pareto encontrada por el algoritmo SPEA2H, tras el entrenamiento y optimización de redes neuronales recurrentes de Jordan, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	7	191.942100	191.501400
	11	173.795100	173.256500
	10	174.157600	173.599800
	3	211.580800	212.141900
	4	200.857100	199.876200
	9	181.203200	181.033200
	8	181.859500	181.210100
CATS2			
	10	215.599000	219.550100
	7	234.988900	239.444500
	5	284.528400	289.971200
	9	218.693300	221.793200
	3	475.710400	475.263000
	4	365.556100	367.183400
	11	197.393400	200.982600
CATS3			
	10	151.613900	157.621500
	5	157.905100	164.353600
	9	154.943000	160.156800
	4	165.250100	173.494100
	3	165.635800	173.803900
	6	156.225700	162.754600
	8	153.842300	160.406800
	11	149.549200	156.143400
CATS4			
	8	146.345100	154.139100

	3	150.147600	169.426300
	11	146.808500	149.490300
	9	147.600600	154.056200
	5	148.880000	163.030500
	7	148.202200	160.214700
CATS5			
	11	171.381000	231.652400
	10	199.064100	242.440400
	7	211.644100	277.694300
	3	268.938600	336.760300
	9	200.123700	268.582800
	6	211.632200	301.240900

7.2. El algoritmo híbrido NSGA2H.

El algoritmo híbrido NSGA2H surge a partir de la combinación del esquema principal del algoritmo evolutivo multiobjetivo NSGA-II, con un operador de búsqueda local.

El operador de búsqueda local se aplica tras el proceso de evolución, y antes de la evaluación de casa solución. Al igual que para el algoritmo SPEA2H, implementaremos la técnica de hibridación baldwiniana en el método desarrollado. La figura 4.12 muestra el esquema básico del algoritmo NSGA2H, el cual puede obtenerse modificando el ciclo general del algoritmo NSGAII de la figura 4.9.

El Algoritmo 4.10 muestra la modificación realizada al esquema básico de NSGA-II, para obtener el algoritmo híbrido NSGA2H. Los cambios necesarios para realizar la hibridación tienen lugar en el paso 3.5. Para cada nueva solución descendiente generada mediante el cruce, en el paso 3.5.1 se aplica el operador de búsqueda local un número prefijado de iteraciones. Seguidamente, en el paso 3.5.2 se modifica el valor de adecuación f_1 de la solución original con el valor de adecuación de la solución mejorada. La estrategia de hibridación aplicada es por tanto la baldwiniana, debido a que esta ha sido la que mejores resultados nos ha proporcionado en el capítulo 3.

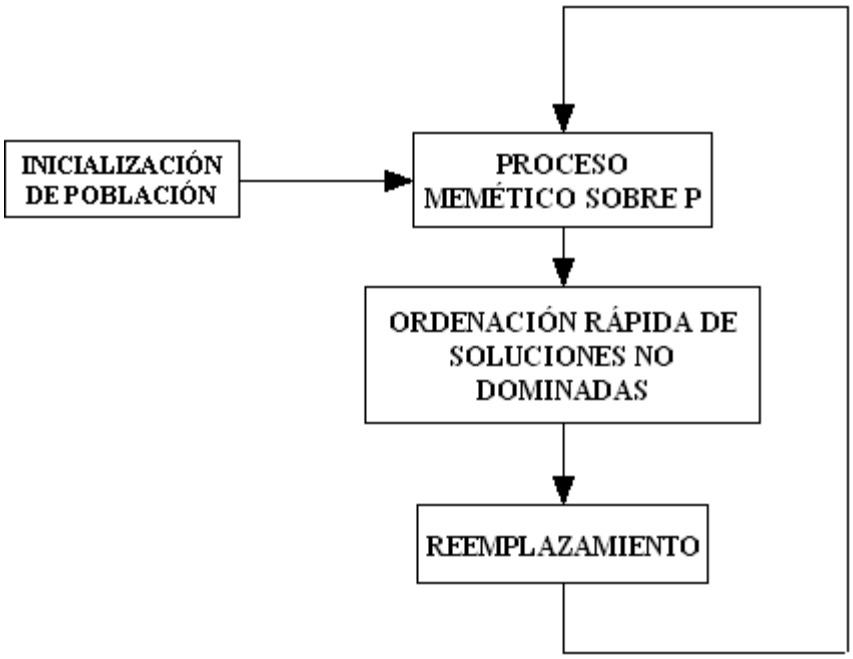

Figura 4.12. Esquema general del método híbrido NSGA2H

1. Asignar t= 0; P(t)= Inicializar población con N soluciones
2. Para cada solución s_i de P(t), i=1..N, hacer
2.1. s_i'= BL(s_i, I_1)
2.2. Si ($f_1(s_i') < f_1(s_i)$),Asignar $f_1(s_i) = f_1(s_i')$
3. Mientras (no se cumpla condición de parada), hacer
3.2. Asignar T'= selección de N soluciones sobre T
3.3. Asignar H= cruce de soluciones en T'
3.4. Asignar H'= mutación de soluciones en H
3.5. Para cada solución s_i de H', i=1..N, hacer
3.5.1. s_i'= BL(s_i, I_1)
3.5.2. Si ($f_1(s_i') < f_1(s_i)$),Asignar $f_1(s_i) = f_1(s_i')$
3.6. T= {H'}+{P(t)}
3.7. Ordenación rápida multiobjetivo elitista de T
3.8. Reemplazamiento de soluciones en P(t) por T
4. Devolver A

Algoritmo 4.10. Procedimiento general del método híbrido NSGA2H

El algoritmo multiobjetivo NSGA-II híbrido ha sido utilizado para optimizar y entrenar RNRD en el benchmark CATS, con los siguientes parámetros:

- **Criterio de parada:** 10000 soluciones evaluadas
- **Probabilidad de cruce:** 1.0
- **Probabilidad de mutación:** p_1=0.5 , p_2= 0.1
- **Tamaño de la población**: 50 individuos
- **Alelos de un gen:** entero en [3, 12] para el número de neuronas ocultas, y número real en el intervalo [-5, 5] para los pesos
- **Operador de selección:** Torneo binario
- **Operador de cruce:** Multiobjetivo que genera 2 descencientes. Heurístico para codificación de los pesos
- **Operador de mutación:** Multiobjetivo por Desplazamiento
- **Búsqueda local:** BFGS
- **Iteraciones de búsqueda local:** 20
- **Entradas a la red:** 1 (valor de la serie temporal en el instante actual)
- **Salidas de la red:** 1 (valor de la serie temporal en el instante siguiente)
- **Neuronas ocultas:** mínimo=3; máximo=12

Se han realizado 30 ejecuciones del algoritmo NSGA-II híbrido con los parámetros anteriores, para cada tipo de red.

Las tablas A.101 a A.115 muestran los resultados obtenidos por las ejecuciones del algoritmo NSGA-II híbrido, para entrenamiento y optimización de redes neuronales recurrentes de Elman, Jordan y Completas. No obstante, las tablas 4.10 a 412 muestran un resumen de las mismas, calculando la frontera de Pareto obtenida tras las 30 ejecuciones de cada algoritmo.

Tabla 4.10: Frontera de Pareto encontrada por el algoritmo NSGA2H, tras el entrenamiento y optimización de redes neuronales recurrentes de Elman, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	3	190.125100	191.573200
	4	175.859800	176.421300
	6	166.801300	167.436800
CATS2			
	4	235.786600	245.923200
	8	166.434000	168.433200
	3	583.660600	591.868600
CATS3			
	4	156.121200	166.444300
	3	158.909100	167.167100
	7	145.248100	153.481100
	6	146.992900	154.440300
CATS4			
	3	178.419700	177.727400
	8	148.886300	149.439900
	6	150.818100	150.914400
	5	150.313200	161.980100
	4	163.576300	164.099300
CATS5			
	7	165.441400	251.834800
	11	147.823100	211.155300
	3	253.733100	311.016000
	5	181.979500	259.310600

Tabla 4.11: Frontera de Pareto encontrada por el algoritmo NSGA2H, tras el entrenamiento y optimización de redes neuronales recurrentes Completas, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	4	163.584900	163.247100
	6	162.676300	162.634300
	3	164.879800	164.517800
CATS2			
	3	153.872900	153.862300
CATS3			
	6	152.731400	159.494400
	3	157.348000	164.318100
CATS4			
	6	139.372200	140.029200
	3	140.859800	141.641400
CATS5			
	4	25154.150000	30336.660000
	3	222682.600000	220522.900000
	6	843.136500	931.783700

Tabla 4.12: Frontera de Pareto encontrada por el algoritmo NSGA2H, tras el entrenamiento y optimización de redes neuronales recurrentes de Jordan, en el problema CATS

Problema	**Neuronas**	**MSE Entrenamiento**	**MSE Test**
CATS1			
	10	178.647700	177.932800
	5	218.639400	217.968500
	3	236.814300	235.348700
	7	190.581400	190.116100
	6	203.664200	202.742600
	4	227.928200	227.493800
	9	178.526000	178.033600
	11	175.355400	174.596900

	8	189.106300	188.304900
CATS2			
	3	594.569500	595.316500
	4	399.755800	401.158000
	9	234.306000	238.230300
	8	240.537100	242.436500
	6	309.660300	315.099600
	10	202.524700	205.246400
CATS3			
	10	153.928800	160.314300
	6	165.641800	172.459700
	4	175.752900	181.240200
	11	147.691100	154.547600
	9	153.537400	160.383800
	3	190.305300	197.130300
	7	156.920000	165.389100
CATS4			
	9	146.253700	151.392700
	3	150.643000	178.206000
	4	149.525400	168.359900
	8	146.579800	155.245600
	5	147.501900	159.520400
	10	144.972600	146.814900
	6	147.827400	157.357800
CATS5			
	7	196.843800	279.265700
	4	252.220800	342.083200
	8	168.976700	213.838100
	5	268.207900	340.104500
	6	234.229500	300.332100
	3	275.375100	420.691400

A continuación, el apartado 7 analiza los resultados obtenidos por los algoritmos multiobjetivo clásicos e híbridos, en el problema del entrenamiento y optimización de una red neuronal recurrente dinámica.

8. Experimentación.

Este apartado muestra la comparación de los resultados obtenidos porSPEA2 y NSGA2 clásicos e híbridos, para resolver el problema del entrenamiento y la optimización de RNRD.

Las figuras 4.13 a 4.27 muestran la frontera de Pareto encontrada, a lo largo de las 30 ejecuciones de cada algoritmo, para cada modelo de red y problema.

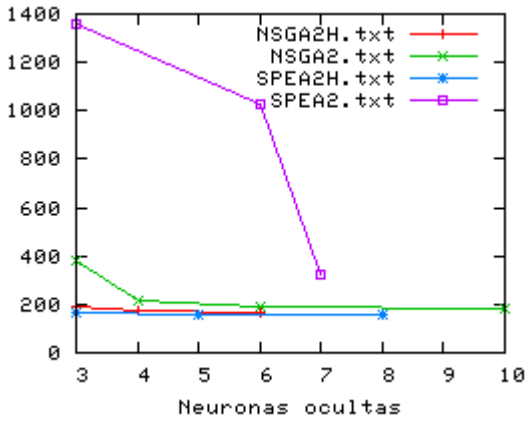

Figura 4.13. Frontera Pareto de red de Elman, problema CATS1

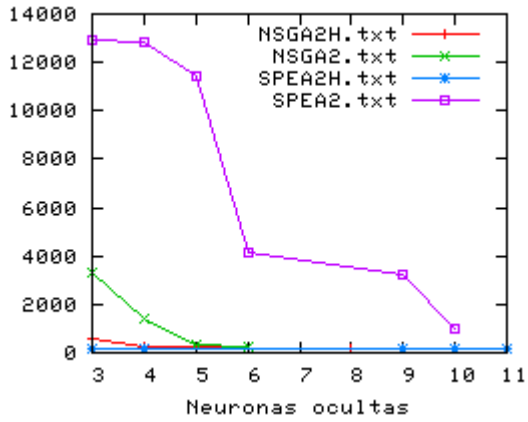

Figura 4.14. Evolución de diversidad en red de Elman, problema CATS2

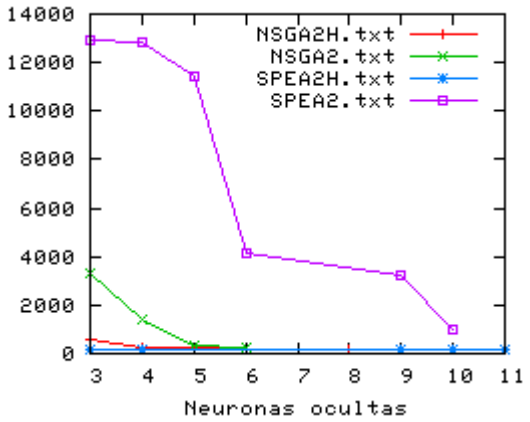

Figura 4.15. Frontera Pareto de red de Elman, problema CATS3

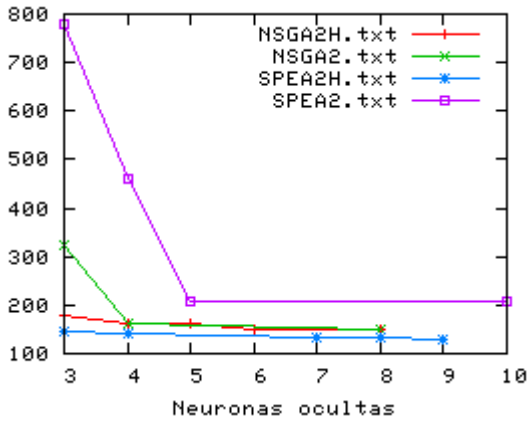

Figura 4.16. Frontera Pareto de red de Elman, problema CATS4

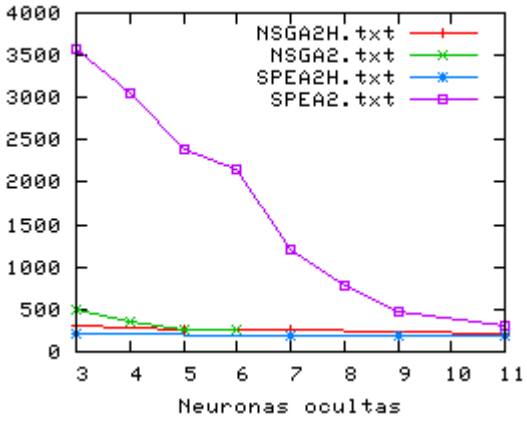

Figura 4.17. Frontera Pareto de red de Elman, problema CATS5

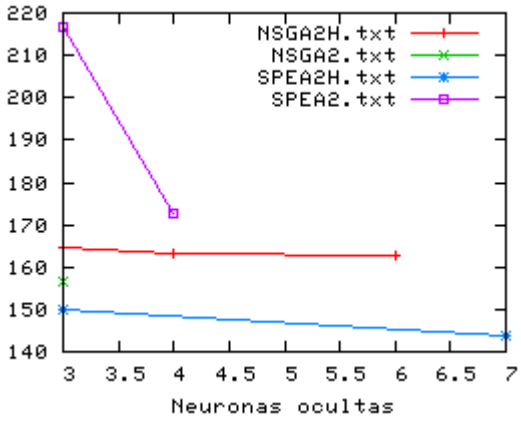

Figura 4.18. Frontera Pareto de red Completa, problema CATS1

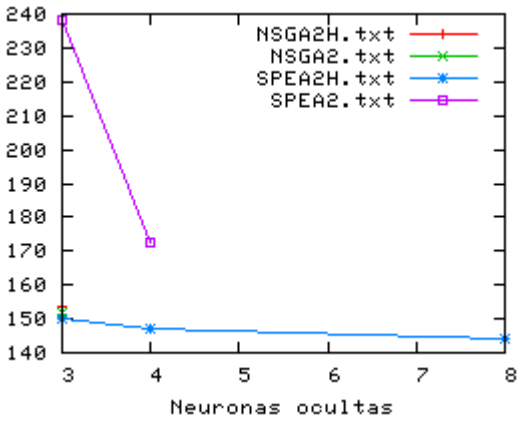

Figura 4.19. Frontera Pareto de red Completa, problema CATS2

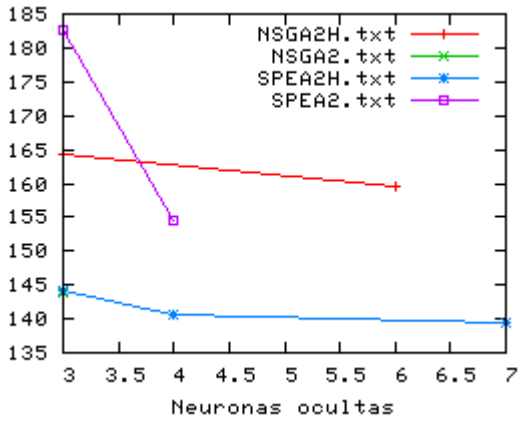

Figura 4.20. Frontera Pareto de red Completa, problema CATS3

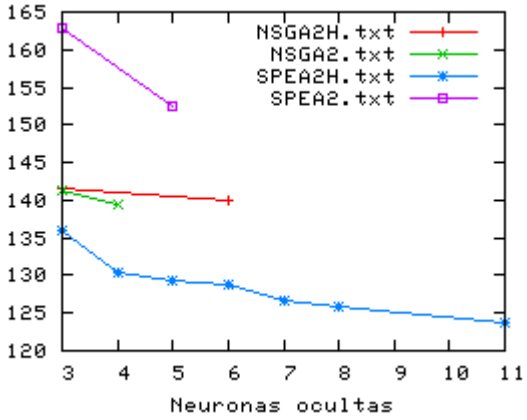

Figura 4.21. Frontera Pareto de red Completa, problema CATS4

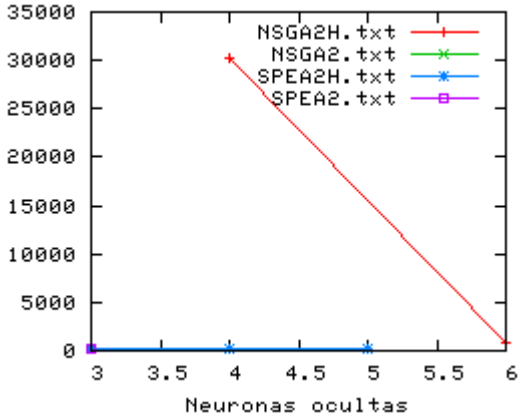

Figura 4.22. Frontera Pareto de red Completa, problema CATS5

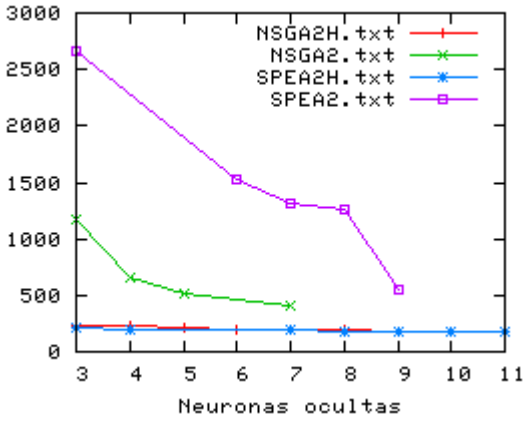

Figura 4.23. Frontera Pareto de red de Jordan, problema CATS1

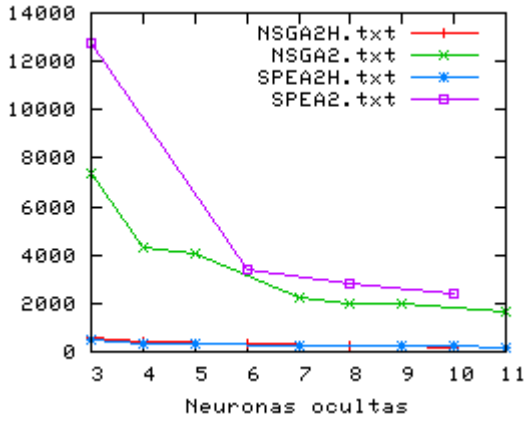

Figura 4.24. Frontera Pareto de red de Jordan, problema CATS2

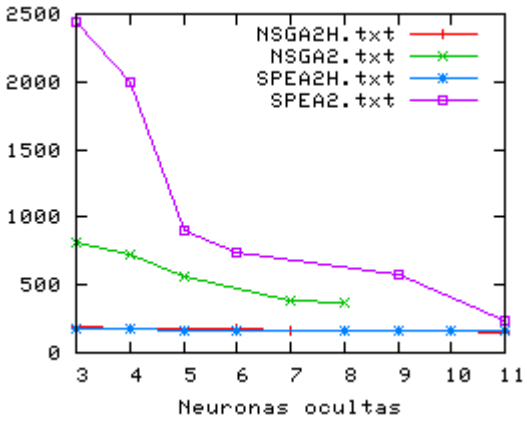

Figura 4.25. Frontera Pareto de red de Jordan, problema CATS3

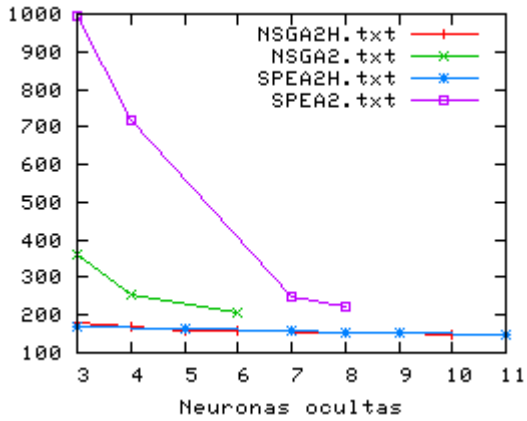

Figura 4.26. Frontera Pareto de red de Jordan, problema CATS4

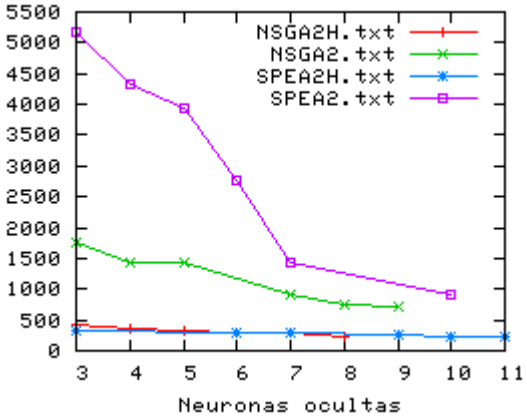

Figura 4.27. Frontera Pareto de red de Jordan, problema CATS5

Podemos observar la eficacia superior de los algoritmos multiobjetivo híbridos, frente a los métodos clásicos. Analizaremos cada tipo de algoritmo por separado.

