UNIVERSIDAD DE GRANADA

DISEÑO DE SISTEMAS INTELIGENTES EN PLATAFORMAS DE COMPUTO PARALELAS

TESIS DOCTORAL

Alberto Guillén Perales

2007

Departamento de Arquitectura y Tecnología de Computadores

Editor: Editorial de la Universidad de Granada
Autor: Alberto Guillén Perales
D.L.: Gr. 1551 - 2007
ISBN: 978-84-338-4400-2

UNIVERSIDAD DE GRANADA

DISEÑO DE SISTEMAS INTELIGENTES EN PLATAFORMAS DE COMPUTO PARALELAS

Memoria presentada por

Alberto Guillén Perales

Para optar al grado de

DOCTOR EN INGENIERÍA INFORMÁTICA

Fdo. Alberto Guillén Perales

D. Ignacio Rojas Ruiz, **D. Jesús González Peñalver** y **D. Héctor Pomares Cintas**, Profesores Titulares de Universidad del Departamento de Arquitectura y Tecnología de Computadores

CERTIFICAN

Que la memoria titulada:**"Diseño de Sistemas Inteligentes en Plataformas de Computo Paralelas"** ha sido realizada por **D. Alberto Guillén Perales** bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de Doctor en Ingeniería Informática.

Granada, a 4 de Junio de 2007

Fdo. Ignacio Rojas Ruiz	Fdo. Jesús González Peñalver	Héctor Pomares Cintas
Director de la Tesis	Director de la Tesis	Director de la Tesis

A mi familia

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

ÍNDICE DE ALGORITMOS

1. INTRODUCCIÓN

Este primer capítulo se presenta el problema que se trata de resolver en esta memoria: la aproximación de funciones. Esta introducción comenzará con una definición coloquial para, posteriormente, ser formulada formalmente.

Además este capítulo también presenta brevemente el resto de contenidos de la memoria comentando capítulo a capítulo como se aborda la solución al problema planteado.

Por último, tras el resumen de contenidos, se describen las principales aportaciones resultantes del trabajo realizado, teniendo en cuenta la innovación que suponen a nivel cientfífico.

1.1. Aprendiendo lo observado

El trabajo desarrollado en esta memoria está motivado por la intrínseca necesidad del ser humano por satisfacer su ansia de conocimiento con una anticipación en el tiempo (¿Cuantas veces se mira al cielo preguntándose a uno mismo si lloverá?). Durante la historia de la humanidad se han empleado métodos de lo más variopinto para intentar conocer qué iba a ocurrir en un futuro, pasando por dados, huesos, cartas, etc. Incluso hoy en día todavía se puede leer en cualquier periódico el horóscopo, que no es más que otro método para poder **predecir** lo que va a ocurrir basándose en la posición de los astros.

Al mirar el significado de la palabra **predecir** en el diccionario de la Real Academia de la Lengua Española, aparece como definición:

> *(Del latín praedicĕre). Anunciar por revelación, ciencia o conjetura algo que ha de suceder.*

En esta definición puede observarse como se describen las bases sobre las cuales se pueden realizar predicciones. Las técnicas y metodologías desarrolladas en el trabajo expuesto a continuación se corresponden con la palabra ciencia dentro de la definición de predecir.

La humanidad, además de basar sus predicciones en elementos revelados o basados en conjeturas, ha ido desarrollando técnicas y métodos basados en la **ciencia**. ¿Qué es exactamente la ciencia? Pues regresando al diccionario de la Real Academia se define ciencia como:

> *(Del latín scientĭa). Conjunto de conocimientos obtenidos mediante la observación y el razonamiento, sistemáticamente estructurados y de los que se deducen principios y leyes generales,*

y tras esta definición aparecen una serie de complementos entre los cuales conviene destacar:

- *exactas: matemáticas: (Del latín mathematĭca). 1. femenino. Ciencia deductiva que estudia las propiedades de los entes abstractos, como números, figuras geométricas o símbolos, y sus relaciones.*

- *naturales. 1. femenino, plural. Las que tienen por objeto el estudio de la naturaleza, como la geología, la botánica, la zoología, etc. A veces se incluyen la física, la química, etc.*

Las herramientas que se emplean en esta memoria para realizar las predicciones están basadas en las ciencias exactas aunque los modelos fueron desarrollados mediante el análisis y observación de las ciencias naturales.

El trabajo desarrollado en esta memoria trata sobre el diseño de ciertos modelos matemáticos que permiten predecir, aunque para ser más exactos, se debe formular el problema de forma abstracta y exacta evitando ambigüedades que pueda originar la definición de la palabra. La definición formal del problema a resolver es: dados una serie de observaciones $\{(\vec{x}_k; y_k); k = 1, ..., n\}$ con su correspondiente salida $y_k = F(\vec{x}_k) \in \mathbb{R}$ y $\vec{x}_k \in \mathbb{R}^d$, se desea obtener una función $\mathcal{F}$ tal que $y_k \simeq \mathcal{F}(\vec{x}_k)$. Como se puede ver en la definición de predicción, existe una mención explícita a la dimensión temporal, sin embargo, en la definición matemática, al abstraer al máximo posible, no se identifica al tiempo como una variable en concreto. En realidad se podría seguir considerando la palabra predecir puesto que para el modelo que proporciona las salidas, una vez creado, cada entrada nueva es temporalmente posterior a su creación y, consecuentemente, futura en el tiempo. Para evitar confusiones, a partir de ahora se utilizará el término **aproximación funcional** para hacer referencia al problema que se desea

resolver: dada una hipotética función (que asigna una salida a un vector de entrada) se desea construir un modelo que sea capaz de aproximar a la función hipotética, es decir, que para el mismo conjunto de entradas con sus salidas conocidas, sea capaz de generar aproximadamente las mismas salidas.

La humanidad puede obtener innumerables beneficios si realmente es capaz de desarrollar modelos y metodologías de diseño que puedan ser capaces de aproximar funciones.

La memoria de esta tesis está organizada del siguiente modo:

- El Capítulo 2 presenta con detalle el modelo matemático escogido para resolver el problema planteando en los párrafos anteriores. El modelo consiste en un tipo especial de red neuronal, las redes neuronales de funciones de base radial (*Radial Basis Function Neural Network, RBFNN*). Además del modelo, también se describen dos metodologías clásicas de diseño basadas en métodos de optimización local y global. Este capítulo asienta las bases sobre las que se han desarrollado todas las innovaciones presentadas en esta memoria

- El Capítulo 3 analiza los problemas presentes en una de las etapas dentro de la metodología de diseño de optimización local presentadas en el Capítulo 2. Más concretamente, se centra en la inicialización de uno de los parámetros de una RBFNN mediante algoritmos de *clustering* haciendo un estudio de los elementos que pueden ser mejorados para, posteriormente, proponer un nuevo algoritmo que suple las carencias observadas en el algoritmo original. A continuación, dentro del mismo capítulo, se hace un estudio del comportamiento del nuevo algoritmo cuando se utilizan distintos métodos de particionamiento de datos propuestos en la bibliografía. Por último, se presenta un nuevo algoritmo diseñado desde una nueva perspectiva que permite realizar un diseño

más completo que con los algoritmos anteriores considerando todos los parámetros que definen el modelo matemático utilizado para la aproximación.

- El Capítulo 4 trata sobre la segunda metodología presentada en el Capítulo 2: la optimización global. La innovación del trabajo consiste en adaptar un algoritmo evolutivo ya existente en la bibliografía al problema concreto del diseño de una RBFNN con el objetivo de poder aproximar funciones desconocidas. En este contexto se presentan nuevos operadores de cruce y mutación basados en nuevas heurísticas y también se presenta un nuevo método que permite mantener un equilibrio entre los distintos aspectos que requiere el diseño de una RBFNN.

- Puesto que la metodología propuesta en el Capítulo 4 requiere una capacidad de computo considerable, en el Capítulo 5 se presenta una solución a este problema paralelizando el algoritmo. Esta paralelización se realiza a nivel funcional de grano grueso puesto que cada procesador realizará una operación compleja distinta sobre un conjunto de datos. También se obtiene un paralelismo de datos dado que se pueden distribuir los datos en subconjuntos y realizar las mismas operaciones sobre ellos, obteniéndose una paralelización de grano fino cuando los conjuntos de datos están tan distribuidos entre los procesadores que su tamaño es muy reducido.

- En el Capítulo 6, se utiliza la metodología de diseño de RBFNN propuesta en esta memoria para resolver dos problemas del mundo real. El primero consiste en el cálculo de la cantidad de combustible necesaria para llevar a cabo el proceso de reducción de mineral para obtener

Níquel. El segundo problema consiste en la predicción del peso que tendrá un recién nacido, con el objetivo de prever ciertas maniobras y complicaciones que pueden producirse durante el parto.

- Finalmente, las principales aportaciones científicas derivadas del presente trabajo, junto con las conclusiones más relevantes obtenidas a lo largo del trabajo desarrollado en esta memoria se exponen en el Capítulo 7.

A un nivel global, la memoria del trabajo desarrollado comprende la descripción formal del modelo matemático que aproxima funciones así como de diversas metodologías para su diseño existentes. Tras esto, se proponen nuevos algoritmos que permiten crear modelos capaces de aproximar con gran precisión y son aplicados a problemas del mundo real, mostrando la utilidad del trabajo realizado.

En la Figura 1.1 se muestra un diagrama el cual representa los distintos elementos abordados en esta memoria así como sus interacciones. Los círculos amarillos representan los modelos de computación ya existentes y los cuadros de texto en color salmón representan las innovaciones aportadas por esta memoria. También ha de considerarse innovación cada flecha roja puesto que el simple hecho de relacionar estas herramientas y modelos entre sí para poder resolver un problema concreto ya es, de por sí, un trabajo innovador.

Las principales aportaciones que se presentan en esta memoria son:

- en el campo del campo de las redes neuronales de funciones de base radial (RBFNN), se proponen nuevas técnicas dentro de las dos metodologías de diseño (optimización local y global).

- Dentro de la metodología de optimización local mediante un primer paso de inicialización de parámetros con algoritmos de *clustering*, se

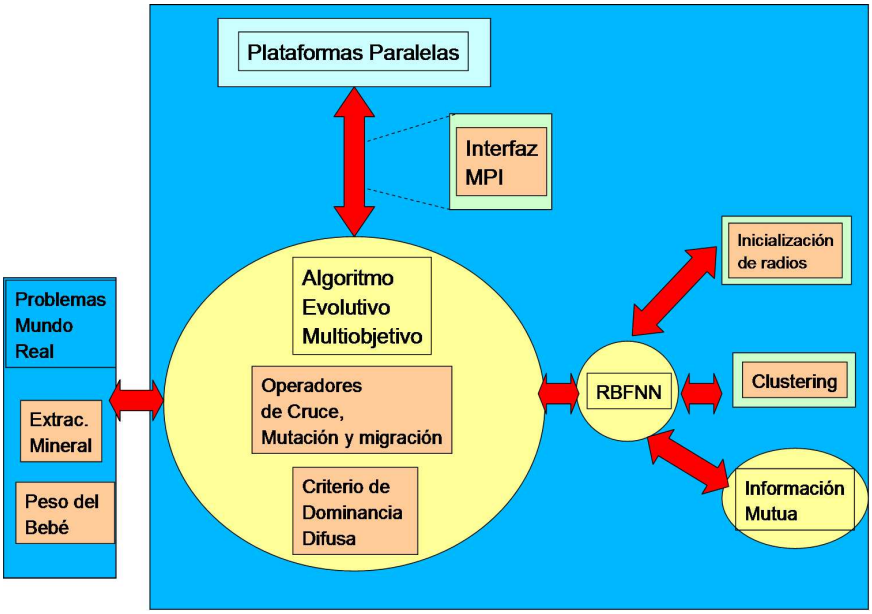


Fig. 1.1: Contenidos abarcados por la memoria.

ha propuesto un algoritmo que, partiendo de otro desarrollado en la bibliografía, obtiene considerables mejores tanto a nivel de calidad en las RBFNN diseñadas como de la eficiencia del algoritmo.

- Se ha realizado un estudio detallado de una de las mejoras más significativas con respecto al algoritmo propuesto con anterioridad, haciendo una interesante aportación a nivel de algoritmos de *clustering* para la inicialización de los centros de cada función de base radial (RBF).

- Además del aspecto de la inicialización de los centros de las RBF, también se ha considerado un aspecto poco estudiado en la bibliografía: la inicialización de los radios. La nueva metodología propuesta proporciona mejores resultados que las técnicas previas gracias a un mecanismo de supervisión.

- Dentro de la segunda metodología de diseño de las RBFNN mediante

optimización global, se ha presentado una adaptación de un prestigioso algoritmo multiobjetivo propuesto en la bibliografía. Esta adaptación ha comprendido el desarrollo de nuevos operadores de cruce así como un nuevo mecanismo para mantener un compromiso entre el tamaño de las redes y su error de aproximación.

- En lo referente a paralelismo, el algoritmo evolutivo propuesto para diseñar RBFNN ha sido paralelizado mediante una especialización en islas, permitiendo que cada una evolucione los distintos aspectos del diseño de RBFNN.

- A nivel de arquitecturas paralelas, para poder implementar y ejecutar el algoritmo en el mayor número posible de éstas, se ha desarrollado una nueva interfaz entre MATLAB y las bibliotecas de paso de mensajes, cuyo tiempo de sobrecarga en la comunicación es menor que otra interfaz desarrollada con anterioridad.

- En el campo de la interdisciplinariedad y de la investigación para la búsqueda de aplicaciones, se han encontrado dos problemas del mundo real sobre los que se ha podido ejecutar el algoritmo propuesto en esta memoria obteniendo resultados satisfactorios.

2. FUNDAMENTOS

Este capítulo describe el modelo de computación que permite resolver el problema expuesto en el Capítulo 1, es decir, la aproximación de una función a partir de un conjunto de entradas.

Tras describir el modelo se presentan distintas alternativas de diseño mediante las cuales, se pueden obtener buenos aproximadores. Las metodologías de diseño presentadas constituyen la base sobre la cual se ha desarrollado el trabajo innovador presentado en los siguientes capítulos.

2.1. Redes Neuronales de Funciones de Base Radial

Dentro del amplio abanico de tipos de redes neuronales, existe un tipo de red cuyo nombre completo es red neuronal de funciones de base radial (Radial Basis Function Neural Network, RBFNN)(Broomhead and Lowe, 1988). Una RBFNN se define matemáticamente como:

$$\mathcal{F}(\vec{x}_k; C, R, \Omega) = \sum_{i=1}^{m} \phi(\vec{x}_k; \vec{c}_i, r_i) \cdot \Omega_i \tag{2.1}$$

donde $C = \{\vec{c}_1, ..., \vec{c}_m\}$ son los centros de cada una de las RBF, $R = \{r_1, ..., r_m\}$ son los valores de los radios de las RBF, $\Omega = \{\Omega_1, ..., \Omega_m\}$ es el conjunto de los pesos correspondientes a cada una de las unidades de procesamiento y $\phi(\vec{x}_k; \vec{c}_i, r_i)$ representa a una RBF. La arquitectura de una RBFNN está representada gráficamente en la Figura 2.1 donde se puede apreciar cómo el vector de entrada es procesado por cada una de las RBF que constituyen la red para, posteriormente, proporcionar una salida resultante de la suma ponderada del resultado proporcionado por cada RBF. Dentro de las posibles funciones de activación de base radial, las funciones gausianas (Eq. 2.2), se muestran particularmente apropiadas para ser usadas por una red neuronal para aproximar funciones (Bors, 2001); (Rojas et al., 1998); (Poggio and Girosi, 1990); (Park and Sandberg, 1991b).

$$e^{-\frac{||\vec{x}_k - \vec{c}_i||^2}{r_i^2}}. \tag{2.2}$$

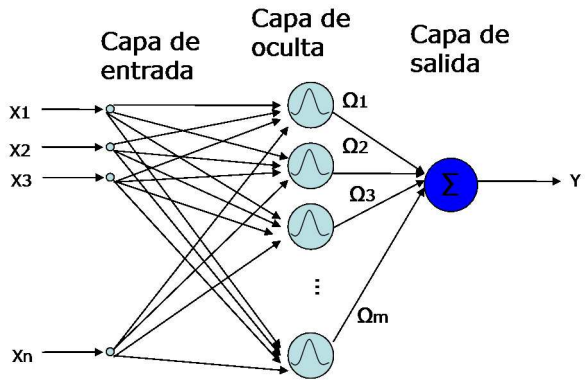


Fig. 2.1: Red Neuronal de Funciones de Base Radial

Aparentemente se ha resuelto el problema con el que trata la memoria, dada una aplicación que requiera aproximar una solución, sólo hay que diseñar una RBFNN que proporcione las salidas deseadas. El problema ahora reside en cómo diseñar una RBFNN que sea capaz de aproximar la función propuesta. Para poder crear una RBFNN, hay que conocer donde estarán centradas las RBF, cual será la amplitud de cada una, cuantas RBF deben componer la red y, por último, conocer los pesos que ponderan las salidas de las RBF. La razón de haber mencionado el cálculo de los pesos en último lugar no es casual puesto que es el único de todos los parámetros cuyo valor puede ser calculado de forma óptima: una vez establecidos los valores y el número de RBF, la obtención del valor de los pesos puede plantearse como un sistema de ecuaciones lineal entre la salida de la función a aproximar y la matriz de activación de las RBF (P):

$$\vec{y} = P\Omega \tag{2.3}$$

donde P es una matriz de dimension $n \times m$ en la que P_{ki} representa el valor de la i-ésima función gaussiana cuando se le proporciona como entrada el

vector de entrada $\vec{x}_k$. Hay muchas formas de resolver este sistema de ecuaciones. Entre las más usadas se encuentran la descomposición de Cholesky (Pomares et al., 2000), la descomposición en valores singulares (*Singular Vale Decomposition*, SVD) (Kanjilal and Banerjee, 1995) y el método de mínimos cuadrados ortogonales (*Orthogonal Least Squares*,OLS) (Chen et al., 1991).

Acotado el problema del diseño de una RBFNN a la búsqueda de los valores para los centros, radios y número de RBF, existen dos vertientes para calcular estos valores: optimización local y optimización global. Estas dos metodologías se presentan en las siguientes secciones, las cuales, asientan los cimientos sobre los que se ha apoyado esta memoria. .

2.2. Diseño de RBFNN Mediante Optimización Local

El proceso de optimización local consiste en (González et al., 2002):

- fijar el número de RBF,
- inicializar las posiciones de los centros utilizando un algoritmo de *clustering*,
- inicializar los valores de los radios usando una heurística que considere las posiciones de los centros,
- aplicar un algoritmo de búsqueda local que reduzca el error de aproximación de la red ajustando los radios y las posiciones de los centros.

A la hora de aplicar esta metodología, se puede fijar el número de RBF, diseñar la RBFNN y si el error final no es el deseado, se diseña de nuevo otra red con un número mayor de RBF. Una de las desventajas que presenta esta metodología es que, si se incrementa desmesuradamente el número de

RBF pueden aparecer problemas de sobreaprendizaje, es decir, que la red aprenda muy bien el conjunto de datos que se le muestra pero no sea capaz de generalizar y proporcionar una solución aceptable cuando procese un vector de entrada nuevo. Otro elemento en contra de esta técnica es lo dificultoso que puede ser hacer una correcta inicialización desde donde la búsqueda local comience ya que hay una alta probabilidad de caer en mínimos locales impidiendo obtener mejores soluciones.

Los centros de las funciones base de la red definen la ubicación de las mismas en el espacio de entrada. Cada una de ellas debería estar situada en una zona del espacio de entrada en la que haya ejemplos de E/S que la activen, de forma que contribuya a la salida de la red. Una función que no se active por ningún ejemplo de entrenamiento solamente contribuirá a aumentar la complejidad de la red inútilmente, además de producir un sistema de ecuaciones mal condicionado para el posterior cálculo de los pesos de la red.

Desde la aparición de las redes de funciones base radiales, la mayoría de los autores han usado algoritmos de *clustering* para inicializar sus centros. En (Moody and Darken, 1989), uno de los primeros trabajos sobre este tema, se utilizó el algoritmo de las C medias (*Hard C-means*, HCM). Posteriormente se han propuesto otros algoritmos de *clustering* más potentes que mejoran las redes iniciales de las que parte el proceso de entrenamiento, como el algoritmo de las C medias difuso (*Fuzzy C-means*, FCM) usado por Karayiannis en (Karayiannis and Mi, 1997) o el *mean-tracking clustering algorithm* propuesto en (Sutanto et al., 1997). Sin embargo, las técnicas de *clustering* fueron diseñadas inicialmente (Tryon, 1939) para resolver problemas de clasificación (Hartigan, 1975), y hay ciertas diferencias entre este tipo de problemas y el de la aproximación de funciones que deberían ser tenidas en cuenta:

1. En los problemas de clasificación, la variable de salida solamente puede

tomar valores en un intervalo de etiquetas válidas definido a priori (las clases o patrones), mientras que en los problemas de aproximación funcional, la función de salida puede tomar cualquier valor dentro de un intervalo de números reales.

2. En los problemas de aproximación de funciones es aceptable que el modelo artificial produzca una salida diferente de la esperada, siempre que aquella esté "suficientemente cerca" de ésta. Este tipo de variaciones solamente provocan pequeños incrementos del error de aproximación. En los problemas de clasificación no suele haber definido un operador de distancia entre las clases, es más, ni siquiera tienen que estar relacionadas entre sí, con lo que el asignación de una etiqueta equivocada para un ejemplo de entrada puede provocar un error inaceptable.

3. El objetivo de los algoritmos de aproximación funcional es la creación de un modelo que sea capaz de interpolar valores que no se han presentado durante el entrenamiento. Sin embargo, en un problema de clasificación la interpolación no tiene sentido, ya que las salidas del sistema deben restringirse al conjunto de clases aprendidas durante el proceso de entrenamiento.