Con respecto a los métodos multiobjetivo clásicos, podemos observar que SPEA-II obtiene una frontera de Pareto más amplia en la mayoría de los casos, abarcando un conjunto de redes con un número de neuronas ocultas y un error MSE de test muy variado.

No obstante, las soluciones proporcionadas por NSGA-II dominan a las obtenidas mediante SPEA-II, en gran parte de los problemas. Aunque la frontera de Pareto obtenida por NSGA-II no es tan amplia como la proporcionada por SPEA-II, las soluciones tienen mayor calidad.

Con respecto a los algoritmos multiobjetivo híbridos, se observa la situación contraria. Ambos modelos (SPEA2H y NSGA2H) son capaces de mejorar las fronteras de Pareto obtenidas por los métodos multiobjetivo clásicos.

Las soluciones del algoritmo híbrido SPEA2H mejoran a las de NSGA2H en gran parte de los problemas (redes de Elman y completas). En el caso del entrenamiento y optimización de redes de Jordan, podemos observar una equivalencia en las fronteras de Pareto de ambos métodos (figuras 4.23 a 4.27).

Dadas estas circunstancias, podemos afirmar que el algoritmo multiobjetivo híbrido SPEA2H ha proporcionado, en general, mejores resultados que el híbrido NSGA2H. Las razones atribuidas a este comportamiento tienen su base en las conclusiones dadas para los algoritmos híbridos de entrenamiento, en el capítulo 3:

El algoritmo multiobjetivo híbrido SPEA2H desarrollado sigue un esquema evolutivo estacionario, mientras que NSGA2H utiliza un esquema generacional. El hecho de aplicar un operador de búsqueda local sobre una población hace que el número de generaciones del algoritmo disminuya, si el criterio de parada es el número de evaluaciones de soluciones.

Por ejemplo, supongamos que el criterio de parada es evaluar 10000 soluciones, y el operador de búsqueda local hace 25 evaluaciones cada vez que es ejecutado. Suponiendo un tamaño de población igual a 50, la inicialización y primera evaluación de las soluciones conlleva un total de 50*25= 1250 evaluaciones.

En un esquema genético estacionario, en el que en cada generación se producen un total de 2 soluciones, cada generación utiliza un total de 2*25= 50 evaluaciones. Si el criterio de parada es 10000, entonces habrá un total de (10000-1250)/50= 175 generaciones. Estenúmero de generaciones es suficiente para que el proceso evolutivo actúe produciendo resultados adecuados mediante los operadores de selección, cruce, mutación y reemplazamiento.

Sin embargo, en un esquema genético generacional en el que cada iteración hace un total de 50*25= 1250 evaluaciones, el algoritmo itera un total de (10000-1250)/1250= 7 generaciones. Así, este modelo genético no tiene posibilidad de explotar el efecto de los operadores evolutivos en un número tan bajo de generaciones.

Capítulo 5

Resultados

1. Introducción.

Hasta el momento, los capítulos 2, 3 y 4 han mostrado la comparación de los diferentes modelos de algoritmos utilizados para entrenar RNRD en el benchmark *CATS*, con diferentes parámetros. Como resultado, hemos obtenido una serie de conclusiones que pueden ayudar a seleccionar un conjunto de operadores genéticos, estrategias de hibridación y modelos de evolución para resolver el problema del entrenamiento y la optimización de la estructura topológica de una RNRD. Para finalizar nuestro estudio, este capítulo profundiza en el análisis de los resultados obtenidos, abordando el problema según los criterios siguientes:

- Respecto al problema del entrenamiento, nos centraremos en el estudio del error MSE producido por cada modelo, teniendo como base las conclusiones obtenidas en los capítulos 2 y 3. Este estudio incluye la comparación estadística entre los métodos clásicos, evolutivos e híbridos, la cual no se había realizado hasta el momento. También analizaremos el tiempo de ejecución medio de cada algoritmo, para concluir qué tipo de modelos de entrenamiento podrían ser adecuados en diferentes tipos de situaciones donde el tiempo de ejecución sea un factor determinante o no. Finalmente, comparamos los métodos de entrenamiento propuestos con otras técnicas utilizadas previamente para predicción de series temporales en el benchmark *CATS*.

- Respecto al problema de la optimización de la estructura interna de una RNRD, compararemos los resultados obtenidos por los algoritmos propuestos en el capítulo 4 frente a una prueba de ensayo y error realizada para encontrar el número de neuronas óptimo de una RNRD. De este modo, podremos contrastar si los métodos propuestos cumplen el objetivo de reducir el número de experimentos necesario para optimizar y entrenar los modelos de redes recurrentes estudiados.
- Por último, validaremos el comportamiento de los modelos propuestos, aplicándolos para entrenar y optimizar RNRD en problemas de series temporales de índole económica.

El capítulo está organizado de la siguiente forma: El apartado 2 muestra la comparación entre los diferentes métodos aplicados al benchmark CATS y los modelos de algoritmos evolutivos e híbridos estudiados en este documento. El apartado 3 se centra en aplicaciones sobre series de datos reales de tipo económico. Utilizaremos los diferentes modelos de entrenamiento desarrollados para la predicción de cada serie, y realizamos un estudio de la mejora producida en el entrenamiento de cada modelo de red, en contraposición a los algoritmos clásicos de entrenamiento.

2. El benchmark CATS.

Partiendo del estudio de diversidad y convergencia realizado sobre algoritmos evolutivos e híbridos en los capítulos 2 y 3, este apartado pretende mostrar los resultados alcanzados por cada modelo de red en los métodos de entrenamiento que mejor comportamiento han presentado, para el problema CATS [LEN04].

En primer lugar, analizaremos los resultados de cada subproblema CATS por separado en las secciones 2.1 a 2.5. Finalmente, en la sección 2.6 compararemos los resultados de los mejores algoritmos, frente a los resultados de las diferentes técnicas que se han aplicado al problema.

2.1. Subproblema CATS1.

La tabla 5.1 muestra los resultados obtenidos por los métodos desarrollados en los capítulos 2 y 3, junto con el test de equivalencia estadística de Kruskal-Wallis.

Tabla 5.1: Resultados del error medio por los algoritmos híbridos en contraposición a los modelos genéticos clásicos y los algoritmos BPTT y RTRL, y equivalencia estadística

Algoritmos	Tiempo	MSE Mediano (trn)	MSE Mediano (tst)	pValue
SGA-BAL/PA/LI/BLX/DES FRNN	*8.17*	*147.932*	*147.774*	*1 (x)*
CHC-BAL FRNN	*10.27*	*148.111*	*148.239*	*0.5793 (x)*
SS FRNN	*8.47*	*154.554*	*154.438*	*0.008875 (-)*
SGA-BAL/PA/SO/BLX/ALE Elman	*17.27*	*159.786*	*161.198*	*0.001406 (-)*
SS Elman	*15.00*	*160.445*	*161.938*	*0.6467 (x)*
CHC-BAL Elman	*23.83*	*160.173*	*162.120*	*0.5058 (x)*
BFGS FRNN	*1.90*	*162.565*	*162.318*	*0.8476 (x)*
BFGS Jordan	*11.47*	*165.430*	*164.825*	*0.007129 (-)*
BFGS Elman	*11.27*	*165.126*	*165.063*	*0.3292 (x)*
SGA-BAL/PA/LI/ARI/ALE Jordan	*13.47*	*166.503*	*165.885*	*1.042e-10 (-)*
CHC-BAL Jordan	*18.17*	*167.805*	*167.212*	*0.004325 (-)*
SS Jordan	*11.37*	*171.577*	*171.152*	*0.0001211 (-)*
MGA/PE/LI/BLX/ALE FRNN	*5.87*	*175.049*	*174.726*	*0.02658 (-)*

Clearing/BLX FRNN	*5.87*	*175.227*	*175.579*	*0.4077 (x)*
GGA/PA/TO/BLX/DES FRNN	*6.00*	*182.086*	*183.743*	*0.2198 (x)*
MGA/PE/TO/BLX/ALE Elman	*6.13*	*205.164*	*210.895*	*1.208e-05 (-)*
Clearing/BLX Elman	*6.40*	*210.938*	*214.724*	*0.5346 (x)*
SGA/PE/TO/ARI/ALE FRNN	*6.00*	*222.715*	*223.860*	*0.1354 (x)*
GGA/PA/TO/HEU/DES Elman	*6.20*	*282.243*	*294.621*	*9.193e-06 (-)*
CHC/BLX Elman	*6.27*	*317.827*	*323.433*	*0.04758 (-)*
MGA/PE/LI/HEU/ALE Jordan	*4.27*	*379.575*	*376.489*	*0.06043 (x)*
SGA/PA/TO/HEU/DES Elman	*6.27*	*351.647*	*383.992*	*0.0345 (-)*
Clearing/BLX Jordan	*4.47*	*474.869*	*471.397*	*0.001336 (-)*
GGA/PA/TO/BLX/DES Jordan	*4.13*	*656.588*	*651.389*	*1.627e-08 (-)*
CHC/BLX FRNN	*4.47*	*985.654*	*985.911*	*1.794e-06 (-)*
CHC/BLX Jordan	*4.40*	*985.654*	*985.911*	*1 (x)*
SGA/PE/TO/HEU/DES Jordan	*4.40*	*1165.199*	*1160.770*	*0.0004337 (-)*
BPTT Jordan	*24.93*	*11415.440*	*11491.140*	*2.631e-11 (-)*
BPTT Elman	*22.38*	*11243.061*	*11481.100*	*0.8377 (x)*
RTRL Elman	*20.00*	*11262.080*	*12131.960*	*0.5535 (x)*
RTRL Jordan	*20.00*	*36728.930*	*36190.930*	*0.4589 (x)*

RTRL FRNN	*20.00*	*61801.790*	*60696.980*	*0.8126 (x)*

La columna 1 muestra el algoritmo y la red entrenada. Seguidamente, la columna 2 muestra el tiempo de ejecución medio del algoritmo. Las columnas 3 y 4 muestran el MSE de entrenamiento y test, respectivamente. Por último, la columna 5 ilustra el resultado del test de equivalencia estadística con un nivel de confianza de 0.05. Un símbolo *(x)* junto al valor del test indica la equivalencia con el algoritmo anterior, mientras que *(-)* significa que los resultados son estadísticamente peores.

Las conclusiones que pueden obtenerse a partir de la tabla 5.1 para el problema *CATS1* son:

- **Red que ha proporcionado mejores resultados:** red recurrente completa
- **Mejor algoritmo de entrenamiento para la red Completa** SGA-BAL y CHC-BAL
- **Mejor algoritmo de entrenamiento para la red de Elman:** SGA-BAL, CHC-BAL y Scatter Search
- **Mejor algoritmo de entrenamiento para la red de Jordan:** BFGS

Centrando el estudio sobre los métodos híbridos desarrollados en el apartado 3, el algoritmo genético estacionario híbrido con estrategia baldwiniana ha sido el que mejor comportamiento ha tenido. Sin embargo, la propuesta híbrida CHC-BAL presenta resultados adecuados, llegando a proporcionar resultados equivalentes a SGA-BAL en el caso de las redes recurrentes completas y de Elman. Independientemente del tipo de red, los peores resultados han sido alcanzados por los algoritmos clásicos de entrenamiento de RNRD. La Figura 5.1 muestra el ajuste y la predicción realizados por la mejor solución obtenida al entrenar una red neuronal recurrente completa, mediante el algoritmo SGA-BAL.

Si consideramos como prioridad principal el tiempo de ejecución, una única ejecución de los algoritmos de programación no lineal es mucho más rápida que el resto de métodos

evolutivos empleados. Sin embargo, las soluciones obtenidas dependen excesivamente de los pesos iniciales de la red, proporcionando así diversos tipos de solucionesen cuanto a precisión del MSE. Por ello, resulta necesario realizar una ejecución multiarranque de los algoritmos clásicos de entrenamiento para poder llegar a una solución adecuada. En los experimentos realizados, la ejecución multiarranque de los algoritmos tiene como condición de parada el tiempo máximo de ejecución del algoritmo híbrido que mejores resultadosha proporcionado.

Respecto a los algoritmos evolutivos, podemos comprobar que el tiempo de ejecución de los métodos híbridos es mayor que el de los métodos clásicos. Estoes debido a la utilización del operador de búsqueda local, el cual consume un tiempo de ejecución considerable en el cálculo del gradiente del error para cada solución. No obstante, la diferencia de tiempo con respecto a los algoritmos evolutivos híbridos para entrenar RNRD en este problema es de aproximadamente 3 segundos.

Como conclusión de los párrafos anteriores, proponemos el uso de los algoritmos híbridos desarrollados en el capítulo 3 para entrenar RNRD (en especial el método híbrido SGA-BAL), en situaciones en las que se necesite un aprendizaje óptimo yel tiempo de ejecución para el proceso de entrenamiento no sea un aspecto clave. En cambio,si la aplicación requiere que la red neuronal aprenda correctamente en poco tiempo, la mejor opción a considerar son los métodos de programación no lineal Quasi-Newton (BFGS y LM).

Hemos realizado un experimento mediante el método de ensayo yerror para calcular el número de neuronas ocultas óptimas de cada red en el problema CATS1, con los métodos de entrenamiento genéticos híbridos y clásicos (utilizando estrategia evolutiva estacionaria). Tras 30 ejecuciones con 3, 4, 5, 6, 7, 8, 9, 10, 11 y 12 neuronas ocultas encada modelo de red (un total de 3 ejecuciones por configuración neuronal oculta), hemos construido una frontera Pareto manualmente, la cual utilizamos para compararla con los métodos de entrenamiento y optimización desarrollados.

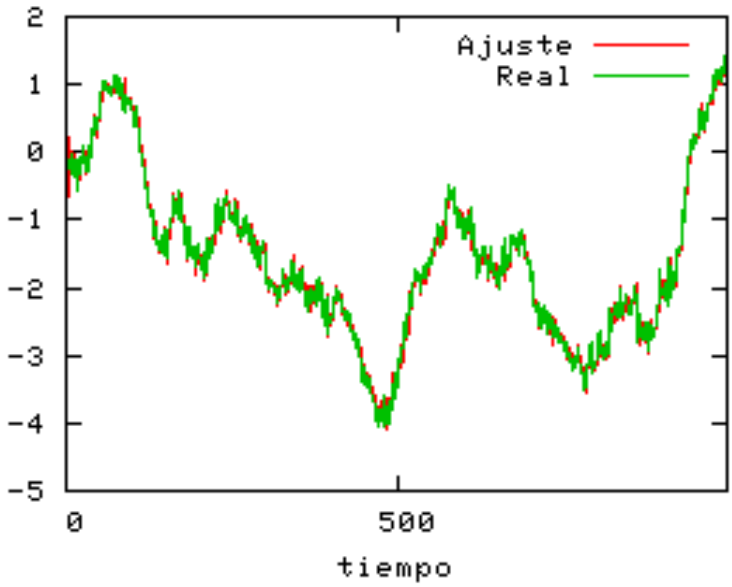

Figura 5.1. Ajuste y predicción realizado en el problema CATS1

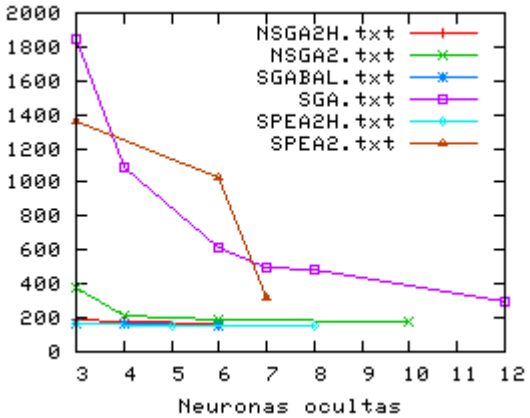

Figura 5.2. Fronteras de Pareto calculadas para redes de Elman

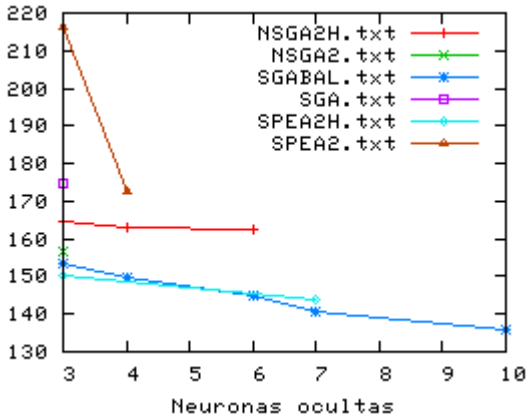

Figura 5.3. Fronteras de Pareto calculadas para redes Completas

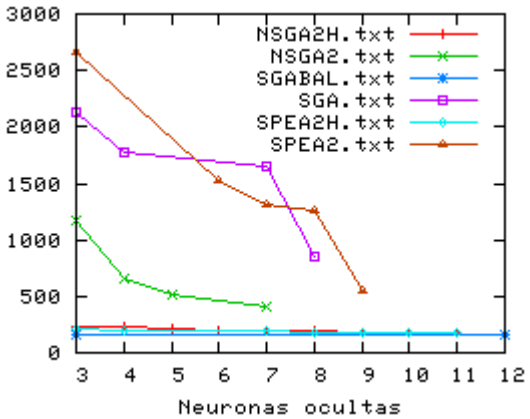

Figura 5.4. Fronteras de Pareto calculadas para redes de Jordan

Las figuras 5.2 a 5.4 muestran los resultados de la comparación de las fronteras de Pareto calculadas mediante las diferentes propuestas multiobjetivo, junto con las fronteras calculadas mediante el método de ensayo y error. En ellas se puede observar que, para los métodos multiobjetivo clásicos, la prueba de ensayo y error puede compararseen términos de frontera de Pareto a la obtenida mediante SPEA-II. Sin embargo, el procedimiento NSGA-II supera ambos métodos con notable diferencia, ya que las soluciones obtenidas por esteúltimo dominan a las demás.

Con respecto a los métodos multiobjetivo híbridos, la búsqueda multiobjetivo por ensayo y error puede equipararse a la realizada mediante SPEA2H, mejorando a los resultados proporcionados por NSGA2H. Además, la frontera obtenida por SPEA2H engloba una diversidad de soluciones mayor, considerando un número superior de redes con neuronas ocultas. De este modo, podemos concluir que la búsqueda multiobjetivo híbridaSPEA2H es una opción prometedora, a tener en cuenta para entrenar y optimizar RNRD.

2.2. Subproblema CATS2.

La tabla 5.2 muestra los resultados obtenidos por los métodos desarrollados en los capítulos 2 y 3, junto con el test de equivalencia estadística de Kruskal-Wallis.