Estas diferencias hacen necesaria la introducción de nuevas técnicas de *clustering* especialmente diseñadas para resolver problemas de aproximación funcional.A continuación se describe el algoritmo CFA (González et al., 2002), el cual fue especialmente diseñado para la inicialización de los centros de las RBF en problemas de aproximación funcional.

2.2.1. Inicialización de los centros: Clustering para problemas de aproximación funcional (CFA: Clustering for Function Approximation

Los algoritmos de *clustering* tradicionales solamente tienen en cuenta la disposición de los ejemplos de entrenamiento en el espacio de entrada. Por tanto, si se utilizan para inicializar un aproximador de funciones, colocarán más centros en las zonas del espacio de entrada en las que haya una mayor densidad de ejemplos, ignorando por completo la salida de la función objetivo. El algoritmo CFA (González et al., 2001a) incorpora la información referente a la salida esperada para cada vector de entrada del conjunto de entrenamiento. Esta información le permite realizar una inicialización supervisada de los centros, de forma que incrementa su densidad en aquellas zonas del espacio de entrada en las que la salida de la función objetivo es más variable, en vez de en aquellas en las que haya más vectores de entrenamiento. La primera forma de colocar los centros es más adecuada en problemas de aproximación funcional, ya que trata de minimizar la varianza no explicada por el modelo, mientras que la segunda, típica de los algoritmos de *clustering* tradicionales, no tiene demasiado sentido en este tipo de problemas.

En problemas de aproximación funcional se pueden cubrir con un solo centro las zonas del espacio de entrada en las que la función objetivo es constante, independientemente del número de ejemplos que las representen en el conjunto de entrenamiento, mientras que otras zonas las en las que la función objetivo es más variable, quedarían mejor descritas si se incrementara el número de centros que las resumen.

Como el resto de los algoritmos de *clustering*, el algoritmo CFA revela la estructura subyacente de los datos en el espacio de entrada, pero de forma que se preserve la homogeneidad de las salidas de los ejemplos pertenecientes a un

mismo *cluster*. Para llevar a cabo esta tarea, el algoritmo CFA incorpora un conjunto $O = \{o_1, ..., o_m\}$ que representa una estimación de la salida esperada para los puntos que pertenezcan a un mismo *cluster*. El valor de o_i se obtiene como una media ponderada de las salidas de los ejemplos de entrenamiento pertenecientes al *cluster* Cl_i. La función de distorsión a minimizar en este caso se define como:

$$\varepsilon = \frac{\sum_{i=1}^{m} \sum_{\vec{x}_k \in Cl_i} \|\vec{x}_k - \vec{c}_i\|^2 \omega_{ik}}{\sum_{i=1}^{m} \sum_{\vec{x}_k \in Cl_i} \omega_{ik}} \tag{2.4}$$

donde ω_{ik} pondera la influencia de cada ejemplo de entrenamiento $\vec{x}_k$ en la posición final del centro i-ésimo. Este índice informa sobre la controversia existente entre el valor de salida que toma el centro y el valor de salida del ejemplo $\vec{x}_k$. Cuanto mayor sea la distancia entre la salida esperada para $\vec{x}_k$ y la salida estimada para el *cluster* Cl_i al que pertenezca, mayor será su influencia en el resultado final. Matemáticamente, ω_{ik} se define como

$$\omega_{ik} = \frac{|F(\vec{x}_k) - o_i|}{\max\limits_{k=1}^{n} \{F(\vec{x}_k)\} - \min\limits_{k=1}^{n} \{F(\vec{x}_k)\}} + \vartheta_{\text{mín}}, \quad \vartheta_{\text{mín}} > 0 \tag{2.5}$$

El primer sumando de esta expresión calcula una distancia normalizada (en el intervalo [0,1]) entre $F(\vec{x}_k)$ y o_i, mientras que el segundo sumando es un umbral de mínima contribución (cuando no exista controversiaω_{ik} tomará el valor $\vartheta_{\text{mín}}$). A medida que el valor de $\vartheta_{\text{mín}}$ decrece, el algoritmo CFA fuerza a los centros a concentrarse en las zonas del espacio de entrada en las que la función objetivo sea más variable. Esta acción preserva la homogeneidad en la salida de los ejemplos $\vec{x}_k$ pertenecientes a un mismo *cluster* Cl_i, ya que se está dando más importancia a la minimización de la distancia entre $F(\vec{x}_k)$ y o_i en la función de distorsión. Por otro lado, si el valor de $\vartheta_{\text{mín}}$ crece, el

algoritmo CFA empieza a dar más importancia a la distancia entre $\vec{x}_k$ y $\vec{c}_i$, adquiriendo el comportamiento de un algoritmo de *clustering* tradicional.

La organización básica del algoritmo CFA consiste en la iteración de tres pasos: La partición del conjunto de puntos de entrenamiento, la actualización de los centros y sus salidas estimadas, y la migración de centros desde *clusters* con una distorsión baja hacia *clusters* con una gran distorsión.

2.2.1.0.1. *Partición de los ejemplos de entrenamiento.*

La partición de los ejemplos de entrenamiento se realiza mediante la aplicación de la siguiente función de pertenencia:

$$\vartheta(\vec{x}_k, \vec{c}_i) = \begin{cases} 1 & \text{si } \|\vec{x}_k - \vec{c}_i\|^2 < \|\vec{x}_k - \vec{c}_j\|^2 \quad \forall i \neq j \\ 0 & \text{en otro caso} \end{cases} \tag{2.6}$$

2.2.1.0.2. *Actualización de los centros y sus salidas esperadas*

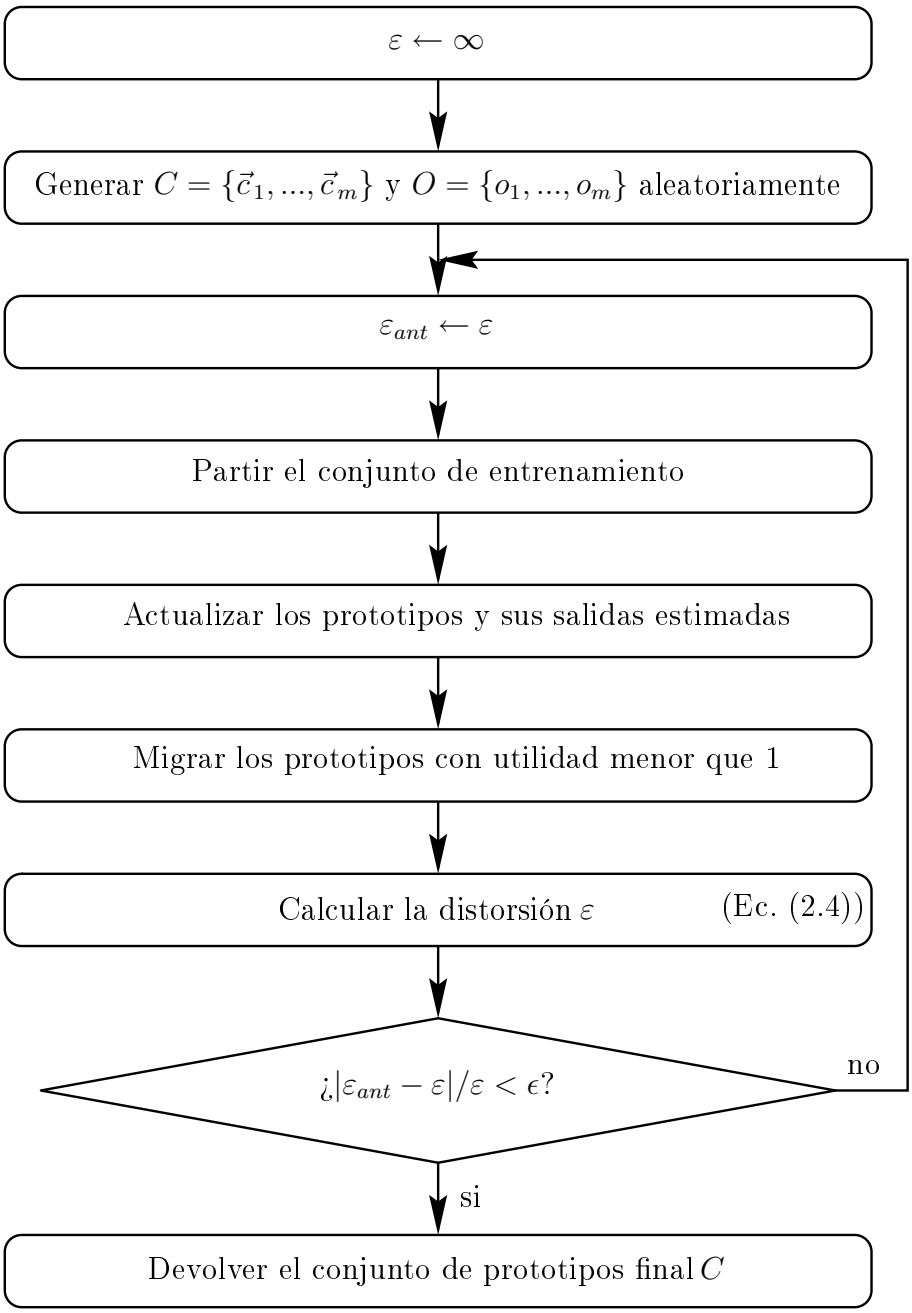
Una vez que todos los vectores de entrada han sido asignados a algún *cluster* Cl_i, se deben actualizar los centros y la estimación de sus salidas. Esta actualización se lleva a cabo mediante un proceso iterativo que calcula los valores de $\vec{c}_i$ y o_i como una media ponderada de los vectores de entrenamiento pertenecientes al *cluster* Cl_i y sus salidas respectivamente:

$$\vec{c}_i = \frac{\sum\limits_{\vec{x}_k \in Cl_i} \vec{x}_k \omega_{ik}}{\sum\limits_{\vec{x}_k \in Cl_i} \omega_{ik}} \tag{2.7}$$

$$o_i = \frac{\sum\limits_{\vec{x}_k \in Cl_i} F(\vec{x}_k) \omega_{ik}}{\sum\limits_{\vec{x}_k \in Cl_i} \omega_{ik}} \tag{2.8}$$

y el algoritmo que lleva a cabo la actualización es:

$\varepsilon' \leftarrow \infty$



Alg. 2.1: Algoritmo CFA

repetir

$\varepsilon'_{ant} \leftarrow \varepsilon'$

Actualizar $\vec{c}_i$ usando (2.7), $\quad i = 1, ..., m$

Actualizar o_i usando (2.8), $\quad i = 1, ..., m$

Actualizar ω_{ik} usando (2.5), $\quad k = 1, ..., n, \quad i = 1, ..., m$

Calcular ε' usando (2.4)

hasta $|\varepsilon'_{ant} - \varepsilon'|/\varepsilon' < \epsilon$

La actualización de los centros tiene que ser un proceso iterativo porque $\vec{c}_i$ y o_i son interdependientes. Para la actualización de $\vec{c}_i$ se usa el valor anterior de o_i, por tanto, el proceso debe iterar hasta que se alcance el equilibrio y $\vec{c}_i$ converja a la posición correcta para cada *cluster* Cl_i. Este estado se alcanza cuando la distorsión de actualización ε' no varía significativamente en dos iteraciones sucesivas.