Tabla 5.2: Resultados del error medio por los algoritmos híbridos en contraposición a los modelos genéticos clásicos y los algoritmos BPTT y RTRL, y equivalencia estadística

Algoritmos	Tiempo	MSE Mediano (trn)	MSE Mediano (tst)	pValue
SGA-BAL/PE/EQ/BLX/DES FRNN	*8.27*	*148.821*	*149.044*	*1 (x)*
CHC-BAL FRNN	*10.43*	*148.880*	*149.172*	*0.9234 (x)*
SS FRNN	*8.37*	*150.225*	*150.589*	*0.1316 (x)*

BFGS FRNN	*20.00*	*152.101*	*152.125*	*1.306e-07 (-)*
BFGS Jordan	*10.53*	*153.750*	*153.907*	*5.228e-11 (-)*
SGA-BAL/PA/SO/ARI/DES Elman	*17.67*	*154.962*	*155.166*	*0.007129 (-)*
BFGS Elman	*13.33*	*155.768*	*155.929*	*0.5946 (x)*
SGA-BAL/PE/SO/BLX/ALE Jordan	*15.73*	*156.858*	*157.690*	*0.005961 (-)*
CHC-BAL Elman	*22.57*	*159.241*	*159.097*	*0.5346 (x)*
SS Elman	*15.60*	*158.847*	*159.137*	*0.442 (x)*
GGA/PA/TO/ARI/ALE FRNN	*5.43*	*162.655*	*162.613*	*0.7007 (x)*
CHC-BAL Jordan	*20.50*	*163.506*	*164.825*	*0.04594 (-)*
MGA/PE/TO/BLX/ALE FRNN	*5.60*	*168.909*	*169.238*	*0.9176 (x)*
Clearing/BLX FRNN	*6.00*	*175.561*	*176.244*	*0.2939 (x)*
SS Jordan	*13.77*	*196.038*	*200.479*	*3.057e-05 (-)*
MGA/PE/LI/BLX/ALE Elman	*6.33*	*259.107*	*257.751*	*2.864e-05 (-)*
SGA/PA/LI/ARI/ALE FRNN	*5.50*	*305.670*	*305.665*	*0.1278 (x)*
Clearing/BLX Elman	*6.47*	*402.288*	*406.071*	*0.0001732 (-)*
GGA/PA/TO/HEU/ALE Elman	*5.73*	*545.356*	*547.414*	*4.216e-05 (-)*
CHC/BLX Elman	*5.87*	*757.920*	*766.382*	*0.005202 (-)*
SGA/PA/LI/HEU/ALE Elman	*6.03*	*1172.150*	*1165.984*	*0.0006373 (-)*

MGA/PE/LI/BLX/ALE Jordan	*4.50*	*1639.133*	*1621.425*	*0.0001837 (-)*
Clearing/BLX Jordan	*4.43*	*1711.522*	*1695.789*	*0.4077 (x)*
GGA/PA/TO/BLX/DES Jordan	*4.13*	*2704.839*	*2662.358*	*2.714e-08 (-)*
CHC/BLX FRNN	*4.27*	*3992.292*	*3920.093*	*1.534e-07 (-)*
CHC/BLX Jordan	*4.20*	*3992.292*	*3920.093*	*1 (x)*
SGA/PE/LI/HEU/ALE Jordan	*4.27*	*5699.664*	*5587.805*	*1.111e-07 (-)*
RTRL Elman	*20.00*	*46655.340*	*46514.930*	*2.872e-11 (-)*
RTRL Jordan	*20.00*	*91028.330*	*91449.550*	*5.317e-10 (-)*
RTRL FRNN	*20.00*	*115400.800*	*115560.200*	*0.9882 (x)*
BPTT Jordan	*24.50*	*316550.000*	*319743.400*	*1 (x)*
BPTT Elman	*24.33*	*593478.000*	*600548.400*	*0.02554 (-)*

La notación utilizada en la tabla 5.2 es la misma que para la tabla 5.1. Las conclusiones que podemos obtener de la misma se resumen a continuación:

- **Red que ha proporcionado mejores resultados:** red recurrente completa
- **Mejor algoritmo de entrenamiento para la red Completa** SGA-BAL. CHC-BAL y Scatter Search
- **Mejor algoritmo de entrenamiento para la red de Elman** SGA-BAL y BFGS
- **Mejor algoritmo de entrenamiento para la red de Jordan** BFGS

A partir de las conclusiones anteriores, podemos observar una gran similitud con los resultados obtenidos anteriormente para el problema *CATS1*. El algoritmo híbrido que mejores resultados ha proporcionado para el problema *CATS2* es SGA-BAL, aunque BFGS también ha tenido un comportamiento adecuado en las redes de Elman y de Jordan, siendo en este último modelo el mejor algoritmo de entrenamiento.

La Figura 5.5 muestra el ajuste y la predicción realizados porla mejor solución obtenida al entrenar una red neuronal recurrente completa, mediante el algoritmo SGA-BAL. Seguidamente, las Figuras 5.6 a 5.8 muestran los resultados de la comparación de las fronteras de Pareto calculadas mediante cada algoritmo multiobjetivo, frente a las fronteras calculadas mediante el método de ensayo y error.

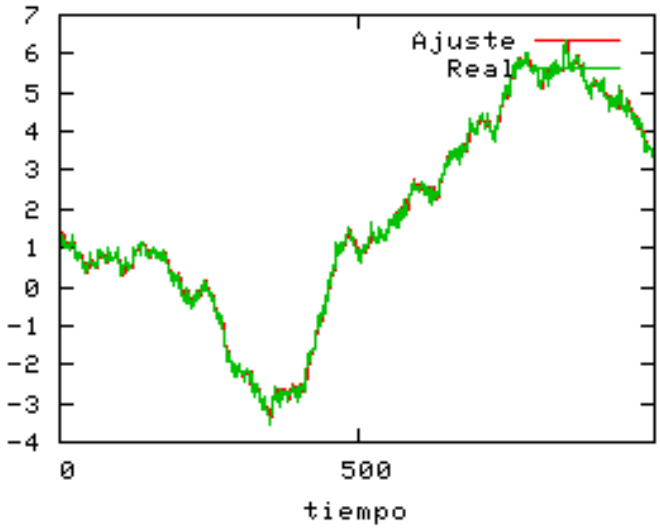

Figura 5.5 Ajuste y predicción realizado en el problema CATS2

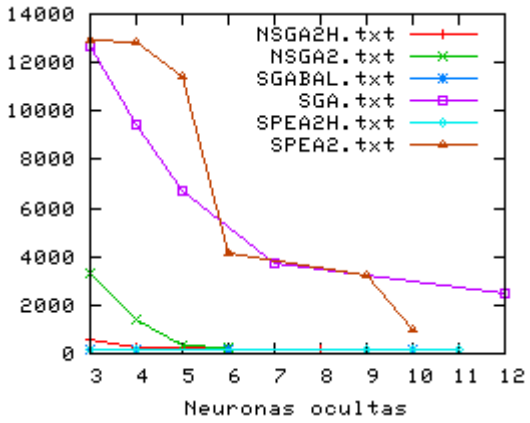

Figura 5.6. Fronteras de Pareto calculadas para redes de Elman

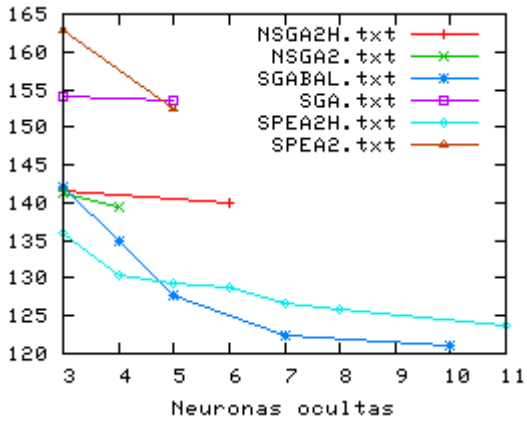

Figura 5.7. Fronteras de Pareto calculadas para redes Completas

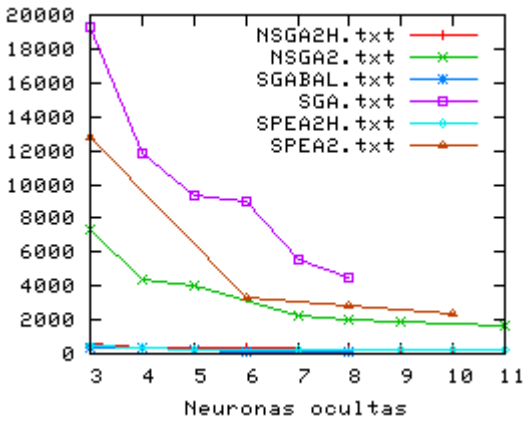

Figura 5.8. Fronteras de Pareto calculadas para redes de Jordan

La frontera de Pareto calculada manualmente mediante la prueba de ensayo y error puede compararse a la obtenida mediante SPEA-II. Mientras que el método de ensayo y error obtiene mejores resultados que SPEA-II para un número reducido de neuronas ocultas, el método multiobjetivo mejora los resultados del procedimiento de ensayo y error cuando el número de neuronas ocultas es elevado. No obstante, el procedimiento NSGA-II supera ambos métodos (SPEA-II y ensayo y error) con notable diferencia, ya que las soluciones obtenidas por este último dominan a las demás.

Con respecto a los métodos multiobjetivo híbridos, la búsqueda por ensayo y error puede equipararse a la realizada mediante SPEA2H, mejorando a los resultados proporcionados por NSGA2H. Los motivos de este comportamiento pueden deducirse a partir de las conclusiones de los capítulos 3 y 4: SPEA2H utiliza un esquema evolutivo estacionario en el que únicamente se generan dos nuevas soluciones en cada iteración, mientras que NSGA2H crea tantos descendientes como sea el tamaño de la población. Debido al número de evaluaciones realizadas para cada nuevo individuo en la búsqueda local, NSGA2H itera un total de generaciones muy inferior al de SPEA2H.

El efecto final es que los operadores evolutivos tienen una menor influencia en el esquema de NSGA2H, aproximándose a una búsqueda multiarranque con el algoritmo de búsqueda local. De este modo, SPEA2H puede explorar y explotar de una forma más adecuada el espacio de búsqueda, produciendo mejores soluciones.

2.3. Subproblema CATS3.

La tabla 5.3 muestra los resultados obtenidos por los métodos desarrollados en los capítulos 2 y 3, junto con el test de equivalencia estadística de Kruskal-Wallis.

Tabla 5.3: Resultados del error medio por los algoritmos híbridos en contraposición a los modelos genéticos clásicos y los algoritmos BPTT y RTRL, y equivalencia estadística

Algoritmos	Tiempo	MSE Mediano (trn)	MSE Mediano (tst)	pValue
SGA-BAL/PE/LI/HEU/ALE FRNN	*8.10*	*133.630*	*141.688*	*1 (x)*
CHC-BAL FRNN	*10.20*	*133.733*	*141.837*	*0.9352 (x)*
SS FRNN	*8.37*	*137.712*	*146.096*	*0.0006036 (-)*
SGA-BAL/PA/TO/BLX/DES Elman	*17.43*	*140.338*	*147.945*	*0.008498 (-)*
BFGS FRNN	*1.83*	*141.251*	*148.248*	*0.2550 (x)*
SS Elman	*19.63*	*140.966*	*149.147*	*0.4508 (x)*
CHC-BAL Elman	*23.87*	*141.368*	*149.515*	*0.2458 (x)*
BFGS Elman	*9.93*	*142.930*	*150.183*	*0.0228 (-)*
BFGS Jordan	*10.70*	*143.451*	*150.353*	*0.442 (x)*
SGA-BAL/PA/LI/ARI/ALE Jordan	*13.20*	*144.660*	*151.506*	*2.204e-05 (-)*
CHC-BAL Jordan	*17.83*	*145.544*	*152.733*	*0.004128 (-)*
SS Jordan	*12.90*	*148.734*	*155.486*	*0.003108 (-)*
MGA/PE/EQ/ARI/ALE FRNN	*5.93*	*150.886*	*158.446*	*0.2675 (x)*

Clearing/BLX FRNN	*5.80*	*154.020*	*160.483*	*0.03988 (-)*
GGA/PA/TO/ARI/ALE FRNN	*5.73*	*156.579*	*162.945*	*0.3750 (x)*
MGA/PE/TO/HEU/ALE Elman	*6.23*	*163.685*	*173.098*	*0.00326 (-)*
Clearing/BLX Elman	*6.07*	*166.873*	*177.609*	*0.2254 (x)*
SGA/PE/TO/ARI/DES FRNN	*5.93*	*183.446*	*190.527*	*0.002439 (-)*
GGA/PA/TO/HEU/DES Elman	*6.03*	*217.297*	*226.799*	*0.001086 (-)*
MGA/PE/LI/HEU/ALE Jordan	*4.10*	*257.148*	*268.036*	*0.009267 (-)*
CHC/BLX Elman	*5.97*	*275.187*	*285.057*	*0.9176 (x)*
SGA/PA/TO/HEU/DES Elman	*6.00*	*309.540*	*324.737*	*0.003940 (-)*
Clearing/BLX Jordan	*4.43*	*326.532*	*336.573*	*0.4247 (x)*
GGA/PA/TO/BLX/DES Jordan	*4.03*	*491.657*	*500.855*	*5.709e-09 (-)*
CHC/BLX FRNN	*4.40*	*702.039*	*716.693*	*0.0003093 (-)*
CHC/BLX Jordan	*4.33*	*702.039*	*716.693*	*1 (x)*
SGA/PA/TO/HEU/DES Jordan	*4.30*	*907.098*	*924.574*	*1.128e-05 (-)*
RTRL Elman	*20.00*	*10493.660*	*10365.730*	*2.872e-11 (-)*
RTRL Jordan	*20.00*	*24915.790*	*24408.530*	*1.055e-08 (-)*
RTRL FRNN	*20.00*	*27334.750*	*26937.010*	*0.3750 (x)*
BPTT Jordan	*20.87*	*1243722.000*	*1241978.000*	*1.485e-08 (-)*

BPTT Elman	*20.00*	*1241633.000*	*1242010.000*	*0.5490 (x)*

Las conclusiones que podemos obtener de la tabla 5.3, similares a las de los problemas anteriores, se resumen a continuación:

- **Red que ha proporcionado mejores resultados:** red recurrente completa
- **Mejor algoritmo de entrenamiento para la red Completa** SGA-BAL y CHC-BAL
- **Mejor algoritmo de entrenamiento para la red de Elman** SGA-BAL, CHC-BAL, Scatter Search y BFGS
- **Mejor algoritmo de entrenamiento para la red de Jordan** BFGS

La Figura 5.9 muestra el ajuste y la predicción realizados porla mejor solución obtenida al entrenar una red neuronal recurrente completa, mediante el algoritmo SGA-BAL.

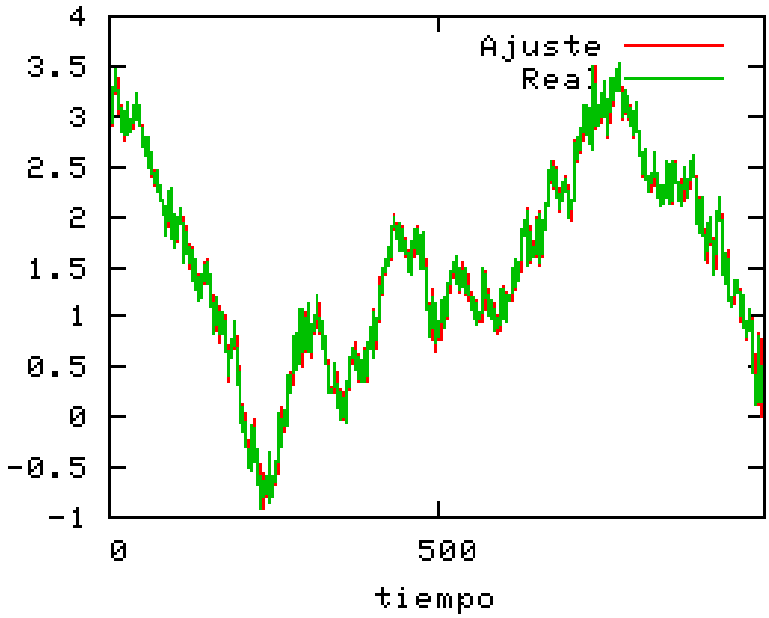

Figura 5.9 Ajuste y predicción realizado en el problema CATS3

Las figuras 5.10 a 5.12 muestran los resultados de la comparación de las fronteras de Pareto calculadas mediante cada algoritmo multiobjetivo, frente con las fronteras calculadas mediante el método de ensayo y error.

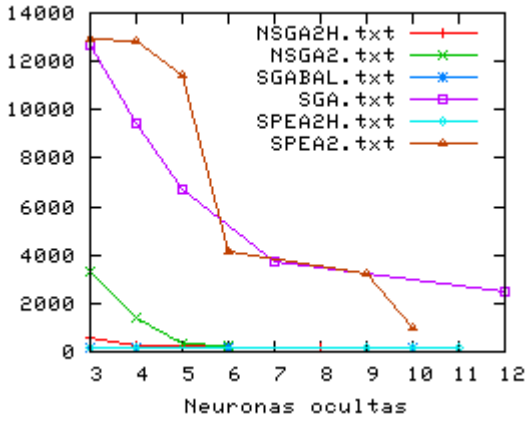

Figura 5.10. Fronteras de Pareto calculadas para redes de Elman

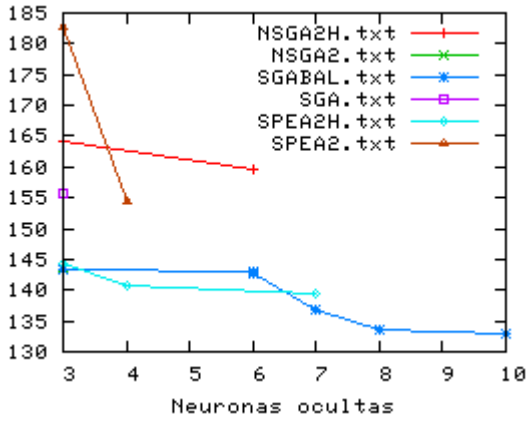

Figura 5.11. Fronteras de Pareto calculadas para redes Completas

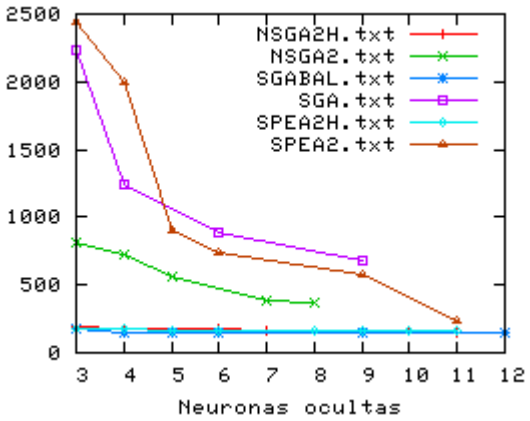

Figura 5.12. Fronteras de Pareto calculadas para redes de Jordan

Las Figuras 5.10 a 5.12 muestran que las soluciones proporcionadas por losalgoritmos multiobjetivo tienen el mismo comportamiento que en los problemas anteriores, frente a las soluciones obtenidas por el método de ensayo y error. Tanto los métodos propuestos híbridos de entrenamiento como de optimización multiobjetivo están demostrando un alta robustez, al producir resultados similares en los diferentes problemas en losque están siendo aplicados.

2.4. Subproblema CATS4.

La tabla 5.4 muestra los resultados obtenidos por los métodos desarrollados en los capítulos 2 y 3, junto con el test de equivalencia estadística de Kruskal-Wallis.

Tabla 5.4: Resultados del error medio por los algoritmos híbridos en contraposición a los modelos genéticos clásicos y los algoritmos BPTT y RTRL, y equivalencia estadística

Algoritmos	Tiempo	MSE Mediano (trn)	MSE Mediano (tst)	pValue
CHC-BAL FRNN	*10.57*	*124.826*	*126.279*	*1 (x)*
SGA-BAL/PE/TO/HEU/DES FRNN	*8.40*	*125.713*	*126.996*	*0.2904 (x)*
SS FRNN	*8.63*	*127.928*	*130.181*	*0.08106 (x)*
BFGS FRNN	*2.27*	*136.560*	*137.163*	*2.331e-09 (-)*
SGA-BAL/PA/SO/HEU/ALE Elman	*18.10*	*135.601*	*139.989*	*0.2805 (x)*
SS Elman	*14.50*	*134.635*	*141.605*	*0.004128 (-)*
CHC-BAL Elman	*22.60*	*136.992*	*143.411*	*0.8476 (x)*
BFGS Elman	*9.40*	*143.686*	*143.812*	*0.3007 (x)*
BFGS Jordan	*7.80*	*145.614*	*146.652*	*0.1242 (x)*
MGA/PE/LI/BLX/ALE FRNN	*5.57*	*146.142*	*151.452*	*0.02109 (-)*
SS Jordan	*9.13*	*145.782*	*151.610*	*0.9882 (x)*
SGA-BAL/PA/SO/BLX/DES Jordan	*10.07*	*146.241*	*153.305*	*0.3077 (x)*
GGA/PA/TO/ARI/ALE FRNN	*5.37*	*152.465*	*155.573*	*0.08367 (x)*
Clearing/BLX FRNN	*5.90*	*153.523*	*158.552*	*0.01949 (-)*
CHC-BAL Jordan	*15.53*	*147.230*	*158.744*	*0.2428 (x)*

MGA/PE/LI/HEU/ALE Elman	*5.83*	*166.317*	*187.923*	*0.0002459 (-)*
SGA/PE/TO/ARI/DES FRNN	*6.10*	*187.081*	*191.722*	*0.8245 (x)*
Clearing/BLX Elman	*6.17*	*169.757*	*214.639*	*0.2089 (x)*
MGA/PE/LI/HEU/ALE Jordan	*4.17*	*161.466*	*230.870*	*0.002822 (-)*
GGA/PA/TO/HEU/ALE Elman	*5.87*	*223.532*	*244.441*	*0.3440 (x)*
CHC/BLX Elman	*5.87*	*220.494*	*261.136*	*0.1103 (x)*
GGA/PA/TO/HEU/ALE Jordan	*4.03*	*191.413*	*276.353*	*0.0173 (-)*
Clearing/BLX Jordan	*4.33*	*175.295*	*289.253*	*0.9646 (x)*
SGA/PA/TO/HEU/ALE Elman	*5.80*	*304.228*	*373.622*	*0.006522 (-)*
SGA/PA/EQ/HEU/DES Jordan	*4.27*	*226.227*	*385.851*	*0.9764 (x)*
CHC/BLX FRNN	*4.53*	*253.387*	*431.089*	*0.1433 (x)*
CHC/BLX Jordan	*4.17*	*253.387*	*431.089*	*0.1433 (x)*
RTRL Elman	*20.00*	*5522.947*	*7484.283*	*2.872e-11 (-)*
BPTT Jordan	*29.83*	*6414.651*	*8088.371*	*0.1510 (x)*
BPTT Elman	*20.83*	*7104.133*	*7233.360*	*0.1060 (x)*
RTRL Jordan	*20.00*	*8623.251*	*10597.320*	*0.004325 (-)*
RTRL FRNN	*20.00*	*14206.700*	*16980.310*	*0.1882 (x)*

Las conclusiones que podemos obtener de la tabla 5.4 se resumen a continuación:

- **Red que ha proporcionado mejores resultados:** red recurrente completa
- **Mejor algoritmo de entrenamiento para la red Completa** CHC-BAL, SGA-BAL y Scatter Search
- **Mejor algoritmo de entrenamiento para la red de Elman:** SGA-BAL
- **Mejor algoritmo de entrenamiento para la red de Jordan:** BFGS

Los métodos que mejores resultados han proporcionado han sido los métodos híbridos desarrollados en el capítulo 3, principalmente cuando son aplicados sobre RNRD completas. En segundo lugar, les siguen los métodos de programación no lineal basados en fórmulas Quasi-Newton, siendo el método BFGS la mejor opción a considerar para entrenar redes de Jordan. La Figura 5.13 muestra el ajuste y la predicción realizados por la mejor solución obtenida al entrenar una red neuronal recurrente completa, mediante el algoritmo SGA-BAL.

Con respecto al problema de la optimización y el entrenamiento de RNRD mediante algoritmos multiobjetivo clásicos e híbridos, las figuras 5.14 a 5.16 muestran las fronteras de Pareto obtenidas por cada una de estas técnicas, para cada modelo de red estudiado. Las figuras incluyen también la frontera de Pareto calculada manualmente mediante el método de ensayo y error, utilizando los algoritmos genéticos estacionario y genéticos estacionario híbrido estudiados en los capítulos 2 y 3 respectivamente. De las figura 5.14 a 5.16 podemos obtener las siguientes conclusiones:

Con respecto a los algoritmos multiobjetivo clásicos, el método NSGA-II es el que mejores resultados obtiene, ya que la frontera de Pareto proporcionada por este método contiene la mayoría de las soluciones no dominadas encontradas (sin considerar los métodos híbridos). La búsqueda por ensayo y error proporciona mejores resultados que el algoritmo SPEA-II cuando el número de neuronas es pequeño, aunque este mejora cuando el número de neuronas ocultas es elevado, proporcionando un conjunto de soluciones no dominadas más amplio.

Considerando los algoritmos multiobjetivo híbridos, podemos observar que el método híbrido SPEA2H supera al algoritmo NSGA2H. No obstante, las soluciones obtenidas por el algoritmo de ensayo y error pueden ser comparadas con las del algoritmo SPEA2. Al contrario de lo que ocurre en los algoritmos clásicos multiobjetivo, SPEA2H produce mejores resultados que el método de ensayo y error cuando el número de neuronas ocultas es reducido, mientras que el método de ensayo y error ha encontrado mejores soluciones con número de neuronas ocultas elevado. De este modo, no podemos asegurar qué método es mejor o peor, ya que ambos proporcionan soluciones no dominadas en proporción similar.

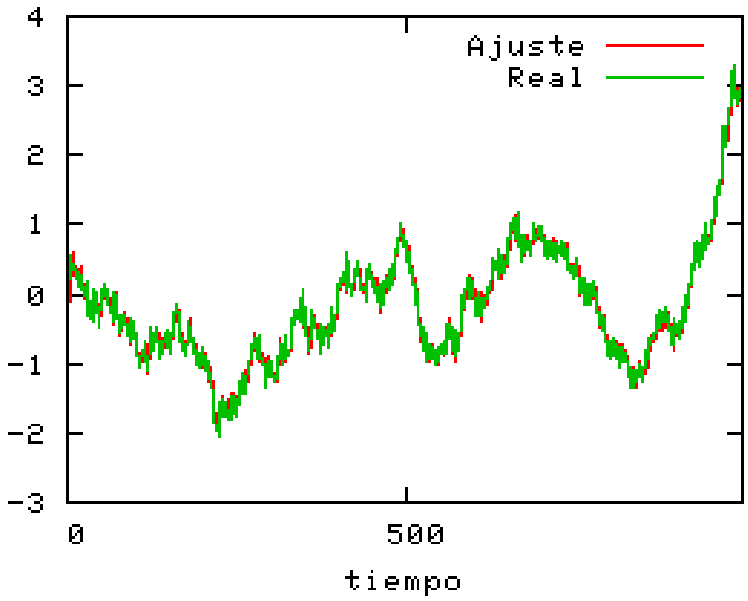

Figura 5.13 Ajuste y predicción realizado en el problema CATS4

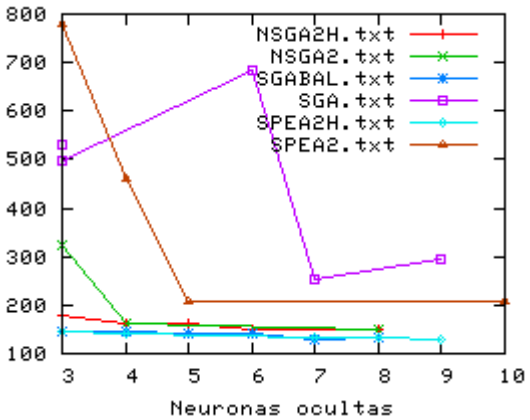

Figura 5.14. Fronteras de Pareto calculadas para redes de Elman

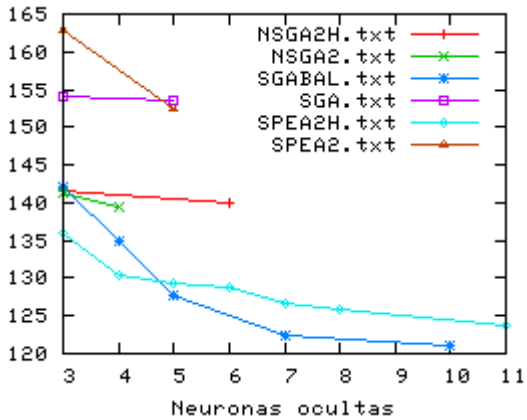

Figura 5.15. Fronteras de Pareto calculadas para redes Completas

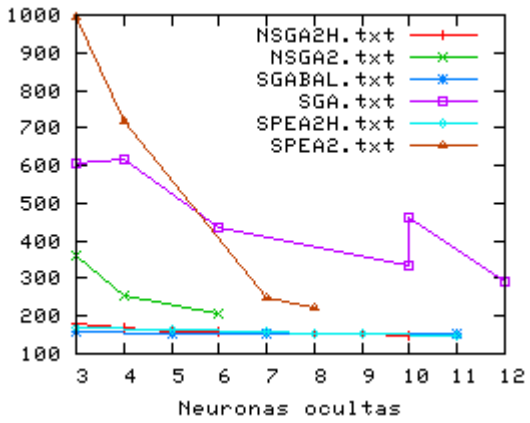

Figura 5.16. Fronteras de Pareto calculadas para redes de Jordan

2.5. Subproblema CATS5.

La tabla 5.5 muestra los resultados obtenidos por los métodos desarrollados en los capítulos 2 y 3, junto con el test de equivalencia estadística de Kruskal-Wallis.

Tabla 5.5: Resultados del error medio por los algoritmos híbridos en contraposición a los modelos genéticos clásicos y los algoritmos BPTT y RTRL, y equivalencia estadística

Algoritmos	Tiempo	MSE Mediano (trn)	MSE Mediano (tst)	pValue
SGA-BAL/PA/TO/BLX/ALE FRNN	*8.27*	*132.378*	*205.223*	*1 (x)*
BFGS Jordan	*12.70*	*138.765*	*205.324*	*0.4161 (x)*
BFGS FRNN	*1.73*	*137.501*	*205.860*	*0.7117 (x)*
CHC-BAL FRNN	*10.10*	*132.371*	*206.373*	*0.9176 (x)*
SGA-BAL/PA/TO/BLX/ALE Jordan	*16.30*	*144.672*	*208.009*	*0.03089 (-)*
BFGS Elman	*12.53*	*139.874*	*208.900*	*0.7117 (x)*

SS FRNN	*8.13*	*136.201*	*210.963*	*0.5642 (x)*
SGA-BAL/PE/LI/BLX/DES Elman	*18.37*	*140.168*	*214.107*	*0.1008 (x)*
CHC-BAL Jordan	*21.10*	*146.530*	*216.313*	*0.03513 (-)*
SS Elman	*15.13*	*140.985*	*218.483*	*0.6152 (x)*
CHC-BAL Elman	*23.53*	*140.558*	*220.156*	*0.4688 (x)*
MGA/PE/EQ/ARI/ALE FRNN	*5.80*	*148.260*	*226.564*	*0.004128 (-)*
SS Jordan	*12.53*	*160.053*	*228.722*	*0.9882 (x)*
Clearing/BLX FRNN	*5.83*	*147.049*	*229.382*	*0.6048 (x)*
GGA/PA/TO/BLX/ALE FRNN	*6.03*	*161.419*	*239.159*	*0.01662 (-)*
MGA/PE/LI/BLX/ALE Elman	*6.00*	*183.376*	*272.943*	*5.786e-05 (-)*
Clearing/BLX Elman	*6.57*	*200.034*	*287.805*	*0.2550 (x)*
SGA/PE/TO/ARI/ALE FRNN	*5.93*	*218.701*	*297.192*	*0.3292 (x)*
GGA/PA/TO/BLX/DES Elman	*6.10*	*310.465*	*415.915*	*3.66e-07 (-)*
CHC/BLX Elman	*6.03*	*331.776*	*444.383*	*0.3366 (x)*
SGA/PA/EQ/HEU/ALE Elman	*6.17*	*455.100*	*624.101*	*9.845e-06 (-)*
Clearing/BLX Jordan	*4.33*	*647.228*	*770.975*	*0.0006036 (-)*
MGA/PE/LI/HEU/ALE Jordan	*4.27*	*666.138*	*775.640*	*0.8941 (x)*
BPTT Elman	*26.60*	*812.265*	*1018.542*	*0.1171 (x)*

GGA/PA/LI/BLX/DES Jordan	*4.20*	*1267.813*	*1389.439*	*2.738e-10 (-)*
BPTT Jordan	*20.63*	*1221.766*	*1427.566*	*0.7007 (x)*
CHC/BLX FRNN	*4.27*	*1374.759*	*1530.040*	*0.2805 (x)*
CHC/BLX Jordan	*4.43*	*1374.759*	*1530.040*	*0.2805 (x)*
SGA/PE/TO/HEU/ALE Jordan	*4.17*	*1827.921*	*1962.023*	*7.316e-07 (-)*
RTRL Elman	*20.00*	*15134.830*	*15445.690*	*2.872e-11 (-)*
RTRL Jordan	*20.00*	*58957.650*	*57934.620*	*4.839e-10 (-)*
RTRL FRNN	*20.00*	*65224.370*	*64189.110*	*0.5346 (x)*

Las conclusiones que podemos obtener de la tabla 5.5 se resumen a continuación:

- **Red que ha proporcionado mejores resultados:** red recurrente completa
- **Mejor algoritmo de entrenamiento para la red Completa** SGA-BAL, CHC-BAL, BFGS
- **Mejor algoritmo de entrenamiento para la red de Elman:** BFGS, Scatter Search y SGA-BAL
- **Mejor algoritmo de entrenamiento para la red de Jordan:** BFGS

Los mejores resultados, para el problema CATS5, han sido obtenidos de forma equivalente para RNRD completas y de Jordan. Mientras que los algoritmos híbridos desarrollados en el capítulo 3 han proporcionado el mejor aprendizaje para las redes recurrentes completas, el método de programación no lineal BFGS lo ha hecho para redes de Jordan. Sin embargo, podemos observar que este método también ha obtenido soluciones equivalentes a los algoritmos híbridos, en redes recurrentes completas.

La red que peores resultados ha obtenido, por tanto, es la red recurrente de Elman. Sin embargo, los mejores resultados que se han proporcionado para este modelo también los ha obtenido el algoritmo genético híbrido estacionario con estrategia baldwiniana, desarrollado en el capítulo 3. La Figura 5.17 muestra el ajuste y la predicción realizados por la mejor solución obtenida al entrenar una red neuronal recurrente completa, mediante el algoritmo SGA-BAL.

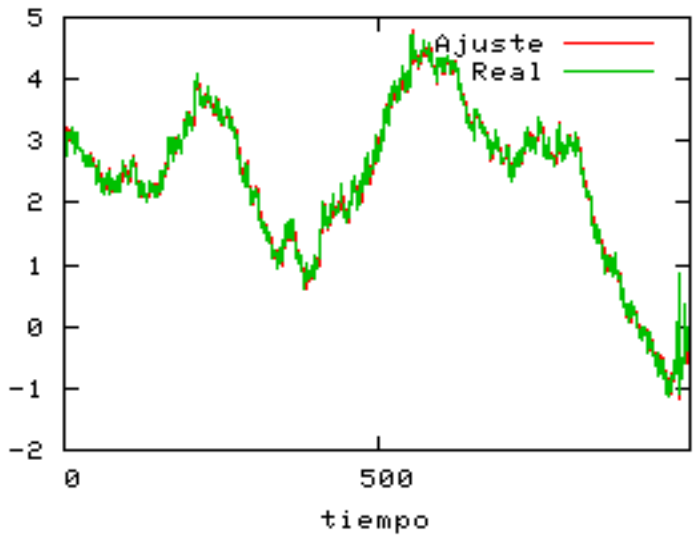

Figura 5.17 Ajuste y predicción realizado en el problema CATS5

Con respecto a la optimización y entrenamiento de las RNRD estudiadas, las figuras 5.18 a 5.20 muestran la frontera de Pareto obtenida por los algoritmos multiobjetivo clásicos e híbridos, desarrollados en el capítulo 4. Las figuras también incluyen la frontera de Pareto calculada manualmente mediante el método de ensayo y error explicado en el apartado 2.1.

Las conclusiones que podemos obtener a partir de las figuras 5.18 a 5.20 son similares a las proporcionadas en los apartados anteriores. Considerando los algoritmos clásicos de optimización, SPEA-II y el método de ensayo y error no pueden ser comparados, ya que ambos contienen un número de soluciones no dominadas muy dispar. Sin embargo, podemos observar que SPEA-II produce mejores soluciones cuando hay un número elevado de neuronas ocultas, mientras que el método de ensayo y error produce mejores soluciones en el caso contrario. Aún así, el método NSGA-II es el que mejores resultados proporciona, ya que sus soluciones dominan a todas las obtenidas mediante SPEA-II y el método de ensayo y error con el algoritmo genético estacionario clásico.

Con respecto a los algoritmos híbridos, podemos observar que los métodosSPEA2H y el método de ensayo y error híbrido son prácticamente equivalentes, porencima de los resultados obtenidos por NSGA2H.

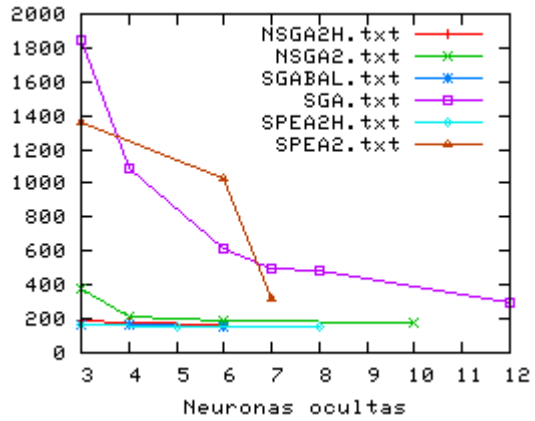

Figura 5.18. Fronteras de Pareto calculadas para redes de Elman

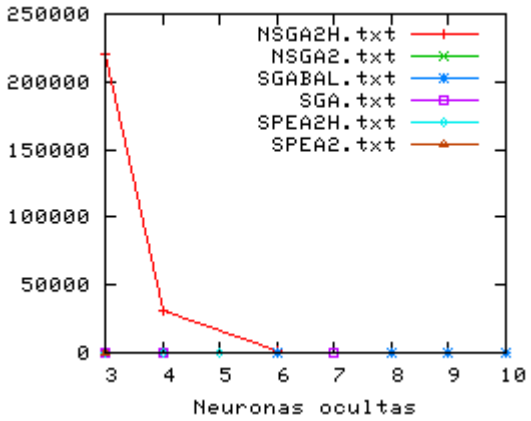

Figura 5.19. Fronteras de Pareto calculadas para redes Completas

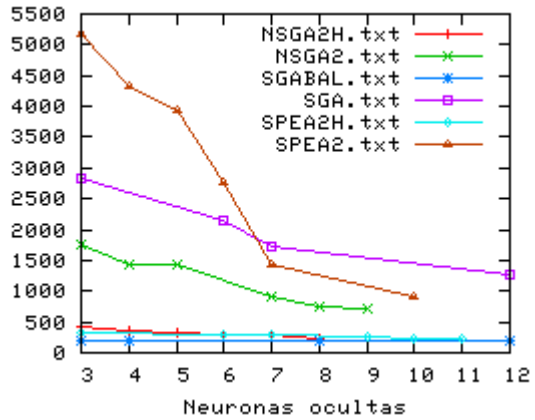

Figura 5.20. Fronteras de Pareto calculadas para redes de Jordan

2.6. Comparación con otros métodos.

En este apartado, comparamos el error del ajuste realizado en la predicción del benchmark CATS con los métodos sometidos a la competición donde se propuso la serie temporal. Existen dos medidas de comparación, E_1 y E_2, calculadas de acuerdo con:

$$E_1 = \frac{\sum_{t=981}^{1000} (y(t) - y'(t))^2}{100} + \frac{\sum_{t=1981}^{2000} (y(t) - y'(t))^2}{100} + \frac{\sum_{t=2981}^{3000} (y(t) - y'(t))^2}{100} + \frac{\sum_{t=3981}^{4000} (y(t) - y'(t))^2}{100} + \frac{\sum_{t=4981}^{5000} (y(t) - y'(t))^2}{100} \quad (5.1)$$

$$E_2 = \frac{\sum_{t=981}^{1000} (y(t) - y'(t))^2}{80} + \frac{\sum_{t=1981}^{2000} (y(t) - y'(t))^2}{80} + \frac{\sum_{t=2981}^{3000} (y(t) - y'(t))^2}{80} + \frac{\sum_{t=3981}^{4000} (y(t) - y'(t))^2}{80} \quad , \quad (5.2)$$

donde $y(t)$ representa al valor de la serie CATS en el instante t, e $y'(t)$ se corresponde con la predicción realizada para el mismo valor.

La razón de uso del criterio de error E_2 es que algunos de los métodos de predicción propuestos no sólo utilizan los valores anteriores de la serie, sino también los posteriores. A continuación, las tablas 5.6 y 5.7 muestran las medidas E_1 y E_2 obtenidas por cada método, incluyendo el algoritmo híbrido estacionario con estrategia baldwiniana desarrollado (aplicado en RNRD completas). La tabla 5.6 está ordenada por la medida E_1, mientras que la tabla 5.7 está ordenada por E_2.

Tabla 5.6: Resultados de los algoritmos de comparación en el problema CATS, ordenados por E_1.

Algoritmo	E_1	E_2
SGA-BALFRNN	***233.354298***	***279.500041***
Kalman Smoother [SAR04]	*408*	*346*
Recurrent Neural Networks [CAI04]	*441*	*402*
Competitive Associative Net [KUR04]	*502*	*418*
Weighted Bidirectional Multi-stream Extended Kalman Filter [HU04]	*530*	*370*
SVCA Model [PAL04]	*577*	*395*
MultiGrid -Based Fuzzy Systems [POM04]	*644*	*542*
D. Quantization Forecasting Method [SIM04]	*653*	*351*
Time -reversal Symmetry Method [VER04]	*660*	*442*
BYY Harmony Learning Based Mixture of Experts Model [CHA04]	*676*	*677*
Ensemble Models [WIC04]	*725*	*222*

Chaotic Neural Networks [KOZ04]	*928*	*762*
E. Block-based Neural Networks [KON04]	*954*	*994*
Time -line Hidden Markov Experts [WAN04]	*1037*	*402*
Fuzzy Inductive Reasoning [CEL04]	*1050*	*278*
Business Forecasting Approach to Mulitlayer Perceptron Modelling [CRO04]	*1156*	*995*
A hierarchical Bayesian Learning Scheme for Autoregressive Neural Networks [ELE04]	*1247*	*1122*
Hybrid Predictor [YEN04]	*1425*	*894*

Tabla 5.7: Resultados de los algoritmos de comparación en el problema CATS, ordenados por E_2.

Algoritmo	**E_1**	**E_2**
Ensemble Models	*725*	*222*
Fuzzy Inductive Reasoning	*1050*	*278*
SGA-BALFRNN	***233.354298***	***279.500041***
Kalman Smoother	*408*	*346*
Double Quantization Forecasting Method	*653*	*351*
Weighted Bidirectional Multi-stream Extended Kalman Filter	*530*	*370*
SVCA Model	*577*	*395*
Recurrent Neural Networks	*441*	*402*
Time -line Hidden Markov Experts	*1037*	*402*
Competitive Associative Net	*502*	*418*
Time -reversal Symmetry Method	*660*	*442*
MultiGrid -Based Fuzzy Systems	*644*	*542*
BYY Harmony Learning Based Mixture of Experts Model	*676*	*677*
Chaotic Neural Networks	*928*	*762*
Hybrid Predictor	*1425*	*894*
Evolvable Block-based Neural Networks	*954*	*994*
Business Forecasting Approach to Mulitlayer Perceptron Modelling	*1156*	*995*
A hierarchical Bayesian Learning Scheme for Autoregressive Neural Networks	*1247*	*1122*

Las tablas 5.6 y 5.7 muestran la eficacia del algoritmo genético híbrido SGA-BAL, para entrenar RNRD completas.

Considerando la medida E_1, el método propuesto ha conseguido superar a las demás técnicas consideradas. En cambio, considerando el criterio E_2, el algoritmo SGA-BAL ha sido superado por las técnicas *Fuzzy Inductive Reasoning* y *Ensemble Models*. Por tanto, podemos concluir que los métodos híbridos propuestos han conseguido entrenar con éxito los modelos de RNRD estudiados. Si nos centramos en el problema CATS, las redes recurrentes completas han resultado ser las más adecuadas para atajar el problema, obteniendo resultados superiores a los existentes con respecto a la medida de error E_1.

Considerando la medida de error E_2, las propuestas realizadas en este documento también han proporcionado resultados bastante prometedores, aunque superadas por las técnicas *Ensemble Models* y *Fuzzy Inductive Reasoning*. No obstante, hay que tener en cuenta que esta medida de error está pensada para los métodos que utilizan tanto los datos anteriores como los posteriores a los valores a predecir, mientras que nuestra propuesta únicamente utiliza los datos anteriores.