2.2.1.0.3. Migración de centros

El algoritmo CFA también incorpora un paso de migración aleatoria para evitar la convergencia hacia mínimos locales. Este proceso desplaza a los centros que se encuentren en zonas del espacio de entrada en las que la función objetivo sea estable hacia zonas con una variabilidad alta en la salida, haciendo que el resultado final sea independiente de la configuración inicial. Para detectar dichas zonas, el proceso de migración se basa en el concepto de utilidad de un centro, definida como:

$$u_i = \frac{\varepsilon_i}{\bar{\varepsilon}} \quad \forall i = 1, ..., m \tag{2.9}$$

donde ε_i es la contribución del i-ésimo *cluster* a la distorsión total:

$$\varepsilon_i = \frac{\sum\limits_{\vec{x}_k \in Cl_i} \|\vec{x}_k - \vec{c}_i\|^2 \omega_{ik}}{\sum\limits_{i=1}^{m} \sum\limits_{\vec{x}_k \in Cl_i} \omega_{ik}} \tag{2.10}$$

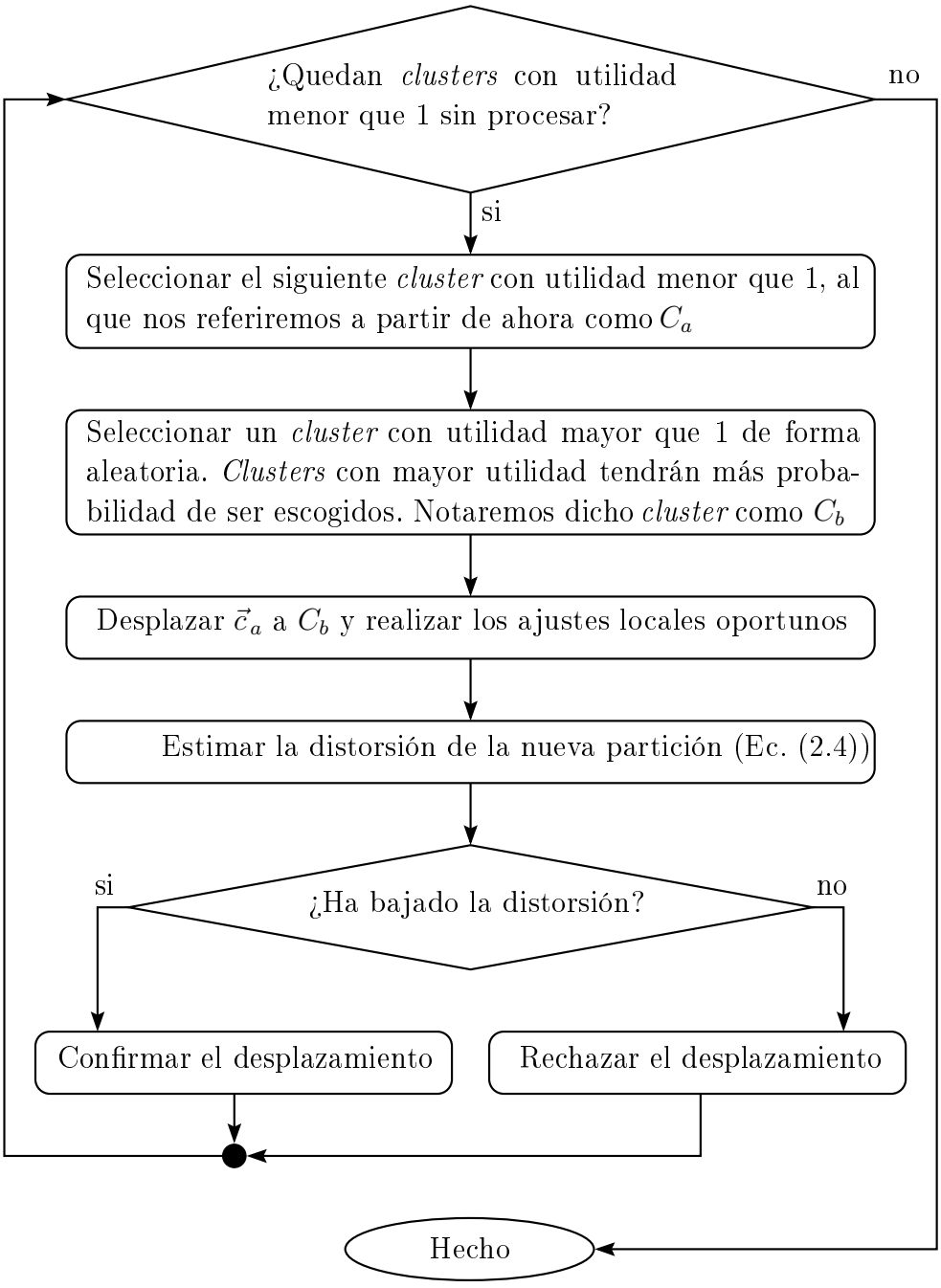
y $\bar{\varepsilon}$ se define como la distorsión media de la partición:

$$\bar{\varepsilon} = \frac{1}{m} \sum_{i=1}^{m} \varepsilon_i = \frac{\varepsilon}{m} \tag{2.11}$$

En una cuantización óptima del conjunto de entrenamiento, cada *cluster* contribuye en igual medida a la distorsión total (Gersho, 1979). Esto significa que el objetivo final se transforma en encontrar una configuración en la que todos los centros tengan una utilidad igual a 1. Por tanto, el proceso de migración, como muestra el algoritmo 2.2, desplaza a los centros$\vec{c}_a$ con utilidad menor que 1 hacia *clusters* con una utilidad mayor que 1. Cuanto mayor sea la utilidad de un *cluster*, más probabilidad tendrá de ser seleccionado para recibir a un centro con utilidad menor que 1.

Los centros $\vec{c}_a$ y $\vec{c}_b$ se colocan en la diagonal principal del paralelepípedo que contiene a Cl_b. Dicha diagonal se divide en tres segmentos, de forma que el segmento central tenga una longitud igual al doble de la de los otros dos segmentos. Después se colocan los centros en los extremos del segmento central.

Tras mover el centro $\vec{c}_a$ al *cluster* Cl_b, se deben realizar ciertos ajustes a los *clusters* afectados. Los centros $\vec{c}_a$ y $\vec{c}_b$ se inicializan aplicando el algoritmo HCM con $m = 2$ de forma local al *cluster* Cl_b, dividiéndolo en dos nuevos *clusters* Cl'_a y Cl'_b. En cuanto a los vectores del *cluster* Cl_a, que se ha quedado sin centro, son cedidos al *cluster* Cl_c más cercano a Cl_a, que pasa a llamarse Cl'_c. Una vez creados los nuevos *clusters* Cl'_a, Cl'_b y Cl'_c, se ejecuta el proceso de actualización (sección 2.2.1.0.2) sobre estos tres *clusters* para obtener sus



Alg. 2.2: Migración de prototipos en al algoritmo CFA

centros y sus salidas estimadas. Tras realizar todos estos ajustes locales, el desplazamiento realizado se acepta solo si la distorsión de la partición obtenida es menor que la de la partición original.

2.2.1.0.4. Efecto del umbral $\vartheta_{\text{mín}}$

El parámetro $\vartheta_{\text{mín}}$ controla la forma en que el algoritmo CFA trata a la variabilidad de la función objetivo. Un valor alto para este parámetro implica que el algoritmo preste más atención al espacio de entrada que al de salida, y conforme su valor va bajando, la variabilidad de la función objetivo empieza a tenerse cada vez más en cuenta.

La tabla 2.1 muestra la media y la desviación estándar de los errores de aproximación obtenidos por CFA, y de los tiempos de computación necesarios para encontrar la configuración final. Como se puede comprobar, cuanto menor es el valor del umbral $\vartheta_{\text{mín}}$, mejor es la aproximación obtenida. Sin embargo, conforme este parámetro decrece, el tiempo de computación crece drásticamente mientras que el error de aproximación solo sufre pequeños decrementos. Empíricamente se ha establecido un valor igual a 0.001 ya que mantiene un equilibrio entre la calidad de los resultados y el tiempo de ejecución.

2.2.2. Inicialización de los radios de las funciones base de la red

Una vez que se han inicializado los centros de las funciones base de la red, el siguiente paso en el diseño es encontrar unos valores iniciales adecuados para los radios o amplitudes de las funciones base, de forma que no se quede sin cubrir ningún punto del espacio de entrada.

Aunque el uso de la misma amplitud para todas las funciones base de la red se ha probado suficiente para obtener un aproximador universal (Park

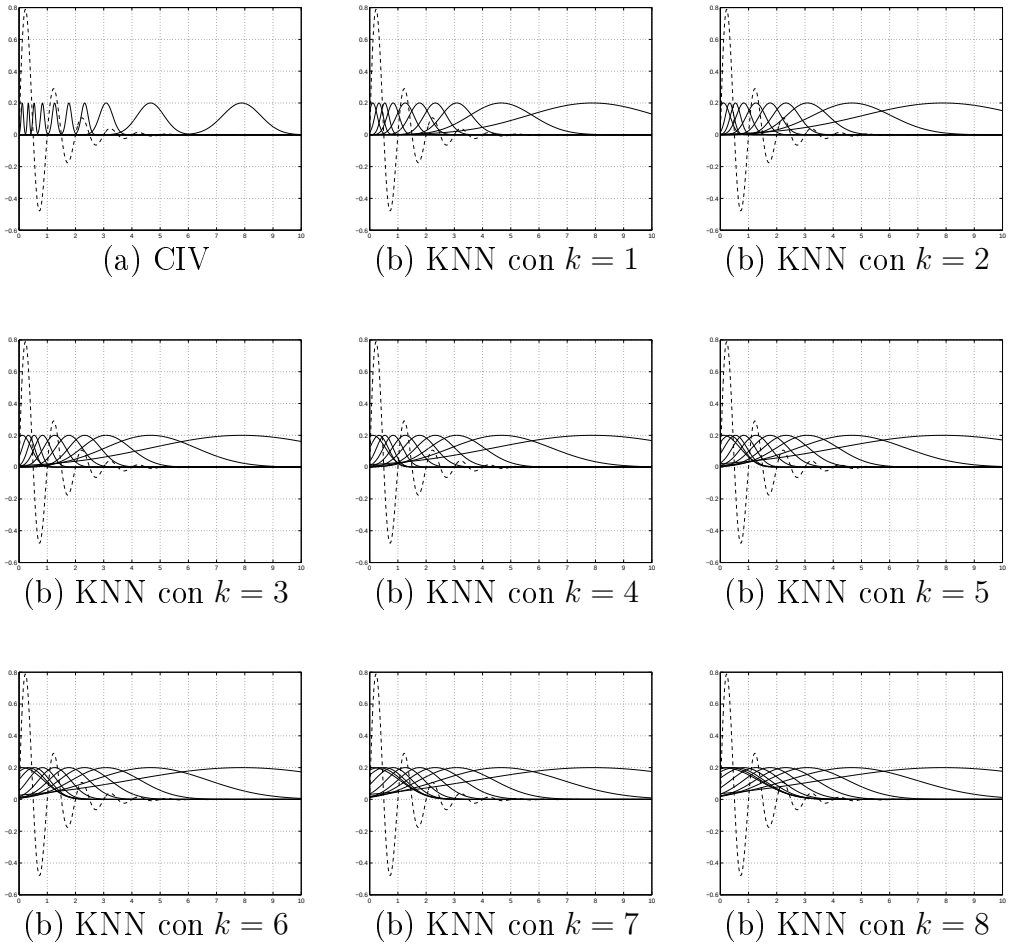
$\vartheta_{\text{mín}}$	NRMSE	Tiempo
10000	0.9641 ± 0.0047	0.5169 ± 0.2159
1000	0.9645 ± 0.0051	0.6031 ± 0.2694
100	0.9644 ± 0.0039	0.5039 ± 0.2024
10	0.9628 ± 0.0041	0.5330 ± 0.2537
1	0.9560 ± 0.0042	0.5921 ± 0.1957
0.1	0.9416 ± 0.0085	1.3134 ± 0.4258
0.01	0.8457 ± 0.0921	8.8414 ± 3.7841
0.001	0.6646 ± 0.1382	56.332 ± 39.640
0.0001	0.6469 ± 0.1281	359.39 ± 255.71