3. Aplicación a series de datos de tipo económico.

Esta sección ilustra el funcionamiento de los métodos propuestos, aplicándolos a entrenamiento y optimización de RNRD para predicción de series de datos de tipo económico. Los conjuntos de datos que utilizamos son los siguientes:

- **USPob**. Este conjunto de datos contiene la variación porcentual de la evolución de la población estadounidense, medida mensualmente desde el año 1950 hasta 2004. Utilizamos el 90% de los datos como entrenamiento, y dejamos el resto para la validación. La figura 5.21 ilustra la evolución completa de la serie *USPob*.

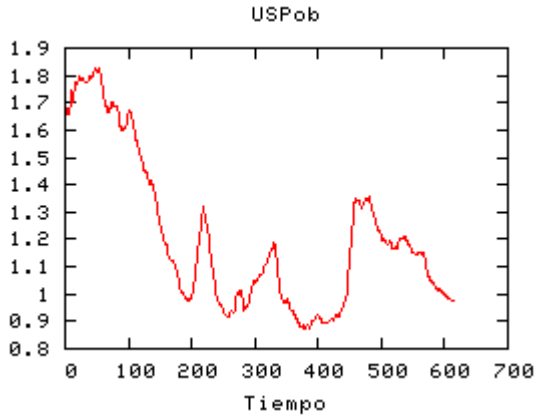

Figura 5.21. Serie USPob completa

- **EurDol**. Este conjunto de datos contiene la evolución de la tasa euro-dólar americano, medida mensualmente desde el año 1994 hasta marzo de 2004. Utilizamos el 90% de los datos para el conjunto de entrenamiento, y el 10% restante como test. La figura 5.22 ilustra la evolución completa de la serie *EurDol*.

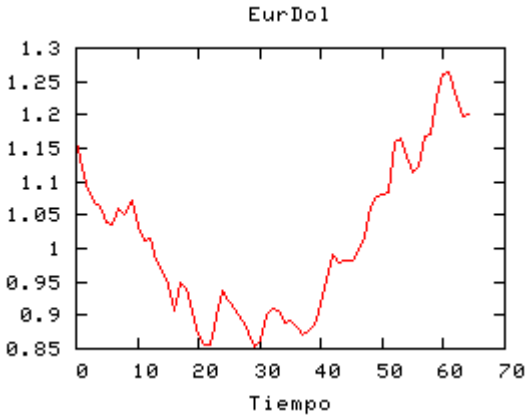

Figura 5.22. Serie EurDol completa

- **PIB_Andalucia.** Este conjunto de datos contiene la evolución del producto interior bruto (PIB) de la comunidad autónoma andaluza, durante los años 1988 a 2002. Utilizaremos los datos correspondientes al intervalo comprendido entre 1988 y 2000 como entrenamiento, mientras que trataremos de predecir el PIB para los años 2001 y 2002.

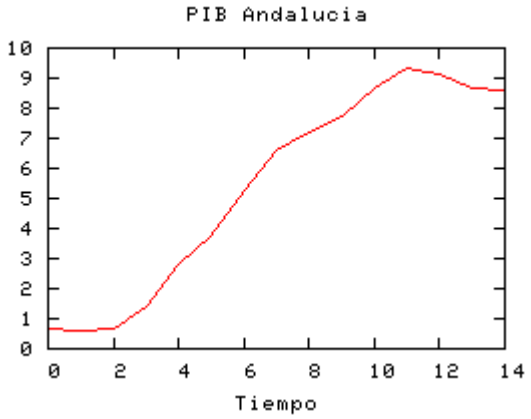

Figura 5.23. Evolución del PIB de Andalucía entre 1988 y 2002

Las series de datos *USPob* y *EurDol* pueden obtenerse de forma gratuita desde la web http://www.economagic.com.

3.1. Conjunto USPob.

Este apartado estudia el comportamiento de los algoritmos de entrenamiento y optimización de RNRD propuestos en los capítulos 2, 3 y 4, para resolver el problema de la predicción de la serie de datos *USPob*. Los experimentos se han realizado siguiendo el mismo esquema utilizado para el problema CATS, durante el desarrollo del documento. La tabla 5.8 muestra un resumen del error MSE medio producido en el entrenamiento y test de 30 ejecuciones de cada tipo algoritmo, junto con el valor del estimador de equivalencia estadística.

Tabla 5.8: Resultados del error medio por los algoritmos híbridos en contraposición a los modelos genéticos clásicos y los algoritmos BPTT y RTRL, y equivalencia estadística

Algoritmos	Tiempo	MSE Mediano (trn)	MSE Mediano (tst)	pValue
SGA-BAL/PA/LI/ARI/ALE Jordan	*2.37*	*0.000*	*0.000*	*1 (x)*
CHC-BAL Jordan	*3.20*	*0.000*	*0.000*	*0.8302 (x)*

MGA/PE/LI/HEU/ALE Jordan	*2.13*	*0.001*	*0.001*	*0.02559 (-)*
SGABAL/PA/LI/BLX/DES FRNN	*3.43*	*0.013*	*0.010*	*9.193e-06 (-)*
CHCBAL FRNN	*3.57*	*0.014*	*0.011*	*0.9176 (x)*
BFGS FRNN	*0.43*	*0.015*	*0.012*	*0.3440 (x)*
Clearing/BLX Jordan	*2.20*	*0.025*	*0.022*	*0.1882 (x)*
CHCBAL Elman	*5.00*	*0.028*	*0.023*	*0.0001837 (-)*
SGABAL/PA/SO/BLX/ALE Elman	*4.73*	*0.034*	*0.028*	*0.4375 (x)*
GGA/PA/TO/BLX/DES Jordan	*2.07*	*0.051*	*0.044*	*0.0005121 (-)*
CHC/BLX FRNN	*2.20*	*0.050*	*0.046*	*0.8016 (x)*
CHC/BLX Jordan	*2.20*	*0.050*	*0.046*	*0.8016 (x)*
BFGS Elman	*0.53*	*0.061*	*0.051*	*0.1785 (x)*
SGA/PE/TO/HEU/DES Jordan	*2.20*	*0.076*	*0.070*	*0.1515 (x)*
BPTT Jordan	*9.47*	*0.102*	*0.092*	*2.985e-06 (-)*
BPTT Elman	*12.77*	*0.169*	*0.144*	*5.657e-06 (-)*
Clearing/BLX Elman	*3.17*	*0.162*	*0.147*	*0.1882 (x)*
Clearing/BLX FRNN	*3.10*	*0.280*	*0.233*	*0.3440 (x)*
CHC/BLX Elman	*3.13*	*0.325*	*0.274*	*0.006522 (-)*
GGA/PA/TO/HEU/DES Elman	*2.97*	*0.402*	*0.326*	*0.7901 (x)*

SGA/PE/TO/ARI/ALE FRNN	*2.80*	*0.437*	*0.383*	*0.1316 (x)*
MGA/PE/TO/BLX/ALE Elman	*2.90*	*0.503*	*0.435*	*0.01949 (-)*
GGA/PA/TO/BLX/DES FRNN	*2.83*	*0.702*	*0.598*	*0.3077 (x)*
SGA/PA/TO/HEU/DES Elman	*3.07*	*0.874*	*0.736*	*0.4161 (x)*
MGA/PE/LI/BLX/ALE FRNN	*2.90*	*0.974*	*0.846*	*0.1602 (x)*
RTRL Elman	*91.07*	*10.864*	*10.678*	*6.812e-09 (-)*
RTRL Jordan	*0.00*	*3100.172*	*3098.633*	*2.260e-10 (-)*
RTRL FRNN	*111.67*	*5754.160*	*5756.692*	*0.3147 (x)*

Las conclusiones que podemos obtener de la tabla 5.8 se resumen a continuación:

- **Red que ha proporcionado mejores resultados:** red de Jordan
- **Mejor algoritmo de entrenamiento para la red Completa** SGA-BAL, CHC-BAL y BFGS
- **Mejor algoritmo de entrenamiento para la red de Elman:** SGA-BAL y CHC-BAL
- **Mejor algoritmo de entrenamiento para la red de Jordan:** SGA-BAL, CHC-BAL

Las conclusiones acerca del comportamiento de los diferentes algoritmos de entrenamiento de RNRD, aplicados al problema *USPob*, se asemejan a las obtenidas para el problema *CATS* en tanto en cuanto los mejores resultados los han obtenido los algoritmos híbridos propuestos en el capítulo 3. No obstante, en este problema la red que mejor comportamiento ha conseguido aprender ha sido la red de Jordan. Cabe también destacar que algunos métodos evolutivos no híbridos, como el algoritmo genético mixto, ha conseguido también mejores resultados al entrenar redes de Jordan que los métodos híbridos para entrenar redes Completas y de Elman. La conclusión que podemos obtener de este comportamiento es

que la estructura topológica de la red de Jordan es adecuada pararesolver el problema *USPob*, frente a las otras topologías.

La Figura 5.24 muestra el ajuste y la predicción realizados por lared de Jordan entrenada mediante el método SGA-BAL.

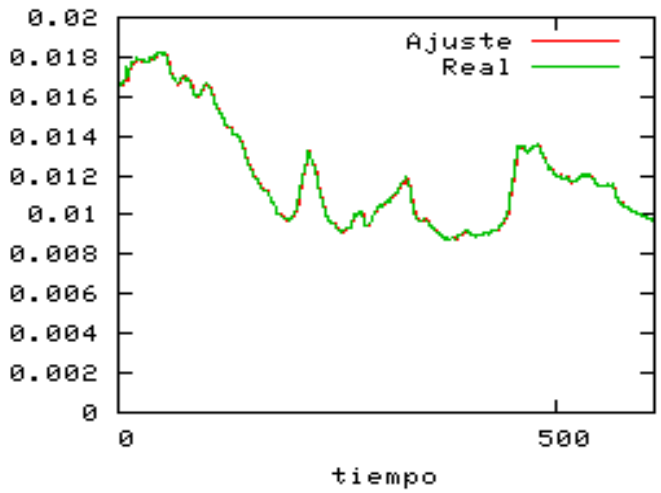

Figura 5.24. Ajuste y predicción realizamos para la serie USPob

Con respecto a los algoritmos multiobjetivo, las tablas 5.9 a 5.11 muestran la frontera de Pareto obtenida tras 30 ejecuciones de cada uno de los modelos considerados, para las redes de Elman, completa y de Jordan, respectivamente. Para comprobar laeficacia de cada uno de estos algoritmos hemos realizado 30 ejecuciones mediante el método de ensayoy error, para encontrar la frontera de Pareto manualmente mediante los algoritmos genéticos estacionarios e híbridos. Las figuras 5.25 a 5.27 muestran la comparación de las diferentes fronteras de Pareto obtenidas.

Tabla 5.9: Frontera de Pareto encontrada mediante los algoritmos multobjetivo clásicos e híbridos, para redes de Elman

Algoritmos	**Neuronas**	**MSE Mediano (trn)**	**MSE Mediano (tst)**
SPEA2	3	1.534052	0.082034
	4	62.570040	0.018333
NSGA2	3	0.218041	0.017707
	7	0.025866	0.006876
SPEA2H	3	0.002181	0.000421

	4	0.001776	0.000266
NSGA2H	3	0.006295	0.000357

Tabla 5.10: Frontera de Pareto encontrada mediante los algoritmos multiobjetivo clásicos e híbridos, para redes Completas

Algoritmos	**Neuronas**	**MSE Mediano (trn)**	**MSE Mediano (tst)**
SPEA2	3	8.928606	0.799600
	4	30.454840	0.493104
NSGA2	3	0.047464	0.002557
SPEA2H	3	0.001933	0.000137
NSGA2H	3	0.009580	0.000077

Tabla 5.11: Frontera de Pareto encontrada mediante los algoritmosmultiobjetivo clásicos e híbridos, para redes de Jordan

Algoritmos	**Neuronas**	**MSE Mediano (trn)**	**MSE Mediano (tst)**
SPEA2	3	0.112015	0.076308
	4	0.228044	0.075672
	5	1.548143	0.048093
	6	3.599553	0.034992
	8	0.281190	0.015413
NSGA2	3	0.002249	0.000072
SPEA2H	3	0.008409	0.000142
	4	0.004773	0.000091
	9	0.000282	0.000091
NSGA2H	3	0.000120	0.000092

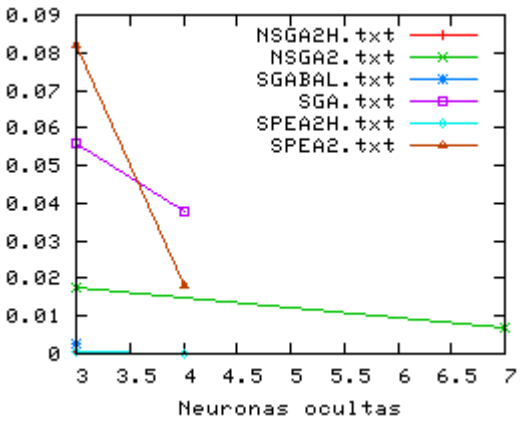

Figura 5.25. Fronteras de Pareto calculadas para redes de Elman

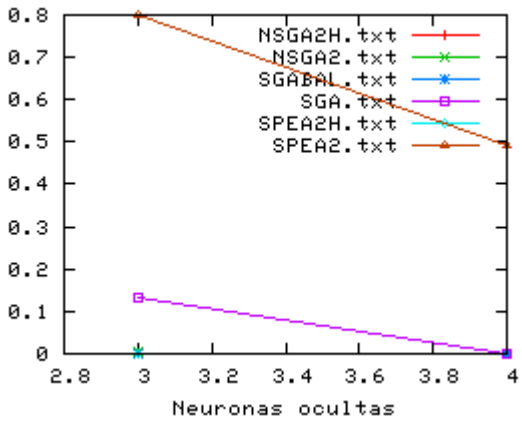

Figura 5.26. Fronteras de Pareto calculadas para redes Completas

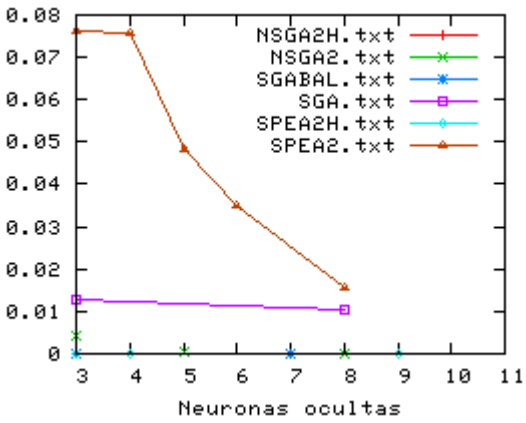

Figura 5.27. Fronteras de Pareto calculadas para redes de Jordan

El algoritmo SPEA-II ha sido el que peores resultados ha proporcionado, en comparación con los algoritmos NSGA-II y de ensayo y error. Sin embargo, al utilizar el algoritmo híbrido SPEA2H, los resultados que se obtienen son muy superiores a los de NSGA2H. Las razones son similares a las expuestas para el problema CATS: El número de iteraciones de SPEA2H es superior al de NSGA2H, de modo que el proceso evolutivo se ve poco afectado por los operadores de evolución en este último método.

3.2. Conjunto EurDol.

Este apartado estudia el comportamiento de los algoritmos de entrenamiento y optimización de RNRD propuestos en los capítulos 2, 3 y 4, para resolver el problema *EurDol*. La tabla 5.12 muestra la comparación de los mejores resultadosde los algoritmos utilizados para entrenar los diferentes modelos de RNRD en este problema. En este caso, la red que mejor comportamiento presenta es la red de Jordan, entrenada mediante el algoritmo híbrido SGA-BAL o, equivalentemente, con el algoritmo CHC-BAL. Por otra parte, la red que peores resultados ha conseguido aprender ha sido de nuevo la red de Elman.

Sin embargo, el aspecto común a todas las redes es que los algoritmos de entrenamiento que mejores resultados han proporcionado son los métodos híbridos propuestos enel capítulo 3, especialmente cuando se utiliza la estrategia baldwiniana. LaFigura 5.28 muestra los resultados del ajuste y la predicción obtenidos por la red de Jordan, entrenada con el algoritmo híbrido estacionario con estrategia baldwiniana.

Tabla 5.12: Resultados del error medio por los algoritmos híbridos en contraposición a los modelos genéticos clásicos y los algoritmos BPTT y RTRL, y equivalencia estadística

Algoritmos	Tiempo	MSE Mediano (trn)	MSE Mediano (tst)	pValue
SGA-BAL/PA/LI/ARI/ALE Jordan	*0.30*	*0.001*	*0.001*	*1 (x)*
CHC-BAL Jordan	*0.30*	*0.001*	*0.001*	*0.865 (x)*
MGA/PE/LI/HEU/ALE Jordan	*0.23*	*0.003*	*0.006*	*3.802e-08 (-)*
Clearing/BLX Jordan	*0.23*	*0.006*	*0.013*	*0.04133 (-)*
BFGS FRNN	*0.07*	*0.012*	*0.014*	*0.7338 (x)*
GGA/PA/TO/BLX/DES Jordan	*0.20*	*0.006*	*0.014*	*0.6681 (x)*
CHC-BAL Elman	*0.57*	*0.012*	*0.019*	*0.07852 (x)*

BFGS Elman	*0.07*	*0.010*	*0.019*	*0.03711 (-)*
SGA-BAL/PA/SO/BLX/ALE Elman	*0.53*	*0.015*	*0.023*	*0.3366 (x)*
CHC-BAL FRNN	*0.37*	*0.021*	*0.027*	*0.06249 (x)*
SGA-BAL/PA/LI/BLX/DES FRNN	*0.43*	*0.025*	*0.030*	*0.1206 (x)*
CHC/BLX FRNN	*0.23*	*0.021*	*0.031*	*0.02028 (-)*
CHC/BLX Jordan	*0.23*	*0.021*	*0.031*	*1 (x)*
SGA/PE/TO/HEU/DES Jordan	*0.23*	*0.026*	*0.035*	*0.4871 (x)*
Clearing/BLX Elman	*0.43*	*0.079*	*0.086*	*6.979e-05 (-)*
Clearing/BLX FRNN	*0.43*	*0.401*	*0.341*	*0.0001286 (-)*
BPTT Elman	*1.10*	*0.647*	*0.537*	*0.4598 (x)*
CHC/BLX Elman	*0.40*	*0.747*	*0.620*	*0.004325 (-)*
GGA/PA/TO/HEU/DES Elman	*0.33*	*0.707*	*0.629*	*0.0646 (x)*
SGA/PE/TO/ARI/ALE FRNN	*0.33*	*1.181*	*0.987*	*0.08367 (x)*
MGA/PE/LI/BLX/ALE FRNN	*0.37*	*1.065*	*0.998*	*0.5742 (x)*
GGA/PA/TO/BLX/DES FRNN	*0.33*	*1.386*	*1.272*	*0.09775 (x)*
SGA/PA/TO/HEU/DES Elman	*0.30*	*1.918*	*1.583*	*0.2805 (x)*
MGA/PE/TO/BLX/ALE Elman	*0.33*	*1.793*	*1.712*	*0.01596 (-)*
RTRL Elman	*0.87*	*11.237*	*11.244*	*1.15e-06 (-)*

RTRL Jordan	*1.01*	*3077.511*	*3081.641*	*2.488e-10 (-)*
RTRL FRNN	*0.93*	*5768.767*	*5767.252*	*0.3219 (x)*

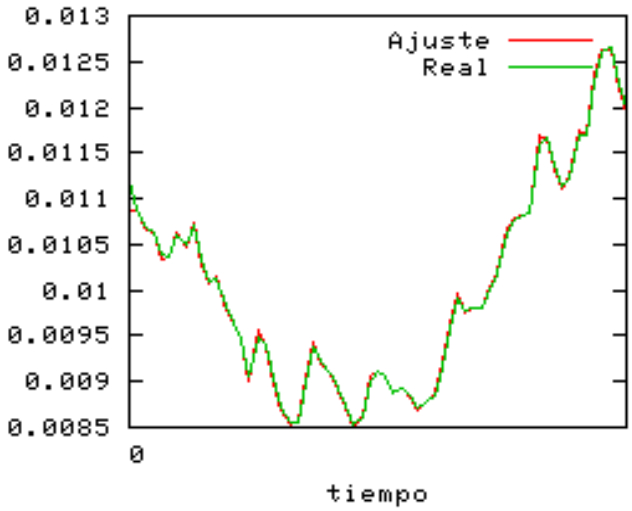

Figura 5.28. Ajuste y predicción realizamos para la serie EurDol

Las conclusiones que podemos obtener de la tabla 5.9 se resumen a continuación:

- **Red que ha proporcionado mejores resultados:** red de Jordan
- **Mejor algoritmo de entrenamiento para la red Completa** BFGS
- **Mejor algoritmo de entrenamiento para la red de Elman:** CHC-BAL
- **Mejor algoritmo de entrenamiento para la red de Jordan:** SGA-BAL, CHC-BAL

Con respecto a las propuestas de algoritmos de entrenamiento y optimización multiobjetivo, las tablas 5.13 a 5.15 muestran los resultados de su aplicación al problema *EurDol*. Los algoritmos híbridos superan en eficacia a los algoritmos clásicos que hemos estudiado. Del mismo modo, mientras que el algoritmo NSGA-II consigue una frontera de Pareto de mayor calidad que la obtenida mediante SPEA-II, el caso contrario se da al contrastar los resultados de SPEA2H y NSGA2H. Las figuras 5.29 a 5.31 muestran esta situación, ilustrando la frontera de Pareto calculada por cada algoritmo tras 30 ejecuciones.