Tab. 2.1: Media y desviación estándar del NRMSE y el tiempo de computación (en segundos) necesitado para aproximar la función f_1 para distintos valores del umbral $\vartheta_{\text{mín}}$

and Sandberg, 1991a); (Park and Sandberg, 1993b), si cada función base tiene la posibilidad de tener un radio individual, se mejoran las prestaciones de la red (Musavi et al., 1992b). En la literatura se proponen diversas heurísticas (Karayiannis and Mi, 1997); (Moody and Darken, 1989) para fijar sus valores. El objetivo es conseguir un cierto grado de solapamiento entre las funciones vecinas para que interpolen de forma suave y continua las regiones del espacio de entrada que representan. Entre las heurísticas más populares se encuentran la de los *k vecinos más cercanos* (Moody and Darken, 1989) y la de la *distancia media de los vectores de entrada más cercanos*(Karayiannis and Mi, 1997), descritas a continuación.

2.2.2.1. Heurística de los K vecinos más cercanos (KNN)

Esta heurística fija el radio de acción de cada función base a un valor igual a la distancia media a los centros de sus K funciones base más cercanas (Moody and Darken, 1989). Esta heurística pasa a ser la heurística del vecino más cercano si $K = 1$. Como se puede observar en la Figura 2.2, conforme



(a) CIV (b) KNN con $k = 1$ (b) KNN con $k = 2$

(b) KNN con $k = 3$ (b) KNN con $k = 4$ (b) KNN con $k = 5$

(b) KNN con $k = 6$ (b) KNN con $k = 7$ (b) KNN con $k = 8$

Fig. 2.2: Diferentes posibilidades para los radios de las funciones base de una red una vez que se han fijado sus centros mediante el algoritmo CFA

se aumenta el valor de K, se incrementa el grado de solapamiento de las funciones base de la red.

2.2.2.2. Heurística de la distancia media de los vectores de entrada más cercanos (CIV)

Esta heurística (Karayiannis and Mi, 1997) determina el radio de la función base radial i-ésima de acuerdo con la siguiente ecuación:

$$r_i = \frac{\sum_{\vec{x}_k \in V_i} \|\vec{x}_k - \vec{c}_i\|}{|V_i|} \tag{2.12}$$

donde V_i es el conjunto de vectores de entrada que están más cerca de $\vec{c}_i$ que de cualquier otro centro y $|V_i|$ representa el número de vectores que lo componen.

Como se puede observar en la Figura 2.2, esta heurística produce menor solapamiento que la anterior, independientemente del valor que tome K.

2.3. Optimización Global: Algoritmos Evolutivos para el Diseño de RBFNN

El concepto de algoritmo evolutivo surge como la adaptación de un algoritmo genético a un problema determinado. Los algoritmos genéticos tal y como los definió Holland en (Holland, 1975) sufren de una serie de debilidades, como la representación de soluciones mediante cadenas de bits y su falta de especificidad. A raíz de la introducción de los algoritmos genéticos, muchos autores (Davis, 1991); (Michalewicz, 1996); (Wright,) han ido adaptando estos algoritmos, en principio genéricos, a problemas particulares, cambiando la representación de las soluciones por otra más natural y cercana al problema y añadiendo operadores específicos y heurísticas para ayudar en el proceso de búsqueda. El único problema que se plantea es que la única demostración formal de convergencia que existe se ha realizado para algoritmos genéticos clásicos, con representación binaria y operadores genéricos (Holland, 1975). Sin embargo, usando una representación más cercana al problema se obtienen mejores resultados en la práctica (Bäck et al., 1997a); (Bäck et al., 1997b); (Davis, 1991); (Michalewicz, 1996); (Wright,). En (Antonisse, 1989) se demuestra que el paralelismo implícito no depende del alfabeto usado para la

representación y se sugiere experimentar con alfabetos de mayor cardinalidad. Por otra parte, parece lógico pensar que un algoritmo obtendrá mejores resultados cuanto más cerca esté del problema a resolver e incorpore más conocimiento específico sobre el mismo.

En (Michalewicz, 1996) se introduce el concepto de individuo para referirse a una solución para un problema codificada mediante una estructura de datos natural y cercana a dicho problema. También se define el concepto de algoritmo evolutivo como un algoritmo genético en el que se habrá cambiado la representación de las soluciones por la estructura de datos más conveniente para su procesamiento y evaluación y en el que se aplicarán operadores evolutivos que tratarán de explotar al máximo la información almacenada en cada individuo mediante el uso de conocimiento específico sobre el problema, manteniendo el resto de características de los algoritmos genéticos clásicos. Estos algoritmos disfrutan de todas las ventajas de los algoritmos genéticos, resuelven sus problemas de precisión, control del efecto de las mutaciones, ajuste local e incorporación de restricciones y además añaden heurísticas y conocimiento específico para ayudar en la búsqueda del óptimo global.

A continuación se presentan los distintos componentes de un algoritmo evolutivo cuyos operadores de cruce y mutación fueron propuestos en (González et al., 2003) para evolucionar específicamente RBFNN que aproximen funciones.

2.3.1. Estructura de los individuos

Los algoritmos genéticos fijan la representación de las posibles soluciones del problema a cadenas de bits. Para problemas complejos, el uso de esta representación supone la construcción de codificadores y decodificadores que conviertan la representación natural de la solución a cadenas de bits y vice-

versa. Además, una vez que la solución está representada como una cadena de bits, en la mayoría de los casos no es una tarea fácil detectar si dicha cadena representa una solución que cumpla las restricciones del problema o bien, qué efectos puede causarle la aplicación de un operador de cruce o mutación clásico que opere a nivel de bits. Otro serio inconveniente es la imposibilidad de obtener soluciones con una alta precisión.

Para codificar una red se emplea una estructura de datos la cual almacena todos los parámetros que definen a una RBFNN utilizando codificación real.

2.3.2. *Los operadores evolutivos*

El diseño de unos operadores evolutivos que sean capaces de usar conocimiento específico del problema proporcionan una serie de ventajas, como la de generar soluciones que superen a las actuales mediante la aplicación de alguna heurística o la incorporación de restricciones para evitar la generación de soluciones que representen una red no válida (Michalewicz, 1996).

Por otra parte, en el problema que estamos abordando, así como en la mayoría de problemas reales, aparecen restricciones sobre algunos valores de los parámetros a optimizar para que las soluciones obtenidas sean de utilidad:

- Las funciones base deben activarse, las funciones base que no se activen no aportarán nada a la salida de la red.
- No debe haber dos funciones base solapadas en la misma zona del espacio de entrada, ya que estarían realizando el mismo trabajo. Una de ellas se podría eliminar y se obtendría una red más simple que daría los mismos resultados que la original (Rojas et al., 2000).

Los algoritmos evolutivos no tienen dificultades para enfrentarse a este tipo de problemas, sólo hay que añadirles un mecanismo para manejar las

restricciones. En la literatura han aparecido diversos enfoques para llevar a cabo esta tarea. El más drástico fue propuesto por Fogel en (Fogel, 1992), y consistía en eliminar de la población las soluciones no válidas. Este enfoque no es el más adecuado, ya que en los problemas de optimización no lineal con restricciones los óptimos globales suelen estar en los extremos del espacio de soluciones, rodeados de soluciones no válidas (Hilliard et al., 1989). Por tanto, al eliminar las soluciones no válidas, estamos evitando también que en una generación futura se pueda generar una buena solución a partir de éstas.

Otra alternativa consiste en corregir las soluciones que no cumplan las restricciones. El problema reside en que el proceso de reparación de soluciones suele ser muy costoso, por lo que este enfoque se usa raras veces en la práctica (Tate and Smith, 1993). Otros autores (Homaifar et al., 1994); (Hilliard et al., 1989) proponen penalizar la aptitud de las soluciones no válidas.

Sin embargo, en (Michalewicz, 1996) se propone el diseño de operadores que no generen soluciones no válidas. Siguiendo esta última filosofía, en (González et al., 2003) se propusieron los operadores de cruce y mutación expuestos a continuación, los cuales, hacen uso del conocimiento específico del problema para guiar la búsqueda hacia zonas del espacio de soluciones prometedoras.

2.3.2.1. Cruce de redes

Este operador de cruce (González et al., 2000) toma dos redes del conjunto de reproductores y genera otras dos redes que combinan la información contenida en sus progenitores. El cruce se realiza a nivel de funciones base. Se seleccionan aleatoriamente un número de funciones base de uno de los progenitores y se intercambian por el mismo número de funciones base del otro progenitor, generando dos descendientes como indica la Figura 2.3.

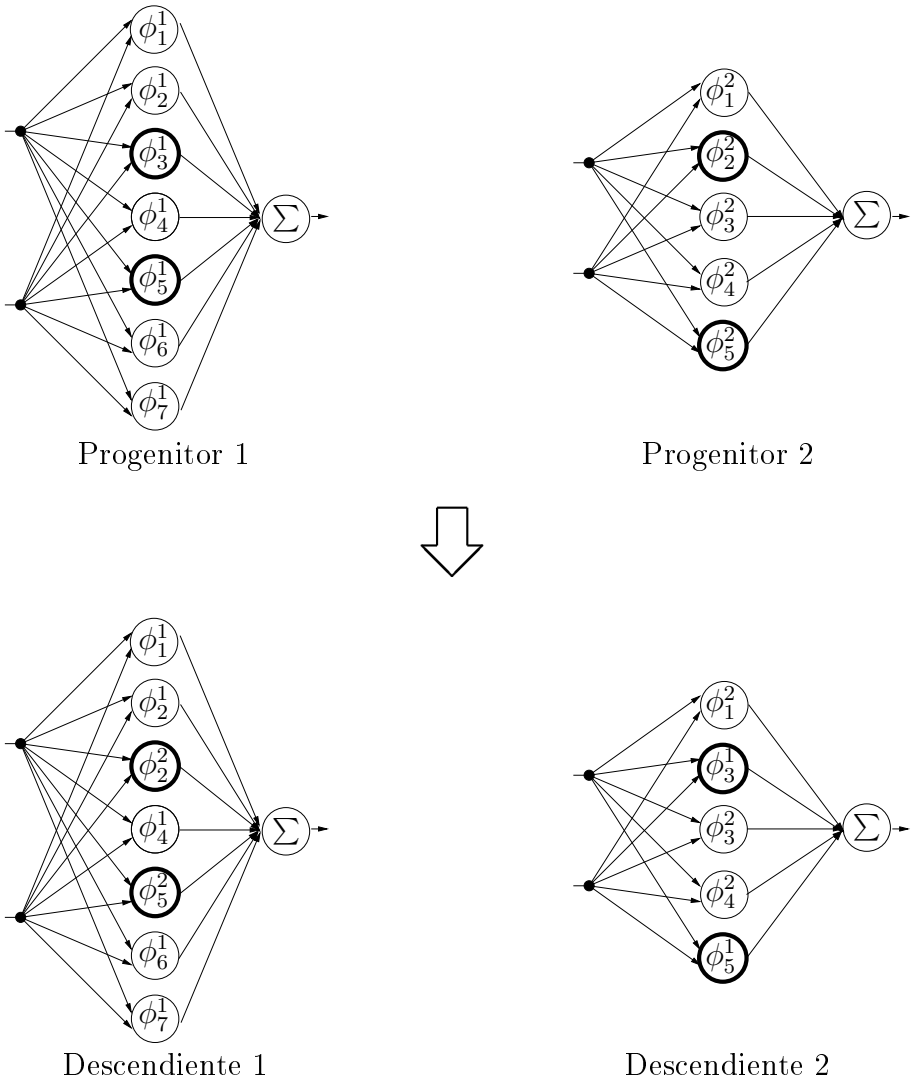


Fig. 2.3: Cruce de redes

La selección de las funciones que se van a intercambiar se hace de forma aleatoria en el primer progenitor, pero no en el segundo. De esta forma se evita que al realizar el intercambio se produzcan redes en las que exista demasiado solapamiento de funciones base en las mismas zonas del espacio de entrada mientras que otras zonas se quedan sin cubrir. En la Figura 2.5 se puede observar cómo el resultado de cruzar las redes de la Figura 2.4 escogiendo las funciones base de forma aleatoria no siempre da buenos resultados, ya que partiendo de dos progenitores que cubren perfectamente todo el espacio de entrada, estamos generando dos descendientes en los que se puede apreciar claramente cómo se ha incrementado el solapamiento en algunas zonas y se ha dejado de cubrir otras, con lo que los puntos del conjunto de entrenamiento que estén ubicados en dichas zonas incrementarán gravemente el error de aproximación de las nuevas redes.

Para evitar este tipo de resultados indeseables, se propuso una solución consistente en intercambiar las funciones base seleccionadas del primer progenitor por aquellas del segundo progenitor cuyos centros estén más cerca en el espacio de entrada. De esta forma es seguro que las zonas del espacio de entrada que cubrían las funciones seleccionadas para el cruce en la primera red serán cubiertas o parcialmente cubiertas por las funciones de la segunda red que las sustituyan y viceversa. En la Figura 2.6 se puede observar un resultado típico de un cruce incorporando esta restricción.

2.3.2.2. Operadores de mutación

Esta sección presenta varios operadores de mutación específicos para redes de funciones base radiales desarrollados en (González et al., 2003). Todos estos operadores aplican cambios aleatorios a las soluciones para ayudar al mantenimiento de una población diversa y facilitar el escape de mínimos lo-

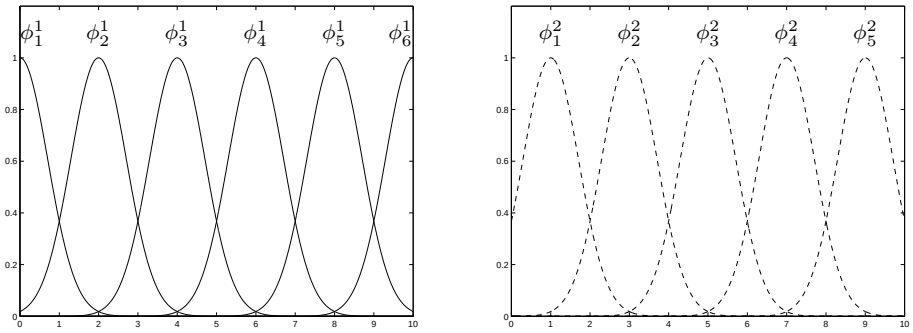


Fig. 2.4: Funciones base de las dos redes seleccionadas para el cruce

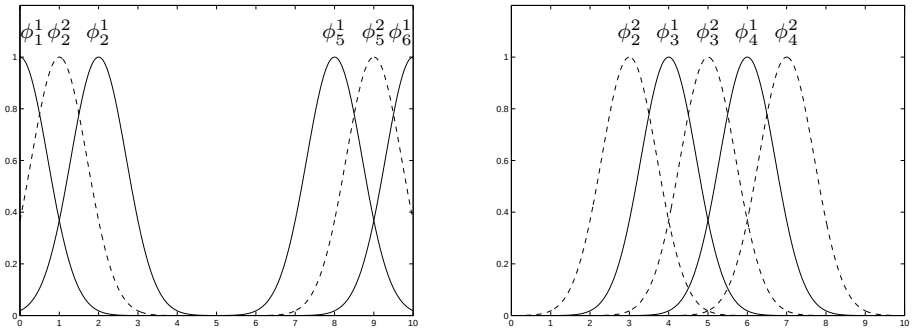


Fig. 2.5: Descendientes tras un cruce aleatorio de las funciones base

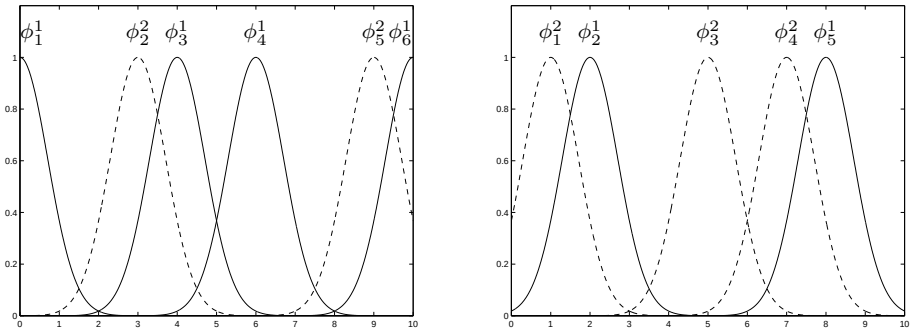


Fig. 2.6: Descendientes tras el cruce propuesto

cales (Goldberg, 1989a); (Holland, 1975). Esto se obtiene mediante el empleo de alguna heurística o la aplicación de cambios de forma ciega.

2.3.2.2.1. MLA (Mutación Localmente Aleatoria de redes).

Si no se dispone de información sobre el problema que se pretende resolver, la opción más sencilla es construir un operador que aplique cambios completamente aleatorios a las soluciones a las que afecte. Es decir, todos los parámetros que definan al individuo tendrán la misma probabilidad de ser alterados, y cuando se escoja uno, se le aplicará una modificación totalmente aleatoria, siempre que se mantengan las restricciones impuestas para que el individuo represente a una solución válida (Michalewicz, 1996). Los operadores de este tipo realizan cambios a las soluciones de una forma totalmente ciega, ya que dan igual probabilidad a la alteración de cada parámetro, y a su vez, una vez escogido el parámetro, le aplican una modificación aleatoria uniforme. Esta opción es una forma directa de adaptar el operador clásico de mutación de los algoritmos genéticos (Holland, 1975) a un problema determinado, ya que altera a las soluciones usando la misma filosofía, que es simplemente la generación de nuevos individuos para ampliar el espacio de soluciones explorado por el algoritmo y facilitar el escape de óptimos locales.

Hay muchas posibilidades para construir un operador de este tipo aplicado al problema del diseño de RBFNN, ya que una red está definida por un gran número de parámetros libres. Los pasos que va a seguir este operador son:

- Decidir qué función base va a ser alterada. Esta decisión se toma mediante una distribución aleatoria uniforme en la que cada función base tiene la misma probabilidad de ser escogida.

- Una vez que se escoge la función base, se debe decidir entre alterar

su centro o su radio. Esta segunda elección se hace aleatoriamente con igual probabilidad para ambas posibilidades.

- Realizar la alteración y aplicar el método de Cholesky para obtener los pesos óptimos para la nueva red (González et al., 2000); (González et al., 2001b).

2.3.2.2.1.1. Alteración local aleatoria del centro de una función base

La modificación del centro de una función base se realiza aplicándole un vector desplazamiento con un módulo menor que su radio para mantener la localidad de las funciones de la red:

$$(\vec{c}_i)' = \vec{c}_i + \vec{d}, \quad (|\vec{d}| \leq r_i) \tag{2.13}$$

Cada una de las componentes del vector desplazamiento $\vec{d}$ se escoge de la siguiente forma

$$d_j = \text{signo}() \cdot \text{u}(0, r_i) \tag{2.14}$$

donde la función signo() se define como

$$\text{signo}() = \begin{cases} 1 & \text{si u}(0,1) < 0{,}5 \\ -1 & \text{en otro caso} \end{cases} \tag{2.15}$$

y la función $\text{u}(a, b)$ devuelve un valor real perteneciente al intervalo $[a, b)$ escogido aleatoriamente de forma uniforme.

2.3.2.2.1.2. Alteración local aleatoria del radio de una función base

Al igual que en el caso de los centros, la alteración aleatoria del radio de una función base se realiza mediante la aplicación de un desplazamiento aleatorio a su radio

$$(r_i)' = r_i + \text{signo}() \cdot \text{u}(0, r_i) \tag{2.16}$$

De esta forma cada radio puede crecer o decrecer de una forma adecuada a su valor absoluto.

2.3.2.2.2. MME (Mutación hacia el punto de Mayor Error).

Aunque el operador de mutación anterior realiza cambios totalmente aleatorios en centros y radios, usa algo de conocimiento sobre el problema, ya que trata de que las redes producidas estén compuestas por funciones base que se puedan activar por los ejemplos de entrenamiento.

Otra posibilidad para construir un operador de mutación es intentar cubrir el punto de entrenamiento en el que se ha producido un mayor error en la pasada evaluación, $\vec{x}_{e_{\text{máx}}}$. Para ello, como primer paso se calcula el vector de distancias de los centros de todas las funciones base al punto $\vec{x}_{e_{\text{máx}}}$ y construye una distribución de probabilidades inversamente proporcionales a estas distancias, de forma que la probabilidad de escoger una función base para trasladarla al punto de máximo error sea mayor cuanto más cerca se encuentre de dicho punto. Una vez creada la distribución de probabilidad, se escoge aleatoriamente una función base y la traslada al punto $\vec{x}_{e_{\text{máx}}}$.