Tabla 5.13: Frontera de Pareto encontrada mediante los algoritmos multiobjetivo clásicos e híbridos, para redes de Elman

Algoritmos	Neuronas	MSE Mediano (trn)	MSE Mediano (tst)
SPEA2	3	7.468975	0.034557
	4	3.346478	0.016793
NSGA2	3	0.054332	0.007595
	4	0.014273	0.003617
SPEA2H	3	0.015380	0.001397
	4	0.009712	0.001059
NSGA2H	3	0.013974	0.008592

Tabla 5.14: Frontera de Pareto encontrada mediante los algoritmos multiobjetivo clásicos e híbridos, para redes Completas

Algoritmos	Neuronas	MSE Mediano (trn)	MSE Mediano (tst)
SPEA2	3	21.804560	2.702554
	4	12.411660	0.602488
NSGA2	3	0.036809	0.005227
SPEA2H	3	0.006921	0.000849
	6	0.006294	0.000846
NSGA2H	3	0.000871	0.000642

Tabla 5.15: Frontera de Pareto encontrada mediante los algoritmos multiobjetivo clásicos e híbridos, para redes de Jordan

Algoritmos	Neuronas	MSE Mediano (trn)	MSE Mediano (tst)
SPEA2	3	0.384095	0.012746
	4	0.362666	0.001647
	11	0.041145	0.001394
NSGA2	3	0.003788	0.003788
	5	0.005101	0.000726
	8	0.012109	0.000617
SPEA2H	3	0.001133	0.000670

	5	0.001999	0.000604
NSGA2H	3	0.002091	0.000623
	4	0.006917	0.000607
	5	0.004214	0.000604

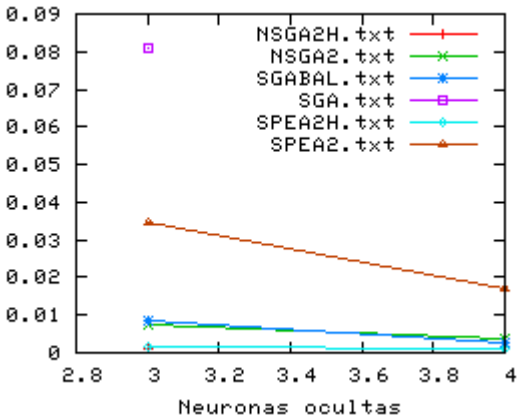

Figura 5.29. Fronteras de Pareto calculadas para redes de Elman

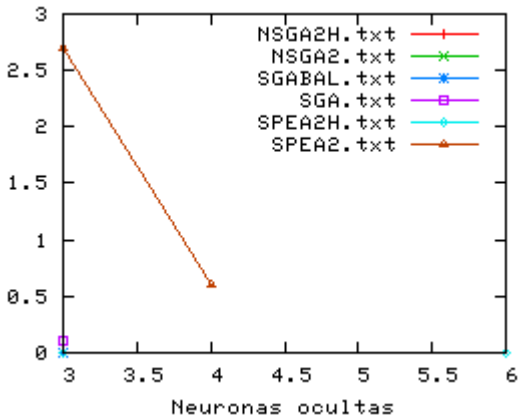

Figura 5.30. Fronteras de Pareto calculadas para redes Completas

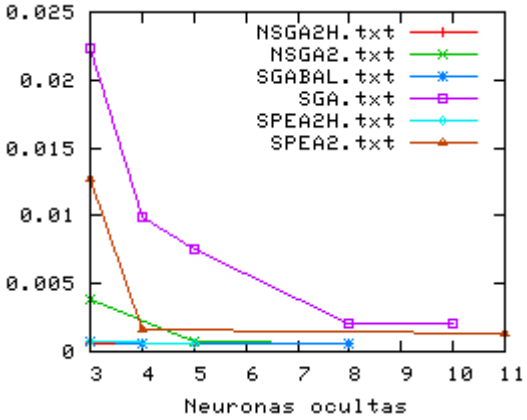

Figura 5.31. Fronteras de Pareto calculadas para redes de Jordan

3.3. Conjunto PIB_Andalucía.

Este apartado estudia el comportamiento de los algoritmos de entrenamiento y optimización de RNRD propuestos en los capítulos 2, 3 y 4, en el problema de predicción de la serie de datos *PIB_Andalucía*. A continuación, la tabla 5.16 muestra los resultados del error

MSE obtenido en los conjuntos de entrenamiento y test, junto con el valor de la equivalencia entre los algoritmos.

Tabla 5.16: Resultados del error medio por los algoritmos híbridos en contraposición a los modelos genéticos clásicos y los algoritmos BPTT y RTRL, y equivalencia estadística

Algoritmos	**Tiempo**	**MSE Mediano (trn)**	**MSE Mediano (tst)**	**pValue**
SGA-BAL/PA/LI/ARI/ALE Jordan	*0.07*	*0.157*	*0.167*	*1 (x)*
CHC-BAL Jordan	*0.10*	*0.157*	*0.168*	*0.9293 (x)*
MGA/PE/LI/HEU/ALE Jordan	*0.07*	*0.157*	*0.171*	*0.03988 (-)*
SGA-BAL/PA/LI/BLX/DES FRNN	*0.13*	*0.152*	*0.185*	*0.7788 (x)*
CHC-BAL FRNN	*0.17*	*0.150*	*0.202*	*0.3671 (x)*
BFGS FRNN	*0.03*	*0.143*	*0.206*	*0.1515 (x)*
Clearing/BLX Jordan	*0.07*	*0.236*	*0.239*	*0.0003666 (-)*
SGA-BAL/PA/SO/BLX/ALE Elman	*0.17*	*0.229*	*0.269*	*0.8592 (x)*
CHC-BAL Elman	*0.20*	*0.265*	*0.284*	*0.4508 (x)*
CHC/BLX FRNN	*0.00*	*1.037*	*0.970*	*1.627e-08 (-)*
/CHC/BLX Jordan	*0.07*	*1.037*	*0.970*	*1 (x)*
SGA/PE/TO/HEU/DES Jordan	*0.07*	*1.018*	*1.000*	*0.3912 (x)*
GGA/PA/TO/BLX/DES Jordan	*0.03*	*1.042*	*1.032*	*0.6681 (x)*
BFGS Elman	*0.03*	*1.184*	*1.057*	*0.5444 (x)*

Clearing/BLX Elman	*0.17*	*1.607*	*1.499*	*0.006522 (-)*
Clearing/BLX FRNN	*0.20*	*2.022*	*1.742*	*0.8941 (x)*
MGA/PE/TO/BLX/ALE Elman	*0.10*	*3.363*	*2.910*	*0.009674 (-)*
GGA/PA/TO/HEU/DES Elman	*0.10*	*4.386*	*3.774*	*0.4508 (x)*
CHC/BLX Elman	*0.13*	*5.922*	*5.118*	*0.03711 (-)*
GGA/PA/TO/BLX/DES FRNN	*0.10*	*6.583*	*5.654*	*0.6898 (x)*
MGA/PE/LI/BLX/ALE FRNN	*0.10*	*7.232*	*7.079*	*0.6574 (x)*
SGA/PA/TO/HEU/DES Elman	*0.10*	*8.184*	*7.245*	*0.2805 (x)*
SGA/PE/TO/ARI/ALE FRNN	*0.10*	*9.508*	*9.186*	*0.008875 (-)*
BPTT Jordan	*0.20*	*17.982*	*20.410*	*1.627e-08 (-)*
BPTT Elman	*0.20*	*21.839*	*21.105*	*0.87113 (x)*
RTRL Elman	*0.73*	*32.113*	*32.155*	*0.03988 (-)*
RTRL Jordan	*0.64*	*3205.439*	*3233.818*	*3.643e-10 (-)*
RTRL FRNN	*0.03*	*5381.300*	*5336.219*	*0.3831 (x)*

Las conclusiones que podemos obtener de la tabla 5.8 se resumen a continuación:

- **Red que ha proporcionado mejores resultados:** red de Jordan
- **Mejor algoritmo de entrenamiento para la red Completa** SGA-BAL, CHC-BAL y BFGS

- **Mejor algoritmo de entrenamiento para la red de Elman:** SGA-BAL y CHC-BAL
- **Mejor algoritmo de entrenamiento para la red de Jordan:** SGA-BAL, CHC-BAL

La Figura 5.32 muestra los resultados del ajuste y la predicción obtenidos por la red de Jordan, entrenada con SGABFGS y estrategia baldwiniana en el problema *PIB_Andalucia*.

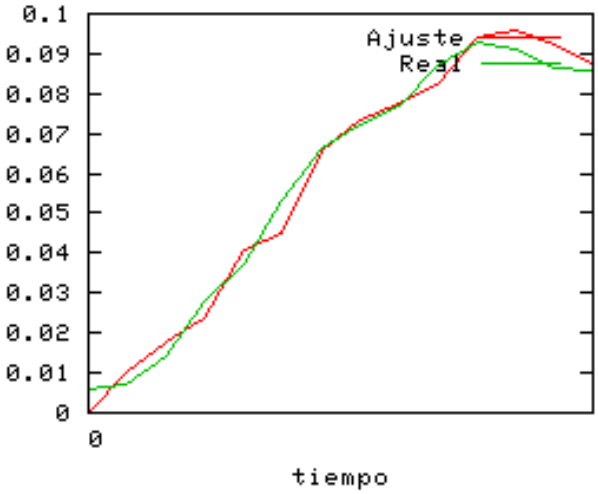

Figura 5.32. Ajuste y predicción realizamos para la serie PIB_Andalucía

Las tablas 5.17 a 5.19 muestran los resultados de la aplicación de los métodos multiobjetivo clásicos e híbridos al problema *PIB_Andalucia*. Las conclusiones obtenidas para estos algoritmos en este problema son equivalentes a las obtenidas para los problemas anteriores. De este modo, los algoritmos híbridos multiobjetivo consiguen encontrar una frontera de Pareto mejor que los métodos clásicos. Entre estos modelos los resultados son similares, llegando a ser muy parecidos a los obtenidos mediante el método de ensayo y error. Las Figura 5.33 a 5.35 muestran la comparación de las fronteras de Pareto encontradas por cada algoritmo.

Tabla 5.17: Frontera de Pareto encontrada mediante los algoritmos multiobjetivo clásicos e híbridos, para redes de Elman

Algoritmos	Neuronas	MSE Mediano (trn)	MSE Mediano (tst)
SPEA2	3	60.535110	3.247595
	4	1.689461	2.197396
	5	21.549400	2.018093

	6	26.281100	1.805955
NSGA2	3	2.939876	0.240507
	4	0.426805	0.152810
SPEA2H	3	0.121105	0.172516
	4	0.113185	0.120258
	5	0.349588	0.108513
	9	0.072617	0.105476
NSGA2H	3	0.104402	0.109283
	4	0.227845	0.091142
	7	0.108060	0.079434

Tabla 5.18: Frontera de Pareto encontrada mediante los algoritmos multiobjetivo clásicos e híbridos, para redes Completas

Algoritmos	**Neuronas**	**MSE Mediano (trn)**	**MSE Mediano (tst)**
SPEA2	3	19.314350	1.471616
	4	34.904930	0.831543
NSGA2	3	2.161730	0.125439
SPEA2H	3	0.153512	0.083339
	5	0.133481	0.078251
NSGA2H	3	0.079071	0.082757
	4	0.079953	0.066002

Tabla 5.19: Frontera de Pareto encontrada mediante los algoritmos multiobjetivo clásicos e híbridos, para redes de Jordan

Algoritmos	**Neuronas**	**MSE Mediano (trn)**	**MSE Mediano (tst)**
SPEA2	3	5.739863	2.954523
	4	9.512045	0.558370
	5	0.469343	0.227020
	11	1.118608	0.163983
NSGA2	3	0.338000	0.181654
	4	0.838058	0.150414
	6	0.162069	0.149221

SPEA2H	3	0.147858	0.147109
	9	0.157177	0.145354
NSGA2H	3	0.144370	0.146849
	4	0.139806	0.141309

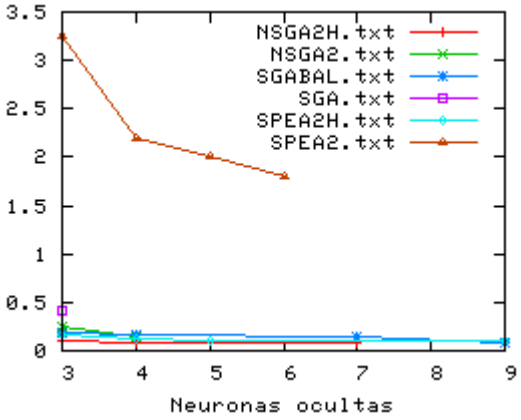

Figura 5.33. Fronteras de Pareto calculadas para redes de Elman

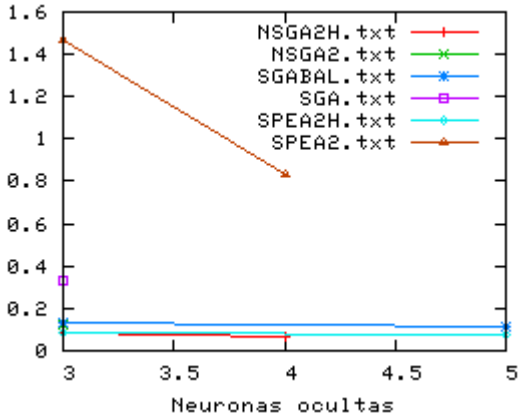

Figura 5.34. Fronteras de Pareto calculadas para redes Completas

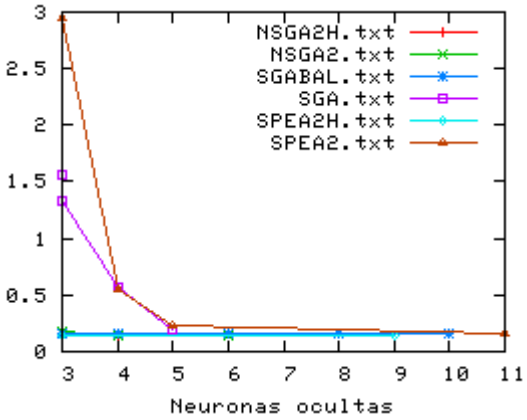

Figura 5.35. Fronteras de Pareto calculadas para redes de Jordan

Conclusiones

Una vez desarrollado nuestro trabajo, a continuación resumimos las principales conclusiones que hemos ido obteniendo a lo largo del documento:

- Hemos estudiado el problema del entrenamiento de las redes neuronales recurrentes dinámicas, partiendo de las limitaciones existentes en los algoritmos clásicos de entrenamiento. El problema se ha abordado considerando diferentes modelos basados en algoritmos evolutivos, los cuales han mejorado los resultados obtenidos por los algoritmos clásicos.
- Hemos estudiado un conjunto de algoritmos evolutivos genéticos, como propuesta para mejorar el entrenamiento de las RNRD. El estudio experimental ha considerado diferentes propuestas que abordan el problema de la diversidad y la convergencia de los modelos evolutivos desde diferentes perspectivas: mediante los operadores genéticos, mediante la separación espacial de los cromosomas, y mediante una propuesta de algoritmo basado en la estrategia de evolución CHC, modificada parapoder operar sobre cromosomas de codificación real.
- Con respecto al estudio realizado sobre los diferentes modelos de algoritmos genéticos clásicos (esquemas estacionario, generacional y mixto), los mejores resultados para entrenar los modelos de RNRD han sido obtenidos por el modelo mixto,especialmente cuando se escoge un reemplazamiento elitista sobre los peores individuos de la población y un operador de mutación aleatoria. En cambio, la selección de los operadores de selección y recombinación dependen en mayor medida deltipo de red a entrenar y del problema, siendo las mejores opciones la selección por torneo binario y lineal, los cruces BLX y Heurístico para redes de Elman y Jordan, y los cruces BLX y aritmético para redes recurrentes completas.
- Considerando el estudio del entrenamiento evolutivo de RNRD mediantelas técnicas de mejora de diversidad y convergencia (algoritmos CHC y Clearing), podemos concluir

que en ambos casos el operador de cruce que mejores resultados proporciona en conjunto es el BLX. Aún así, estos no superan a la propuesta de algoritmo de entrenamiento genético mixto.

- Hemos desarrollado un conjunto de algoritmos híbridos para entrenamiento de RNRD, basados en la estrategia de evolución genética estacionaria y el algoritmo CHC. Los métodos de hibridación considerados han sido el modelo lamarckiano y el baldwiniano. Como conclusión general, el método baldwiniano ha conseguido mejores resultados que el lamarckiano, especialmente en la hibridación basada en el algoritmo CHC. También podemos concluir que la selección de operadores evolutivos no es un aspecto crítico al utilizar estos métodos para entrenar RNRD, especialmente cuando se utiliza una hibridación de tipo baldwiniano.

- Hemos estudiado otros métodos híbridos basados en algoritmos genéticos con estrategia generacional y mixta, cuyos resultados han sido estadísticamente equivalentes a una ejecución multiarranque del operador de búsqueda local BFGS.

- Hemos desarrollado un método para entrenar y optimizar la estructura interna de una RNRD en un problema dado, mediante el uso de algoritmos evolutivos multiobjetivo. Este método desarrolla la representación de cada modelo de redestudiado, junto con los operadores de cruce y mutación necesarios para realizar esta tarea. El método multiobjetivo clásico que mejores resultados ha proporcionado en la sección experimental de cada problema ha sido NSGA-II. Además, estos resultados mejoran en ocasiones a los obtenidos cuando se utiliza el método de ensayo y error para calcular el número de neuronas ocultas óptimo para cada modelo de red recurrente En otros casos, se pueden obtener los mismos resultados. Así, la selección de NSGA-II para entrenamiento y optimización de RNRD es una opción prometedora, ya que libera al usuario de la necesidad de calcular mediante otros métodos la estructura topológica óptima de la red, para posteriormente entrenarla.

- Hemos propuesto diferentes modelos evolutivos híbridos multiobjetivo, basados en los algoritmos SPEA-II y NSGA-II, para el entrenamiento y la optimización de RNRD. Entre estos, el algoritmo híbrido SPEA2H ha mostrado una mayor eficacia para entrenar RNRD. Al igual que ocurre con los métodos de entrenamiento híbridos en el capítulo 3, la utilización de la estrategia baldwiniana sobre un esquema evolutivo estacionario produce los mejores resultados, al compararla sobre otros modelos como el generacional o el mixto.

- Hemos validado los modelos de RNRD estudiados y los algoritmos de entrenamiento presentados, utilizando el benchmark de series temporales CATS Los resultados son muy prometedores, mejorando a los métodos que se presentaron a la competición.
- Hemos aplicado los métodos desarrollados para entrenamiento y optimización de RNRD en problemas de series temporales de índole económico, observando una similitud patente con las conclusiones obtenidas durante la validación de los algoritmos de entrenamiento en el benchmark CATS.

Desarrollos futuros

- En términos generales, podemos decir que la investigación realizada está enmarcada en un área con gran futuro tanto a nivel teórico como práctico.
- Aplicación de los algoritmos desarrollados para entrenamiento de RNRD en problemas de inferencia gramatical.
- Detección de correo SPAM, basándonos en los desarrollos realizados en el punto anterior.
- Desarrollo de software libre y su difusión (sistema Octave, sistema R).
- Aplicación en predicción y reconocimiento de cadenas de proteínas y ADN
- Integración en sistemas de ayuda a la decisión ubicuos distribuidos, de índole econométrica.
- Aplicación en reconocimiento de patrones temporales en señales de video y audio.
- Adaptación para entrenamiento y optimización (estructura de entradas y neuronas ocultas) de redes neuronales RBF en problemas de series temporales

Queremos hacer notar el hecho de que todos los trabajos expuestos para su desarrollo futuro son de un gran interés. El último de todos ya ha sido estudiado y está sometido para publicación en revista de índice de impacto.

Algunos de los objetivos planteados para el desarrollo futuro han sido previamente estudiados por los autores (aplicación a reconocimiento de gramáticas, ADN, proteínas...). No obstante, las mejoras que pueden aportar los algoritmos desarrollados en este trabajo pueden llegar a producir resultados prometedores al ser aplicarlos sobre estos problemas, mejorando las aportaciones realizadas.

Anexo I

Por motivos de espacio, el Anexo I sólo está disponible en la versión electrónica del documento (ver CD adjunto).

Anexo II

Operadores genéticos para codificación real

Este apartado estudia en detalle los diferentes operadores genéticos considerados para realizar la experimentación. Los operadores genéticos realizan un conjunto de acciones realizadas sobre los elementos de una población. Guardan una estrecharelación con los procesos naturales de evolución, de modo que hay operadores de selección, recombinación, mutación y reemplazamiento. Existen diversos tipos de operadores genéticos, dependiendo de la codificación utilizada para representar un cromosoma. A continuación, hacemos una breve revisión de los operadores para codificación real.

1. Operadores de selección

Dada una población de soluciones, el operador de selección escoge los individuos que se utilizarán para obtener nuevos descendientes mediante el operador de cruce. Ligado a este operador está el concepto de *presión selectiva*, la cual se define como la medida mediante la cual la evolución de la población va regida por los mejores individuos. Un buen operador de selección debe dar la posibilidad a los peores individuos de la población de poder reproducirse, ya que éstos pueden codificar algún tipo de información importante.A su vez, debe priorizar la selección de los mejores individuos, para así poder explotar mejor elespacio de soluciones. Otra característica importante de un operador de selección es que debebrindar la posibilidad de que un mismo individuo pueda reproducirse varias veces, por lo que una misma solución puede ser seleccionada múltiples veces. A continuación enumeramos los operadores de selección más conocidos:

Selección por sorteo

Sea una población $P(t)$, formada por N cromosomas, C_i ($1 \leq i \leq N$), y $f(C_i)$ la función de adecuación del cromosoma C_i. La probabilidad de selección del cromosoma C_i se calcula mediante la siguiente fórmula:

$$p_i = \frac{f(C_i)}{\sum_{j=1}^{N} f(C_j)}$$

Una vez calculada la probabilidad de un individuo, el proceso de selección se realiza según indica el siguiente algoritmo:

1. $\{S\} = \emptyset$
2. $q_0 = 0$
3. $q_i = p_1 + p_2 + \ldots + p_i \quad (i = 1..N)$
4. Mientras $|S| < K$, hacer
   - 4.1. Asignar $r = rand(0,1)$
   - 4.2. Asignar $\{S\} = \{S\} + C_i$, $(q_{i-1} < r < q_i)$
5. Devolver $\{S\}$

Algoritmo A.1: Selección por sorteo

En el algoritmo A.1, $\{S\}$ es el conjunto de individuos seleccionados, inicialmente vacío. Los pasos 2 y 3 calculan las probabilidades acumuladas entre los individuos de la población. La selección de individuos se realiza en el paso 4, hasta que se hayan realizado un total de K selecciones. Así, el conjunto $\{S\}$ puede contener varias instancias de un mismo cromosoma

. La selección se lleva a cabo generando un número aleatorio, r, en el intervalo [0, 1]. El cromosoma escogido es aquel cromosoma C_i, cuya probabilidad acumulada sea tal que ($q_{i-1} < r \leq q_i$).

Selección por ruleta

El operador de selección por ruleta es análogo al anterior, salvo que la selección por ruleta se genera un único número aleatorio simple, r, y con él se asignan todas las muestras de modo similar a como se haría al girar una ruleta. Para simular una tirada de ruleta se definen a partir de r los siguientes números:

$$q_i = \frac{r+i-1}{k} \quad \forall i = 1..K$$

Considerando el cálculo de los valores q_i según la fórmula anterior, la selección por ruleta se lleva a cabo de la siguiente forma:

1. *{S}=Ø*
2. *r=rand(0,1)*
3. *Calcular q_i*
4. *Mientras |S|<K, hacer*
   - *4.1.Asignar {S}= {S}+C_i , ($q_{i-1} < r < q_i$)*
5. *Devolver {S}*

Algoritmo A.2: Selección por ruleta

Al utilizar este tipo de selección debemos tener en cuenta la desventaja de que la presión selectiva es muy grande, ya que los peores individuos casi nunca formarán parte del conjunto de padres, y la diversidad puede disminuir en la población si no se trata mediante métodos de aumento de la diversidad.

Selección por torneo

En la selección por torneo, cada elemento del conjunto de padres se toma escogiendo el mejor de los individuos de un conjunto de z elementos tomados al azar desde la población. Este proceso se repite k veces, hasta completar el conjunto de padres. El parámetro z suele ser un entero pequeño, comparado con el tamaño de la población, normalmente 2(torneo binario) o 3 (torneo ternario), debido a que grandes valores de z puede producir una excesiva presión selectiva. El esquema básico de la selección por torneo se muestra en el siguiente algoritmo:

1. {S}=Ø

2. Mientras |S|<K, hacer

2.1. {H}$_z$= Escoger z soluciones aleatoriamente entre la población

2.2. H_m= solución con mejor adecuación en {H}$_z$

2.3. Asignar {S}= {S}+H_m

3. Devolver {S}

Algoritmo A.3: Selección por torneo

Selección equiprobable

La selección equiprobable escoge los K padres aleatoriamente, siguiendo una distribución uniforme, desde la población actual. El esquema básico de la selección equiprobable se muestra en el siguiente algoritmo:

{S} =Ø

Mientras |S|<K, hacer

r= rand(1,N)

Asignar {S}= {S}+C_r

Devolver {S}

Algoritmo A.4: Selección equiprobable

En el esquema anterior, el valor *r* es un número entero, generado aleatoriamente a partir de una distribución uniforme, entre 1 y *N*, siendo *N* el tamaño de la población.

Selección de orden

En la selección por orden, los individuos se ordenan de mejor a peorpor el valor de su función de adecuación. El lugar que ocupa un individuo en dicha lista se llama orden del cromosoma. Según este operador, la selección de un individuo se rigesegún su orden, en lugar de usar la función de probabilidad para seleccionar un individuo. Normalmente, se aplica una interpolación lineal basada en dicho orden, según muestra la siguienteecuación:

$$q_i = m + (M - m)\frac{r(C_i) - 1}{N - 1}$$

En la ecuación anterior, $r(C_i)$ simboliza el orden del cromosoma C_i. El valor N corresponde al número de soluciones dentro de la población. M y m son los valores máximo y mínimo de una función de probabilidad. La selección por orden puede aplicarse siguiendo el esquema ilustrado por el Algoritmo A.4, realizando los cálculosde los valores q_i según indica la ecuación anterior.

2. Operadores de cruce

El cruce consiste en la recombinación de dos o más padres, procedentes de una población de individuos, para obtener nuevas soluciones descendientes. Una característica propia del cruce es que los descendientes deben de tener en sus genes información procedente de sus padres; en caso contrario, estaríamos ante un operador demutación. Generalmente, el cruce está considerado como un operador de explotación del espacio de soluciones, debido a que utiliza la información de los padres para encontrar soluciones mejores en zonas similares del espacio de búsqueda, basadas en dicha información.

Dependiendo del cruce a utilizar, la diversidad puede verse reducida Se debe evitar que el operador de cruce amplíe excesivamente la diversidad, ya que lo que podría estar teniendo lugar es una búsqueda aleatoria. Si, por el contrario, el cruce reduce excesivamente la diversidad, el algoritmo podría converger prematuramente hacia óptimos locales. Se debe procurar que los operadores genéticos mantengan un consenso en cuanto a diversidad y convergencia, para que el espacio de soluciones se explore y explote de forma adecuada.

Dados dos cromosomas padres $C_1= (c_1^1, ..., c_n^1)$ y $C_2= (c_1^2, ..., c_n^2)$, siendo n el tamaño del cromosoma, a continuación explicamos los operadores de cruce paracodificación real que hemos considerado en nuestro estudio.

Cruce plano

El operador de cruce plano genera un hijo, $H= (h_1, h_2, .., h_n)$, donde h_i es un valor escogido de forma aleatoria, a partir de una distribución uniformeen el intervalo $[min(c_i^1, c_{i,}^2), max(c_i^1, c_{i,}^2)]$.

Cruce simple

El cruce simple genera dos hijos, $H_1= (h_1^1, h_2^1, .., h_n^1)$ y $H_2= (h_1^2, h_2^2, .., h_n^2)$, escogiendo una posición aleatoria dentro del cromosoma, i ($1 \leq i \leq n$). Escogida la posición i, los nuevos cromosomas H_1 y H_2 se genera a partir de los dos padres C_1 y C_2 de la siguiente forma:

$H_1= (c_1^1, ..., c_i^1, c_{i+1}^2, ..., c_n^2)$

$H_2= (c_1^2, ..., c_i^2, c_{i+1}^1, ..., c_n^1)$

Cruce aritmético

El cruce simple genera dos hijos, $H_1= (h_1^1, h_2^1, .., h_n^1)$ y $H_2= (h_1^2, h_2^2, .., h_n^2)$, a partir de los padres C_1 y C_2, de la siguiente forma:

$h_i^1= L*c_i^1 + (1-L)*c_i^2$, y

$h_i^2= L*c_i^2 + (1-L)*c_i^1$

El valor L es un número real perteneciente al intervalo (0,1), el cual puede ser fijado como parámetro al algoritmo, o ser modificado en función del número de generaciones realizadas. Si se utiliza la primera opción, el cruce es llamado *cruce aritmético uniforme*. En el segundo caso, el cruce es llamado *cruce aritmético no uniforme*.

Cruce BLX-α

El cruce BLX-α genera un único hijo, $H= (h_1, ..., h_n)$, a partir de los padres C_1 y C_2, de la siguiente forma:

$h_i= rand(m-I\cdot\alpha, M+I\cdot\alpha)$,

donde: $m= min(c_i^1, c_i^2)$; $M= max(c_i^1, c_i^2)$; $I=M-m$. La función *rand(x,y)* genera un número real aleatorio, a partir de una distribución uniforme, enel intervalo [*x*, *y*]. El valor α es un parámetro de entrada al algoritmo.

Cruce discreto

El cruce discreto genera un único hijo, $H= (h_1, ..., h_n)$, a partir de los padres C_1 y C_2, donde h_i se genera de forma aleatoria escogiendo elementos del conjunto $\{c_i^1, c_i^2\}$.

Cruce lineal extendido

El cruce discreto genera un único hijo, $H= (h_1, ..., h_n)$, a partir de los padres C_1 y C_2, donde h_i se genera de la siguiente forma:

$h_i= c_i^1 + a*(c_i^2 - c_i^1)$

En la ecuación anterior, *a* es un valor escogido aleatoriamente en el intervalo *[-0.25, 1.25]*.

Cruce heurístico de Wright

El cruce heurístico de Wright genera un único hijo, $H= (h_1, ..., h_n)$. Supongamos que la función de adecuación del cromosoma padre C_1 es mejor que la de C_2. Entonces h_i se genera de la siguiente forma:

$$h_i= r*(c_i^1-c_i^2) + c_i^1$$

Consecuentemente, si la función de adecuación del cromosoma padre C_2 es mejor que la del cromosoma C_1, el valor h_i se calcula de la siguiente forma:

$$h_i= r*(c_i^2-c_i^1) + c_i^2$$

En las ecuaciones anteriores, el valor r es un número aleatorio, generado a partir de una distribución uniforme en el intervalo $[0, 1]$.

3. Operadores de mutación

El operador de mutación altera arbitrariamente uno o más genes de un cromosoma seleccionado. Los objetivos básicos del operador de mutación son dos: incrementar la diversidad de la población, explorando nuevas zonas del espacio de búsqueda, aún no tenidas en cuenta. La aplicación del operador de mutación previene la convergencia prematura del algoritmo hacia soluciones locales. De este modo, nos aseguramos de que, al menos, la probabilidad de explorar todas las zonas del espacio de soluciones no es nunca cero.

Cada posición en cada uno de los cromosomas de la población puede sufrir un cambio aleatorio de acuerdo a la probabilidad definida por la *probailidad de mutación*, p_m. A continuación, resumimos los operadores de mutación para codificación real que hemos considerado en nuestro estudio:

Mutación aleatoria

Dado un cromosoma $C= (c_1, ..., c_n)$, donde n es el tamaño del cromosoma, el operador de mutación aleatoria actúa sobre el valor c_i, modificándolo con un nuevo valor aleatorio, generado aleatoriamente a partir de una distribución uniforme, en el intervalo donde está definido c_i, $[L_m, L_M]$.

Mutación usando desplazamiento

La motivación para el uso de este operador de mutación es la siguiente: Cuando se optimiza problemas de naturaleza continua con máximos y mínimos locales, en un determinado momento se obtiene una solución situada cerca en el óptimo local. En estos caso puede ser interesante generar cromosomas alrededor de esta solución para aproximarnos al punto cumbre de tal óptimo. Para ello se puede *deslizar* el cromosoma en un valor que lo aumente o disminuya en una pequeña cantidad aleatoria. El máximo valor para el deslizamiento L viene definido por el usuario.

Dado un cromosoma $C= (c_1, ..., c_n)$, donde n es el tamaño del cromosoma, el operador de mutación por desplazamiento actúa sobre el valor c_i, modificándolo con un nuevo valor aleatorio, generado aleatoriamente a partir de una distribución unforme, en el intervalo $[c_i-L, c_i+L]$.

4. Reemplazamiento

Como se ha comentado anteriormente, la presión selectiva se ve afectada, en parte, por cómo los cromosomas descendientes reemplazan a los de la población actual. Existen diversos métodos de reemplazamiento, determinísticos y aleatorios. A continuación, explicamos tres modelos de reemplazamiento de los más comunes.

Reemplazamiento completo

Este modelo de reemplazamiento se ajusta a la metodología del modelo de algoritmo genético generacional. De este modo, todos los descendientes sustituyen a la población actual. Por tanto, deben existir al menos el mismo número de cromosomas descendientes que número de individuos haya en la población. Si hay mayor número de hijos, se deberá de realizar algún proceso de selección para quedarnos con un número descendientes igual al tamaño de la población.

Reemplazamiento por padres

Según este modelo de reemplazamiento, los descendientes sustituyen a alguno de los padres. Existen variantes como que sólo se realiza el reemplazamiento si el descendiente cumple ciertas características; por ejemplo, que se adapte al entorno mejor que el padre a reemplazar.

Reemplazamiento por peores

Según este modelo de reemplazamiento, los descendientes reemplazan a los peores individuos de la población. La utilización de éste método de reemplazamiento en modelos genéticos estacionarios puede producir convergencia prematura al disminuir rápidamente la diversidad.

5. Elitismo

El elitismo es ampliamente usado en algoritmos genéticos. Se encarga de que los mejores individuos de la población no desaparezcan tras la mutación y el cruce. Si al finalizar todos los procesos evolutivos en una generación, los individuos mejor adaptados al entorno no se encuentran dentro de dicha población, se sustituyen algunos cromosomas por estos individuos élite. Generalmente, los individuos sustituidos son los peores de la población, aunque existen otras variantes en las que los individuos sustituidos son seleccionados aleatoriamente en la población.

Bibliografía

[AUS99] Aussem A., Dinamical Recurrent Neural Networks towards prediction and modeling of dynamical systems, Neurocomputing, vol. 28, no. 1-3, pp. 207-232, 1999

[BAC96] Back T. (1996). Evolutionary Algorithms in Theory and Practice. Oxford.

[BAC96] Back T., Fogel D., Michalewicz Z., (1996). Handbook of Evolutionary Computation. Institute of Physics Publishing and Oxford University Press.

[BEN94] Bengio, Y., Simard, P., Frasconi, P. (1994) "Learning Long-Term Dependencies with Gradient Descent is Difficult", IEEE TNN, 5(2), pp. 157-166.

[BEN96] Bentley P. J., Wakefield J. P, (1996), An Analysis of Multiobjective optimization within Genetic Algorithms, Technical Report ENGPJB96, University of Huddersfield, UK.

[BER04] Berretta R., Rodrigues L.F., (2004), A Memetic Algorithm for a Multistage Capacitated lot-sizing problem, International Journal of Production Economics, vol. 87, no. 1, pp. 67-81.

[BOS03] Bosman P.A.N., Thierens D., (2003), The Balance Between Proximity and Diversity in Multiobjective Evolutionary Algorithms, IEEE Transactions on Evolutionary Computation, Vol. 7, No. 2, pp. 174--188.

[BYR95] Byrd R. H., Lu P., Nocedal J., (1995). A Limited Memory Algorithm for Bound Constrained Optimization, SIAM Journal on Scientific and Statistical Computing , 16, 5, pp. 1190-1208.

[BRA99] Bradley E., (1999), "Time-Series Analysis", in Intelligent Data Analysis: An Introduction, Ed. Springer.

[BRO86] Brockwell P.J., Davis R.A., (1986), "Time Series: Theory and Methods", Ed. Springer Series in Statistics

[CAO03] Caorsi S., Massa A., Pastorino M., Raffetto M., Randazzo A., (2003), "Detection of Buried Inhomogeneous Elliptic Cylinders by a Memetic Algorithm," IEEE Transactions on Antennas and Propagation, vol. 51, no. 10 , pp. 2878-2884

[CAI04] Cai X., Zhang N., Venayagamoorthy G., Wunsch D., (2004), Time Series Prediction with Recurrent Neural Networks Using a Hybrid PSO-EA Algorithm, IJCNN'04, Budapest.

[CEL04] Cellier F.E., Nebot A., (2004), Multi-resolution Time-Series Prediction Using Fuzzy Inductive Reasoning, IJCNN'04, Budapest.

[CHA04] Chan H.W., Leung W.C., Chiu K.C., Xu L-, (2004), BYY Harmony Learning Based Mixture of Experts Model for Non-stationary Time Series Prediction, IJCNN'04, Budapest.

[CHE99] Wilamowski B.M., Chen Y., Malinowski A., (1999), Efficient Algorithm for Training Neural networks with one Hidden layer," in Proc. International Joint Conference on Neural Networks, Washington DC, pp.1725-1728.

[COE99] Coello Coello C.A., (1999), A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques, Knowledge and Information Systems. An International Journal, 1(3):269-308.

[COE00] Coello Coello C.A., (2000) An Updated Survey of GA-Based Multiobjective Optimization Techniques. ACM Computing Surveys, 32(2):109-143.

[COE02] Coello Coello C.A.,Van Veldhuizen D.A., Lamont G.B., (2002), Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic Publishers, New York, ISBN 0-3064-6762-3.

[COL03] Collette Y., Starry P., (2003), Multiobjective Optimization. Principles and Case Studies, Springer.

[CON02] Conradie A., Miikkulainen R., Aldrich C., (20023), Intelligent process control utilising symbiotic memetic neuro-evolution, in Proc. IEEE Congress on Evolutionary Computation (CEC 2002), Hawaii, USA, pp. 623-628.

[COR03] Cordón O., Damas S., Santamaría J., (2003), A CHC Evolutionary Algorithm for 3D Image Registration. In T. Bilgic, B.D. Baets, O. Kaynak(Eds.), Fuzzy Sets and Systems, IFSA 2003, Lecture Notes in Artificial Intelligence 2715, pp. 404-411

[COR04] Cordón O., Damas S., Santamaría J., (2004), Feature-based image registration by means of the CHC evolutionary algorithm. Technical Report SCI2S-2004-09. Research Group on Soft Computing and Intelligent Information Systems. University of Granada.

[CRO04] Crone S.F., Kausch H. Pressmar D., (2004), Prediction of the CATS benchmark using a Business Forecasting Approach to Mulitlayer Perceptron Modelling, IJCNN'04, Budapest.

[CUE03] Cuéllar M.P., Delgado M., Pegalajar M., Pérez R, (2003), Predicción del endeudamiento económico español utilizando modelos bioinspirados, in Proc. SIGEF'03 Congress, León, vol. 1. Pp. 489-510.

[CUE04] Cuéllar M.P., Delgado M., Pegalajar M.C., (2004), A Comparative studyof Evolutionary Algorithms for Training Elman Recurrent Neural Networksto predict the Autonomous Indebtedness, in Proc. ICEIS, Porto, Portugal, pp. 457-461.

[CUE05] Cuéllar M.P., Delgado M., Pegalajar M.C., (2005), Redes neuronales recurrentes para predicción de series temporales in SICO'05 (aceptado)

[CUE06] Cuéllar M.P., Delgado M., Pegalajar M.C., Memetic evolutionary training for recurrent neural networks, in IEEE trans. On evolutionary computation (submitted for publication)

[DAN01] Danilo P. Mandic, Chambers J.A., (2001), Recurrent Neural Networks for Prediction, Ed. John Wiley & sons

[DAV99] Davey N, Hunt SP, Frank RJ, (1999), Time Series Prediction and NeuralNetworks, proc. 5th International Conference on Engineering Applications of Neural Networks, pp.93-98

[DAY90] Dayhoff J., (1990), Neural Network Architectures: An Introduction, Ed. Van Nostrand Reinhold.

[DEB99] Deb K., (1999), Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems, Evolutionary Computation, 7(3):205-230.

[DEB01] Deb K., (2001), Multi-Objective Optimization using Evolutionary Algorithms, John Wiley & Sons, Chichester, UK, ISBN 0-471- 87339-X.

[DEB01] Deb K., (2001), Nonlinear goal programming using multi-objective genetic algorithms, Journal of the Operational Research Society, Vol. 52, No. 3, pp. 291--302

[DEB02] Deb K., Pratap A., Agarwal A., Meyarivan T., (2002) A Fast and Elitist Multiobjective Genetic Algorithm: NSGA--II, IEEE Transactions on Evolutionary Computation, Vol. 6, No. 2, pp. 182--197.

[DEL00] Blanco, Delgado, Pegalajar, A genetic algorithm to obtain the optimal recurrent neural network, International Journal of Approximate Reasoning, vol. 23,pp. 67-83, 2000.

[DEL01] Blanco A., Delgado M., Pegalajar M.C., (2001), A Real-Coded genetic algorithm for training recurrent neural networks, Neural Networks, vol. 14,pp. 93-105, 2001.

[DEL04] Cuéllar M.P., Delgado M., Pegalajar M., Pérez R., (2004), "Predicción de Series Temporales mediante Redes Neuronales Recurrentes y algoritmos bioinspirados," in Proc. of MAEB'04 Congress, Córdoba (Spain), Pp. 585-593.

[DEL05] Cuéllar M.P., Delgado M., Pegalajar M.C., (2005), "Multiobjetive Evolutionary Optimization for Elman Recurrent Neural Networks, Applied to Time Series Prediction", in Fuzzy Economical Review, vol X, 1.

[DEL05] Cuéllar M.P., Delgado M., Pegalajar M.C., (2005), "Algoritmos evolutivos híbridos para entrenamiento de redes neuronales recurrentes", in MAEB'05 (Aceptado)

[DIA96] Diaz A., (1996). Optimización Heurística y Redes Neuronales. Editorial Paraninfo.

[DIA02] Dias A.H.F., Vasconcelos J.A., (2002), Multiobjectivegenetic algorithms applied to solve optimization problems, IEEE Transactions on Magnetics, Vol.38, No. 2, pp. 1133--1136

[DIG90] Diggle P.J., (1990), Time Series: A Biostatistical Introduction, Ed. Oxford University Press

[ELE04] Eleuteri A., Acernese F., Barone F., De Rosa R., Milano L., (2004), A hierarchical Bayesian learning scheme for autoregressive neural networks: application to the CATS benchmark, IJCNN'04, Budapest.

[ELM90] Elman J.L., (1990), Finding Structure in Time, Cognitive Science, 14, pp. 179-211.