2.3.2.2.3. MLSVD (Mutación Local de redes usando SVD).

La descomposición SVD proporciona un vector de valores singulares que están asociados a cada una de las funciones base de la red y que determinan su relevancia en la salida, siendo altos los valores singulares relacionados con funciones base importantes y bajos o nulos los asociados con funciones base que se activan poco o bien que se encuentran solapadas con otras.

Estos valores singulares dan una idea de la sensibilidad al movimiento de las funciones base de la red. Si se mueve una función base importante (con un valor singular alto) es posible que empeore el error de aproximación de la red, mientras que si se modifica una función base con un valor singular casi nulo, probablemente mejoren las prestaciones de la red, ya que un valor singular cercano a cero implica que la función base no influye demasiado en la salida de la red, y por tanto, se puede ubicar en alguna posición donde pueda contribuir en mayor medida a la minimización de su error de aproximación.

Teniendo en cuenta esta idea, este operador de mutación seleccionará la función base que será mutada con una probabilidad inversamente proporcional a su valor singular asociado, dando mayor probabilidad de ser mutadas a las funciones base menos importantes y protegiendo de los cambios a las funciones base más sensibles (González et al., 2000); (González et al., 2001b).

Una vez escogida la función base, se le aplicará una modificación local a su centro o a su radio con igual probabilidad.

2.3.2.2.4. MLOLS (Mutación Local de redes usando OLS)

Este método se diferencia del SVD en que tiene en cuenta el valor de la función objetivo para cada vector del conjunto de entrenamiento, con lo que la valoración de la importancia de cada función base de la red depende de su contribución a la reducción del error de aproximación. Ya que esta

es precisamente la medida que se está usando para estimar la bondad del ajuste de los parámetros realizada por el algoritmo evolutivo, parece una buena heurística para decidir qué funciones base se pueden o no modificar en función de su aportación a la reducción del error de aproximación de la red. Aquellas funciones que contribuyan en mayor medida a la reducción del error de aproximación serán más sensibles al movimiento, y una pequeña alteración en una de ellas puede producir que el error de aproximación se incremente de forma impredecible. Sin embargo, las funciones que no contribuyan en nada o contribuyan poco a la reducción del error, tendrán más libertad para cambiar su ubicación sin afectar tanto al error de aproximación de la red.

El método OLS calcula un vector de coeficientes de reducción del error rre en el que habrá una componente asociada a cada función base de la red. Cuanto mayor sea la componente $[rre]_i$, más sensible al movimiento será su función base asociada ϕ_i. Por tanto, para escoger la función base que se va a alterar, se construye una distribución de probabilidad en la que cada función base tendrá una probabilidad de ser alterada inversamente proporcional a su coeficiente de reducción del error $[rre]_i$ (González et al., 2000); (González et al., 2001b). Después se escoge aleatoriamente una función base usando la distribución de probabilidades generada y se procede a alterar localmente su centro o su radio.

2.3.2.2.5. MGSVD (Mutación Global de redes usando SVD)

Otra forma de usar el algoritmo SVD para implementar un operador de mutación guiado puede ser la siguiente: cuanto menor sea el valor singular asociado a una función base, mayor será su libertad de movimiento (González et al., 2001b).

Una posible implementación de este operador consiste en permitir un

desplazamiento de una función base o un cambio en su radio inversamente proporcional a su valor singular asociado. La función base que sufrirá la modificación se escoge aleatoriamente, teniendo todas las funciones base la misma probabilidad de ser escogidas. Una vez seleccionada la función base, se le aplicará una modificación en su radio o en su posición inversamente proporcional a su relevancia en la red. Estas modificaciones se describen en detalle a continuación.

2.3.2.2.5.1. Alteración aleatoria global del centro de una función base según su valor singular asociado

Al igual que en los operadores de mutación anteriores, la modificación del centro de una función base se realiza aplicándole un vector desplazamiento. La diferencia reside en que en esta ocasión no se hace de forma local. El módulo del desplazamiento será mayor cuanto menor sea el valor singular asociado a la función base. De esta forma se consigue que las funciones base poco relevantes de la red tengan libertad de movimientos para cubrir otras zonas del espacio de entrada en las que la red se comporta de forma inadecuada. A las funciones base relevantes sólo se les permitirá realizar movimientos muy localizados, ya que presentan una mayor sensibilidad al movimiento que el resto.

Cada una de las componentes del vector desplazamiento $\vec{d}$ se escoge siguiendo la ecuación

$$d_j = \text{signo}() \cdot \text{u}(0, r_i + \delta_i) \tag{2.17}$$

donde la función signo() sigue la ecuación (2.15), r_i es el radio de la función base, y δ_i es una constante inversamente proporcional al valor singular asociado a la función base:

$$\delta_i = \frac{\sigma_{\text{máx}} - \sigma_i}{\sigma_{\text{máx}} - \sigma_{\text{mín}}} \cdot \Delta_i \tag{2.18}$$

En la ecuación anterior, $\sigma_{\text{máx}}$ y $\sigma_{\text{mín}}$ son los valores singulares máximo y mínimo respectivamente, σ_i es el valor singular asociado a la función base seleccionada para se mutada y Δ_i es el máximo movimiento permitido para la función base i-ésima.

2.3.2.2.5.2. Alteración aleatoria global del radio de una función base según su valor singular asociado

Al igual que en el caso de los centros, la alteración aleatoria del radio de una función base se realiza mediante la aplicación de un desplazamiento aleatorio a su radio:

$$(r_i)' = r_i + \text{signo}() \cdot \text{u}(0, r_i + \delta_i) \tag{2.19}$$

donde δ_i se calcula usando la ecuación (2.18).

2.3.2.2.6. MGOLS (Mutación Global de redes usando OLS)

Este operador evolutivo realiza exactamente la misma función que el anterior, pero usando el método OLS para asignar relevancia a las funciones base en vez de la descomposición SVD de la matriz de activación de la red (González et al., 2001b).

Una vez que una red ha sido seleccionada para sufrir una mutación, todas sus funciones base tienen la misma probabilidad de ser alteradas. La función base elegida ϕ_i sufrirá una alteración en su radio o en su posición de una magnitud inversamente proporcional a su radio de reducción del error$[rre]_i$.

2.3.2.2.6.1. Alteración aleatoria global del centro de una función base según su radio de reducción del error asociado

Este desplazamiento se realiza de forma análoga al que aplica el operador MGSVD. La única diferencia en el proceso está en el cálculo de la constante δ_i, que en este caso depende de los radios de reducción del error:

$$\delta_i = \frac{[rre]_{\text{máx}} - [rre]_i}{[rre]_{\text{máx}} - [rre]_{\text{mín}}} \cdot \Delta_i \tag{2.20}$$

donde $[rre]_{\text{máx}}$ y $[rre]_{\text{mín}}$ son los radios de reducción del error máximo y mínimo respectivamente, $[rre]_i$ es el radio de reducción del error asociado a la función base seleccionada para ser mutada.

2.3.2.2.6.2. Alteración aleatoria global del radio de una función base según su radio de reducción del error asociado

De forma análoga al operador de mutación anterior, se aplica un desplazamiento aleatorio al radio de la función base seleccionada siguiendo la siguiente ecuación:

$$(r_i)' = r_i + \text{signo}() \cdot \text{u}(0, r_i + \delta_i) \tag{2.21}$$

donde δ_i se calcula usando la ecuación (2.20).

2.3.3. El tamaño de la población

El tamaño de la población es un parámetro que influye determinantemente en los resultados obtenidos por el algoritmo evolutivo (Horn, 1997). Un tamaño pequeño produce que la ejecución del algoritmo sea ligera, ya que se tendrán que procesar pocos individuos en cada generación, pero suele

involucrar una rápida pérdida de la diversidad de la población. Por contra, un tamaño elevado hace la ejecución del algoritmo muy lenta, aunque ayuda a mantener una población diversa.

En la bibliografía se pueden encontrar diversos trabajos que estudian teóricamente el tamaño óptimo de la población para problemas abstractos (Alander, 1992); (Goldberg, 1985); (Goldberg, 1989b); (Harik et al., 1999). Sin embargo, estos trabajos sirven de poco para fijar el tamaño de la población de un algoritmo evolutivo adaptado para el diseño de RBFNN, ya que al estar basados en problemas simples y abstractos, hacen suposiciones sobre el problema que son difíciles o imposibles de probar en el campo de la aproximación de funciones mediante RBFNN. Otro serio inconveniente es que se basan en la representación de las soluciones como cadenas de bits, ofreciendo tamaños de población en función de la longitud de la cadena de bits usada para el problema. Aunque esto no es aplicable al problema tratado en esta memoria puesto que no se usan cadenas de bits para codificar las soluciones. Otras conclusiones extraídas de los trabajos anteriores son que el tamaño óptimo de la población está íntimamente relacionado con el tipo de operadores de cruce usados (De Jong and Spears, 1991); (Spears and De Jong,), con la calidad de la población inicial (Reeves, 1993), o con ciertas propiedades de la función de evaluación (Alander, 1992). Para determinar el tamaño de la población en (González et al., 2003), se realizó un estudio empírico siguiendo las sugerencias de (De Jong and Spears, 1991); (Pelikan and Lobo, 1999). Como se muestra en la Tabla 2.2, el error de aproximación de las soluciones no se ve muy afectado por el tamaño de la población a partir de un número determinado. Estos resultados sugieren la elección de un tamaño de población pequeño, de forma que el algoritmo sea lo más rápido posible, pero sin sacrificar a cambio la calidad de la solución final obtenida.

Tam.	Después del A. E.			Después del A. M.		
Pob.	Mín.	Med.	Máx.	Mín.	Med.	Máx.
10	0.0798	0.1291	0.1934	0.0289	0.0311	0.0357
20	0.0791	0.1129	0.1755	0.0261	0.0322	0.0395
30	0.0734	0.0950	0.1109	0.0259	0.0383	0.0549
40	0.0734	0.1016	0.1389	0.0226	0.0395	0.0531
50	0.0609	0.0822	0.1082	0.0251	0.0364	0.0498
100	0.0579	0.0677	0.0748	0.0237	0.0325	0.0474
500	0.0569	0.0619	0.0677	0.0198	0.0321	0.0470
1000	0.0569	0.0666	0.0892	0.0279	0.0459	0.0689

Tab. 2.2: Efecto del tamaño de la población en la solución final para la función f_1

2.4. Conclusiones

Este capítulo ha presentado el modelo que permite resolver el problema de aproximación funcional así como dos metodologías de diseño.

En cuanto a la primera metodología clásica de diseño de RBFNN, se ha mostrado como un algoritmo diseñado específicamente para inicializar los centros de las RBF es capaz de mejorar considerablemente al resto de los algoritmos de *clustering* tradicionales, ya que al tener en cuenta la variabilidad de las salidas esperadas de la función objetivo, es capaz de detectar aquellas zonas del espacio de entrada en las que, es necesario incrementar el número de centros.