[ESH91] Eshelman, L.J. (1991). The CHC Adaptive Search Algorithm How to have safe search when engaging in nontraditional genetic recombination. In Foundations of genetic algorithms 2. Morgan Kaufmann publishers

[ESH93] Eshelman, L.J., Scahffer J.D. (1993). Real-coded genetic algorithms and interval-schemata. In Foundations of genetic algorithms 2. Morgan Kaufmann publishers

[FON95] Fonseca C.M., Peter J. Fleming P.J., (1995), An overview of evolutionary algorithms in multiobjective optimization, Evolutionary Computation, 3(1): pp. 1-16.

[FON98] Fonseca C.M., Peter J. Fleming P.J., (1998), Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms--Part I: A Unified Formulation. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 28(1), pp. 26-37.

[FON98] Fonseca C.M., Peter J. Fleming P.J., (1998), Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms--Part II:A Application Example. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 28(1), pp. 38-47.

[FOO02] Foo S.Y., Stuart G., Harvey B., Meyer-Baese Anke, (2002) Neural Network-based EKG pattern recognition, Engineering Applications of Artificial Intelligence, vol. 15, no. 3-4, pp. 253-260

[FRE91] Freeman J.A., Skapura D.M., (1991), Neural Networks:Algorithms, Applications, and Programming Techniques, Ed. Addison Wesley.

[GAR01] García-Pedrajas N., Sanz-Tapia E.R., Ortiz-Boyer D., Hervás C. (2001), Introducing multi-objective optimization in cooperative coevolution of neural networks. Lecture notes in computer science, pp. 645-652.

[GAR02] García-Gimeno, R; Hervas C. Siloniz M.I. (2002) Improving artificial neural networks with a pruning methodology and genetic algorithms for their application in microbial growth prediction in food. International journal of foodmicrobiology 72, pp. 19-30

[GAR03] García-Pedrajas N., Hervas C., Muñoz-Pérez J., (2003) Covnet: a cooperative coevolutionary model for evolving artificial neural networks. IEEEtransactions on neural networks 14, pp. 575-590

[GER02] Gers F.A., Schmidhuber J., Douglas E. (2002). Learning Nonregular Languages: A Comparison of Simple Recurrent Networks and LSTM. Neural Computation 14(9), pp. 2039-2041

[GER03] Gers F.A., Schraudolph N.N., Schmidhuber J., (2003). Learning Precise Timing with LSTM Recurrent Networks. Journal of Machine Learning Research 3, pp. 115-143

[GLO94] Glover F. (1994). Genetic Algorithms and Scatter Search: Unsuspected Potentials. Statistics and Computing 4, pp. 131-140.

[GLO96] Glover F., Martí R. (1996), Tabu Search. Kluwer AcademicPublishers

[GLO00] Glover F., Laguna M., Mart, R., (2000). Fundamentals of Scatter Search and Path Relinking, Control and Cybernetics 29 (3), pp. 653-684.

[GLO03] Glover F., Laguna M., Martí R., (2003). Scatter Search and Path Relinking: Advances and Applications. In: Glover, F., Kochenberger, G. (Eds.),Handbook of MetaHeuristics, Kluwer, pp. 1-36.

[GLO03] Glover F., Laguna M., Martí R., (2003). Scatter Search.In heory and Applications of Evolutionary Computation: Recent Trends. Ghosh, A., Tsutsui, S. (Eds.). T Springer-Verlag, forthcoming.

[GLO04] Glover F., Laguna M., Martí R., (2004). New Ideas and Applications of Scatter Search and Path Relinking. In New Optimization Techniques in Engineering. Onwubolu, G., Babu, B.V. (Eds.),, Springer-Verlag, forthcoming.

[GLO04] Glover F., Laguna M., Martí R., (2004). Scatter Search and Path Relinking: Foundations and Advanced Designs. In New Optimization Techniques in Engineering. Onwubolu, G., Babu, B.V. (Eds.), Springer-Verlag, forthcoming.

[GOL89] Goldberg D.E., (1989), Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley.

[GUP03] Gupta M.M., Jin L., Homma N., (2003), "Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory", Ed. John Wiley & sons

[HAY99] Haykin S., (1999), "Neural Networks: A Comprehensive Foundation", Ed. Prentice Hall, 2nd edition

[HER98] Herrera F. , Lozano M., Verdegay J.L., (1998). Tackling Real-Coded Genetic Algorithms: operator and tools for behavioural analysis. Artificial Intelligence review, 12, 265-319

[HER02] Hervás-Martínez C., García Pedrajas N., Muñoz-Pérez J., (2002), Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks), Neural Networks, Vol. 15, No. 10, pp. 1259--1278 .

[HER04] Herrera F., Lozano M., Krasnogor N., Molina D., (2004), Real-Coded Memetic Algorithms with Crossover Hill-Climbing. Evolutionary Computation 12:3, pp. 273-302

[HER05] Herrera F., Lozano M., Sánchez A.M., (2003), A Taxonomy for the Crossover Operator for Real-Coded Genetic Algorithms: An Experimental Study. International Journal of Intelligent Systems 18, pp. 309-338

[HU04] Hu X., Wunsch D., (2004), IJCNN 2004 Challenge Problem: Time Series Prediction with a Weighted Bidirectional Multi-stream Extended Kalman Filter, IJCNN'04, Budapest 2004.

[HUS02] Hussein A.A., (2002), An Evolutionary Artificial Neural Networks Approach for Breast Cancer Diagnosis, Artificial Intelligence in Medicine, Vol. 25, No. 3, pp. 265--281.

[HUS03] Hussein A.A., (2003), Speeding up backpropagation using multiobjective evolutionary algorithms, Neural Computation, Vol. 15, No. 11, pp. 2705--2726

[ISH98] Ishibuchi H., Murata T., (1998), Multi-Objective Genetic Local Search Algorithm and Its Application to Flowshop Scheduling. IEEE Transactions on Systems, Man and Cybernetics, 28(3), pp. 392-403.

[ISH03] Ishibuchi H., Murata T., Yoshida T., (2003), Balance Between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling, IEEE Transactions on Evolutionary Computation, Vol. 7, No. 2, pp. 204--223.

[JAE02] Jaeger H., (2002), A tutorial on training recurrent neuralnetworks, covering BPTT, RTRL, EKF, and the echo state network, GMD-report, 43p

[JAS02] Jaszkiewicz A., (2002), Genetic local search for multiple objective combinatorial optimization, European Journal of Operational Research, Vol. 137, No.1, pp. 50--71.

[KAA96] Kaastra I., Boyd M., (1996), "Designing a neural networkfor forecasting financial and economic time series", Neurocomputing, 10, pp. 215-236

[KON04] Kong S.G., (2004), Time Series Prediction with Evolvable Block-based Neural Networks, IJCNN'04, Budapest.

[KOZ04] Kozma R., Beliaev I., (2004), Time Series Prediction Using Chaotic Neural Networks: Case Study of IJCNN CATS Benchmark Test , IJCNN'04,Budapest.

[KU97] Ku K. W. C., Mak M. W., (1997), Exploring the effects of Lamarckian and Baldwinian learning in evolving recurrent neural networks, In Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 617-621, 1997.

[KUR04] Kurogi S., Ueno T., Sawa S., (2004), Batch Learning Competitive Associative Net and Its Application to Time Series Prediction, IJCNN'04, Budapest.

[LEN98] Lendasse A., Verleysen M., De Bodt E., Cottrel M., Grégorie P., (1998), Forecasting time series by Kohonen classification, proc. European Symposium on Artificial neural Networks, pp. 221-226

[LEN04] Lendasse A., Oja E., Simula O., Verleysen M., (2004) Time Series Competition: The CATS Benchmark. in Proc. International Joint Conference on Neural Networks, Budapest (Hungary), pp. 1615-1620

[LIN92] R. Linggard, D.J. Myers, C. Nightingale, (1992), Neural Networks for Vision, Speech and Natural Language, BT. Telecommunications series, Ed Chapman & Hall

[LIU98] Liu X., Begg D. W. , Fishwick R. J., (1998), Genetic approach to optimal topology/controller design of adaptive structures. International Journal for Numerical Methods in Engineering, 41, pp. 815-830.

[LIU99] Liu G.P.,Kadirkamanathan V., (1999), Multiobjective criteria for neural network structure selection and identification of nonlinear systems using genetic algorithms, IEEE Proceedings on Control Theory and Applications, Vol. 146, No. 5, pp. 373--382.

[LO00] Lo C.C., Chang W.H., (2000), A Multiobjective Hybrid Genetic Algorithm for Capacitated Multipoint Network Design Problem, IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics, 30(3):461-470.

[LOZ04] Lozano M., Herrera F., Sánchez A.M., (2004), Operadoresde Cruce con Múltiples Descendientes para Algoritmos Genéticos con Codificación Real: Estudio Experimental. In Proceedings del MAEB'04, Córdoba (Spain), 292-299.

[LU03] Lu H., Yen G.G., (2003), Rank-Density-Based Multiobjective Genetic Algorithm and Benchmark Test Function Study, IEEE Transactions on Evolutionary Computation, Vol. 7, No. 4, pp. 325-343.

[MAH95] Mahfoud S. (1995). Niching methods for genetic algorithms, PhD thesis. University of Illinois.

[MAR03] Martí R., Laguna M. (2003). Scatter Search: Diseño Básico y Estrategias avanzadas. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. 19, pp. 123-130

[MAR03] Martí R., El-Fallahi A. (2003). Tabu and Scatter Search for Training Neural Networks. In Computational Modelling and Problem Solving in the Networked world: Interfaces in Computer Science and Operational Research. Bhargava H.K., Ye N. (eds.). Kluwer Publishers, pp. 79-96

[MAR04] Martí R., El-Fallahi A. (2004). Multilayer neural networks: an experimental evaluation of on-line training methods. Computers & Operational Research 31(9), pp- 1491-1513.

[MAR05] Martí R., Laguna M., Campos V. (2005). Scatter Search vs. Genetic Algorithms: An experimental evaluation with permutation Problems. In metaheuristics optimization via Memory and Evolution: Tabu Search and Scatter Search. Cesar Rego and Bahram Alidaee (eds.). Kluwer Academic Publishers, pp.263-283

[MAR06] Martí R., El-Fallahi A., Lasdon L. (2006). Path relinking and GRG for Artificial Neural Networks. European Journal of Operational Research 169(2), pp.508-519.

[MED01] Medsker L.R., Jain L.C., (2001), "Recurrent Neural Networks. Design and Applications", Ed. CRC Press.

[MEH92] Mehra P., Wah B.W., (1992), "Artificial Neural Networks: Concepts and Theory", Ed. IEEE Computer Society Press

[MEL03] Melián, B., Moreno pérez, J.A., Marcos Moreno-Vega, J., (2003), Metaheuristics: A global view", Inteligencia Artificial (Special Issue on Metaheuristics), vol. 2, no. 19, pp. 7-28.

[MER99] Merz P., Freisleben B., (1999), A comparison of Memetic Algorithms, Tabu Search and Ant Colonies for the Quadratic Assigment Problem," in Proc. of the 1999 Congress on Evolutionary Computation, vol. 3 , pp. 6-9.

[MIC03] Michael Hüsken, Stagge P., (2003), Recurrent Neural Networks for Time Series classification, Neurocomputing, vol. 50, pp. 223-235.

[MOL04] D. Molina, Lozano M., Herrera F., Krasnogor N., (2004), Algoritmos Meméticos con Codificación Real con Técnicas de Ascensión de Colinas Basadasen el Cruce. In Proceedings del MAEB'04 (Tercer congreso español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados), Córdoba (Spain), 254-261, 2004.

[MON89] Montana D.J., Davis L., (1989), Training feedforward neural networks using genetic algorithms. In Proc. of 11th International joint conference on artificial intelligence, pp. 762-767

[MOS03] Moscato, P., Cotta Porras, C., (2003), An Introduction to Memetic Algorithms,"Inteligencia Artificial (Special Issue on Metaheuristics), vol. 2, no. 19, pp. 131-148.

[NAY99] Narayanan S., Azarm S., (1999), On Improving Multiobjective Genetic Algorithms for Design Optimization. Structural Optimization, 18, pp. 146-155.

[OSY85] Osyczka A., (1985), Multicriteria optimization for engineering design", Design Optimization, J. S. Gero, Ed. Academic Press, Inc., New York, NY, pp. 193–227.

[PAL04] Palacios-Gonzalez F., (2004), A SVCA Model for The Competition on Artificial Time Series, IJCNN'04, Budapest.

[PAR01] Parks G.T. , Li J., Balazs M.-E., Miller I., (2001), An empirical investigation of elitism in multiobjective genetic algorithms, Foundations of Computing and Decision Sciences, Vol. 26, No. 1, pp. 51--74

[PAT92] Patricia S.C., Sejnowski T.J., (1992), The Computational Brain, Ed. The MIT Press

[PEG01] Delgado M., Blanco A., Pegalajar M.C., (2001), Identification of fuzzy dinamic systems using max-min recurrent neural networks, Fuzzy Sets and Systems, 122:3, pp. 451-467

[PEG03] Pegalajar M.C., Navarro M.A. , Cuéllar M.P., Pérez R., (2003), A FIR Neural Network to model the autonomous indebtedness, in Proc. SIGEF'03, vol.2, León 2003 (Spain), pp. 199-209

[PEG05] Cuéllar M.P., Delgado M., Pegalajar M.C., (2005), An application of non-linear programming to train recurrent neural networks in time series prediction problems, in Proc. ICEIS'05, Miami (USA)

[PEG05] Delgado M., Pegalajar M.C., (2005), A Multiobjective genetic algorithm for obtaining the optimal size of a recurrent neural network for grammatical inference. Pattern Recognition, 38:9, pp. 1444-1456

[PET96] Petrowski, A. (1996). A Clearing procedure as a niching method for genetic algorithms. In proc. of congress on Evolutionary computation CEC'96.

[PLA98] Plagianakos V.P., Soritopoulos D.G., Vrahatis M.N., (2004), Evolutionary Algorithms for Integer Weight Neural Network Training. Technical Report TR98-04. Department of mathematics, University of Patras. Greece.

[PLA99] Plagianakos V.P., Vrahatis M.N., (1999). Training neural networks with 3–bit integer weights. In Proceedings of Genetic and Evolutionary Computation Conference (GECCO'99), pp. 910–915.

[PLA99] Plagianakos V.P., Vrahatis M.N., (1999). Neural network training with constrained integer weights. in Proceedings of Congress on Evolutionary Computation (CEC'99), pp. 2007-2013, Washington D.C.

[PLA00] Plagianakos V.P., Vrahatis M.N., (2000). Training Neural Networks with Threshold Activation Functions and Constrained Integer Weights. In Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'2000), Como, Italy.

[POL99] Pollock D.S.G., (1999), A Handbook of Time Series Analysis, Signal Processing, and Dynamics, Ed. Academic Press

[POM04] Herrera-Maldonado L.J., Pomares H., Rojas I., Gonzalez J., Awad M., (2004), MultiGrid-Based Fuzzy Systems for Time Series Forecasting: CATS Benchmark IJCNN Competition, IJCNN'04, Budapest.

[PRU02] Prudêncio R.B.C., Ludermir T.B., (2002), Neural Network Hybrid Learning: Genetic Algorithms & Levenberg-Marquardt, in Proc. 26th Annual Conference of the Gesellschaft für Klassifikation (GfKl), pp. 464-472, July 2002.

[RAD94] Radcliffe N.J., Surry P.D., (1994), Formal Memetic Algorithms, in Evolutionary Computing: AISB Workshop, Ed: T. Fogarty, Springer-Verlag, 1994.

[RAW91] Rawlins G.J.E., (1991), Foundations of Genetic Algorithms. ED. Morgan Kauffman.

[SAN02] Sánchez J, Galán C, MartínezS J., Hervas C., (2002), The use of a neural network to forecast daily grass pollen concentration in a mediterranean region: the southern part of the iberian peninsula. Clinical and experimental allergy 32, pp. 1606-1612

[SAR04] Sarkka S., Vehtari A., Lampinen J., (2004), Time Series Prediction by Kalman Smoother with Cross Validat ed Noise Density, IJCNN'04, Budapest.

[SCH95] Schittkowski K., Zillober Ch., (1995), Non-Linear Programming, Technical Report D-95440, Dept. of mathematics, University of Bayreuth, Germany

[SET96] Sette S., Boullar L., Langenhove L.V., (1996), Optimizing a Production Process by a Neural Network/Genetic Algorithm Approach. Engineering Applications in Artificial Intelligence, 9(6):681-689.

[SIM04] Simon G., Lee J.A., Verleysen M., Cottrell M., (2004), Double Quantization Forecasting Method for Filling Missing Data in the CATS Time Series, IJCNN'04, Budapest.

[SHI99] Shimizu Y., (1999), Multi-Objective Optimization for Site Location Problems through Hybrid Genetic Algorithm with Neural Networks. Journal of Chemical Engineering of Japan, 32(1):51-58, 1999.

[SCH00] Schmitt L., (2000). Fundamental Study: Theory of genetic algorithms. Theoretical Computer Science, 259(1-2), 1-61

[SMA96] Smagt P.V.D., Kröse B., (1996), An Introducion to Neural Networks, Ebook. University of Amsterdam, 8th edition.

[TER94] Teräsvirta T., Tjostheim D., Granger C.W.J., (1994), Aspects of Modelling Non-Linear Time Series, in Handbook of Econometrics, vol 4, Ed. R.F.Engle and D.L. McFadden

[THI02] Thiele L., Laumanns M., Deb K., Zitzler E., (2002), Combining Convergence and Diversity in Evolutionary Multi-objective Optimization, Evolutionary Computation, Vol. 10, No. 3, pp. 263--282, Fall 2002 .

[THO91] Thomas-Miller W., Sutton R.S., Werbos P.J., (1991), Neural Networks for Control, Ed. MIT Press, 2nd edition

[TOF03] Toffolo A., Benini E., (2003), Genetic Diversity as an Objective in Multi-Objective Evolutionary Algorithms, Evolutionary Computation, Vol. 11, No. 2, pp. 151-167.

[TSA02] Tsay R.S., (2002), Analysis of Financial Time Series,Ed. Wiley & sons.

[VEE95] Veelenturf L.P.J., (1995), Analysis and Applications ofArtificial Neural Networks, Ed. Prentice Hall

[VEL98] Veldhuizen D.A.V., Lamont G.B., (1998), Multiobjective Evolutionary Algorithm Research: A History and Analysis, Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering,Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

[VEL00] Veldhuizen D.A.V., Lamont G.B., (2000), Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art, Evolutionary Computation, 8(2):125-147, 2000.

[VER04] Verdes P.F., Granitto P.M., Szeliga M.I., Rebola S, Ceccatto H.A., (2004), Prediction of the CATS benchmark exploiting time-reversal symmetry, IJCNN'04, Budapest 2004.

[VIÑ01] Viñals M.P., (2001), "Series Temporales", Ed. Universidad Politécnica de Cataluña

[VIV01] Vivó-Truyols G., Torres-Lapasió J.R., Garrido-Frenich A., García-Álvarez-Coque M.C., (2001), A Hybrid genetic Algorithm with Local Search I.Discrete variables: optimization of complementary mobile phases, Chemometrics and Intelligent Laboratory Systems, vol. 59, no. 1-2, pp. 89-106.

[VIV01] Vivó-Truyols G., Torres-Lapasió J.R., Garrido-Frenich A., García-Álvarez-Coque M.C., (2001), A Hybrid genetic Algorithm with Local Search II. Continuous variables: multibatch peak deconvolution, Chemometrics and Intelligent Laboratory Systems, vol. 59, no. 1-2, pp.107-120.

[WAN95] Wan E.A., Back A., Lawrence S., Tsoi A.C., (1995). A unifying view of some Training Algorithms for Multilayer Perceptrons with FIR filter Synapses. In Neural Networks for Signal Processing 4. J. Viontzos, J. Hwang, E. Wilson (eds). IEEE Press, pp. 146-154

[WAN04] Wang X., (2004), Time-line Hidden Markov Experts for the Prediction of CATS time series, IJCNN'04, Budapest.

[WIC04] Wichard J., Ogorzalek M., (2004), Time Series Prediction with Ensemble Models, IJCNN'04, Budapest.

[WIL89] Williams R.J., Zipser D, (1989) A learning algorithm for continually running fully recurrent neural networks. Neural Computation, vol. 1, pp. 270-280.

[WIL90] Williams R.J., Peng J, "An efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network trajectories," Neural Computation, vol. 2, pp. 491-501, 1990.

[YEN04] Yen-Ping C., Sheng-Nan W., Jeen-Shing W., (2004), A Hybrid Predictor for Time Series Prediction, IJCNN'04, Budapest.

[ZEM03] Zemomi R., Racaceanu D., Zerhonn N., (2003), Recurrent Radial Basis fuction network for Time Series prediction, Engineering appl. Of Artificial Intelligence, vol. 16, no. 5-6, pp. 453-463.

[ZHU97] Zhu C., Byrd R. H., Nocedal J., (1997), L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization, ACM Transactions on Mathematical Software, Vol 23, Num. 4, pp. 550 – 560.

[ZIT98] Zitzler E., Thiele L., (1998), An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach, Technical Report 43, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

[ZIT99] Zitzler E., Thiele L., (1999), Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach, IEEE Transactions on Evolutionary Computation, 3(4), pp57-271.

[ZIT99] Zitzler E., Deb K., Thiele L., (1999), Comparison of Multiobjective Evolutionary Algorithms: Empirical Results, Technical Report 70, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland,

[ZIT00] Zitzler E., Deb K., Thiele L., (2000), Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computation, 8(2):173-195.

[ZIT01] Zitzler E., Laumanns M., Thiele L., (1999), SPEA2: Improving the Strength Pareto Evolutionary Algorithm, Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001.

[ZUR01] Zurera G., García R.M., Martínez J.A., Hervás C, (2001). Optimization of computational neural network for its application of microbial growth in foods. Food science and technology international 7, pp. 159-170