Continuando con la metodología clásica, para fijar el radio de las funciones base, se han presentado las heurísticas más usadas en la literatura (KNN y CIV), y se ha mostrado cómo la primera de ellas produce demasiado solapamiento entre las funciones base conforme K aumenta, llegándose a perder por completo la localidad de las funciones base.

Respecto a la segunda metodología de diseño, se ha descrito un algoritmo evolutivo existente en la bibliografía que establece todos los parámetros que

definen una RBFNN salvo el número de neuronas que debe poseer la red.

Para ello, se adaptaron la representación de los individuos y los operadores de cruce y mutación, añadiendo conocimiento experto para guiar las modificaciones realizadas en la red hacia mejores zonas del espacio solución.

3. NUEVOS ALGORITMOS DE INICIALIZACIÓN DE PARÁMETROS EN LAS METODOLOGÍAS CLÁSICAS DE DISEÑO DE RBFNN

Dentro del capítulo previo se introdujo un algoritmo de *clustering* orientado al diseño de RBFNN que, gracias a su mecanismo de supervisión, es capaz de situar los centros de las RBF en lugares más adecuados.

Este capítulo parte de ese algoritmo, analizando los diversos elementos que pueden ser mejorados para, posteriormente, proponer soluciones que permitan la obtención de mejores resultados. Dentro de los elementos que pueden ser mejorados, el más importante es el tipo de partición del conjunto de entrenamiento, el cual, se analiza en profundidad a lo largo del capítulo puesto que se han propuesto numerosas técnicas al respecto.

Por último, se presenta un nuevo algoritmo que se desmarca de la per-

spectiva del *clustering* clásico y del método de supervisión presentado en el algoritmo CFA para proponer un nuevo enfoque del problema. Gracias a este enfoque, es posible tanto la inicialización de los centros como de los radios de las RBF, siendo esto último un gran avance puesto que las heurísticas empleadas sólo se basaban en la posición de los centros sin aprovechar los mecanismos de supervisión disponibles.

3.1. Clustering difuso para problemas de aproximación funcional (ICFA: Improved Clustering for Function Approximation)

El algoritmo presentado en el capítulo anterior tiene mejor comportamiento que los algoritmos usados tradicionalmente para inicializar los centros de las RBF. Sin embargo, posee una serie de elementos que pueden ser mejorados. El primer trabajo de innovación que se muestra en esta memoria es el desarrollo de un nuevo algoritmo, el *Improved Clusetring for Function Approximation*, (ICFA) (**Guillén** et al., 2005b); (**Guillén** et al., 2007a); (**Guillén** et al., 2005d), que mejora las deficiencias presentes en el algoritmo CFA. Entre los elementos a ser mejorados se encuentran:

- El algoritmo CFA, teniendo en cuenta que está basado en el *Hard C-means*, realiza una partición dura de los vectores de entrada, esto significa que un vector de entrada solo puede pertenecer a un *cluster* en un momento dado. Desde el punto de vista de una red neuronal esto no es muy adecuado ya que un vector de entrada puede activar varias neuronas al mismo tiempo.

- Otra cuestión importante es la corrección del método iterativo propuesto para alcanzar la posición final de los centros en base a un mínimo de la función de distorsión. Las ecuaciones que se presentan para actualizar las posiciones de los centros y de sus salidas esperadas no garantizan la convergencia a un mínimo de la función de distorsión, por lo tanto son cuestionables.

- Respecto al mismo método iterativo, otro elemento susceptible de ser mejorado es la eficiencia. El esquema general del algoritmo muestra un

bucle de actualización de los centros y sus salidas esperadas que itera hasta que la distorsión no disminuya significativamente, dentro de otro bucle que comprueba de nuevo esta condición tras actualizar el valor de la distorsión de los nuevos centros tras migrar. Como se mostrará adelante, solamente hace falta un único bucle que itere hasta que los centros no varíen sus posiciones de forma significativa, haciendo que el algoritmo sea considerablemente más rápido.

- Otro elemento que puede ser optimizado es el proceso de migración. Durante la migración se ha de iterar nuevamente sobre un subconjunto de centros además de tomar decisiones en base a un criterio según un valor aleatorio. Para obtener la máxima eficiencia y robustez, se ha reemplazado el criterio, estableciendo uno fijo y sólo se tiene en cuenta un centro por paso de migración.

- Como último aspecto a mejorar, se ha analizado el hecho de tener que asignar un valor a un parámetro ($\vartheta_{\text{mín}}$) (2.5) que afecta significativamente a los resultados obtenidos por el algoritmo además de a su convergencia y tiempo de ejecución. Como se muestra en la siguiente sección, este parámetro ha sido eliminado de modo que no hace falta una supervisión exhaustiva del resultado del algoritmo para obtener resultados aceptables, en un tiempo reducido.

Las siguientes subsecciones analizan los problemas mostrados y proponen un conjunto de soluciones que, como se mostrará se en sección 3.1.7.3, proporcionan mejores resultados en menor tiempo de ejecución.

3.1.1. Partición de los datos de entrada

El algoritmo ICFA propone un modo distinto de particionar los datos en comparación con el algoritmo CFA. El algoritmo CFA distribuye los vectores de entrada repartiéndolos entre los centros de manera exclusiva de modo que un vector de entrada puede pertenecer únicamente a un centro. Sin embargo, en (Bezdek, 1981) se propuso un algoritmo de *clustering* que colocaba los centros de los *clusters* considerando la posibilidad de que un vector de entrada perteneciese a varios *clusters* con un grado de pertenencia. Este algoritmo es conocido por el nombre de C-medias difuso (Fuzzy C-means, FCM) y la incorporación de la partición difusa le permite obtener unos resultados notablemente superiores al algoritmo clásico donde se realiza una partición dura.

Desde el punto de vista de una RBFNN, el hecho de que un vector de entrada active una única neurona no es del todo adecuado puesto que éste está conectado con todas las RBF de la red. Es más, si un vector de entrada no pudiera activar varias neuronas simultáneamente se perdería la capacidad de generalización e interpolación de la red. Por tanto, parece más adecuado emplear una partición difusa durante la inicialización de los centros de la red para poder ubicarlos teniendo en cuenta que un vector de entrada puede influir en la posición de varios centros. En (**Guillén** et al., 2004b); (**Guillén** et al., 2005c); (**Guillén** et al., 2004a) se realizó un estudio previo de lo beneficioso que puede ser el uso de *clustering* difuso en problemas de aproximación funcional.

3.1.2. Parámetro w

En el algoritmo CFA, la salida estimada de un centro es calculada utilizando como elemento ponderador el parámetro ω. El cálculo de este parámetro

conlleva la definición de otro parámetro ($\vartheta_{\text{mín}}$) que afecta considerablemente al rendimiento tanto a nivel de resultados como de tiempo ejecución. Si el valor de este parámetro no es elegido adecuadamente, los resultados del algoritmo pueden ver reducida su calidad y la convergencia a un mínimo de la función de distorsión puede ser alterada conllevando a un mayor tiempo de ejecución del algoritmo. En el algoritmo ICFA se define un parámetro ponderador w como:

$$w_{ik} = |F(\vec{x}_k) - o_i| \tag{3.1}$$

donde $F(\vec{x}_k)$ es la salida de la función a aproximar y o_i la salida estimada del centro $\vec{c}_i$. Este parámetro no requiere la inicialización de ningún valor haciendo el algoritmo más robusto en lo que se refiere a la calidad de los resultados finales y al tiempo de ejecución.

3.1.3. Función objetivo

Para poder adaptar el algoritmo CFA para utilizar una partición difusa, se ha de redefinir la función que mide la distorsión de la partición. La nueva función de distorsión es:

$$J_h(U, C, W) = \sum_{k=1}^{n} \sum_{j=1}^{m} u_{ik}^h D_{ikW} \tag{3.2}$$

sujeta a las siguientes restricciones:

- $\sum_{i=1}^{m} u_{ik} = 1 \ \forall k = 1...n$
- $0 < \sum_{k=1}^{n} u_{ik} < n \ \forall i = 1...m.$

donde $U = [u_{ik}]$ es la matriz con los valores que indica el grado de pertenencia del vector de entrada $\vec{x}_k$ al centro $\vec{c}_i$, D_{ikW} es la distancia Euclídea

ponderada entre $\vec{c}_i$ y $\vec{x}_k$, $C = [\vec{c}_i]$ es la matriz que almacena las coordenadas de todos los centros de la red, $W = [w_{ik}]$ es la matriz que contienen los valores ponderadores calculados a partir de o_i y h un parámetro que controla cómo de difusa será la partición. En el algoritmo FCM el valor recomendado para el parámetro h es 2, de modo que, hasta que no se especifique lo contrario, también se usará este valor para el algoritmo ICFA.

Dado que lo que se desea es concentrar más centros en las zonas donde hay más variabilidad en la salida, es necesario modificar el criterio de similitud entre un centro y un vector de entrada. Esta es la razón para ponderar la distancia Euclídea. En condiciones normales, cuanto más cerca esté un centro de un vector de entrada, mayor será el grado de pertenencia del vector de entrada a ese centro concreto. Sin embargo, en el caso de la inicialización de los centros, esto no tiene por qué ser cierto ya que puede haber situaciones en las cuales haya vectores de entrada con poca variabilidad distribuidos en una gran parte del espacio de entrada, los cuales necesiten una única RBF para ser modelados. En estas situaciones, si se utiliza como criterio de similitud la distancia Euclídea, se situarán varios centros en esa área para cubrir la densidad de vectores de entrada. También existe la posibilidad de que en una zona no muy extensa del espacio de entrada, la variabilidad de la función sea alta y se requieran varias RBF para un correcto modelado pero si se utiliza la distancia Euclídea no se situarán más centros ahí debido a que el área, al ser pequeña, estará cubierta con una RBF.

El efecto de ponderar la distancia Euclídea utilizando el parámetro$w$ tiene como objetivo compensar la lejanía o cercanía de un centro con respecto a sus vectores de entrada considerando los valores de salida. El cálculo de la distancia ponderada se obtiene mediante:

$$D_{ikW} = \|\vec{x}_k - \vec{c}_i\|^2 \cdot w_{ik}^2 \tag{3.3}$$

donde $w_{ik} = |Y_k - o_i|$ es el parámetro que considera la salida de cada vector de entrada y la posición hipotética de un centro en el eje de salida. El efecto de w sobre la distancia es el siguiente: cuando w tome valores pequeños, la distancia entre un centro y un vector de entrada se verá reducida, es decir, si un vector y un centro tienen una salida similar, la distancia entre ellos será reducida implicando esto un incremento en el valor de la función de pertenencia de ese centro con respecto a ese vector.

3.1.4. Proceso iterativo de actualización

Una vez redefinida la función de distorsión a minimizar, se puede establecer un método para alcanzar la solución. Tal y como se propuso en (Bezdek, 1981), se utiliza un proceso de optimización alterna que consiste en aplicar el método iterativo de Picard sobre las funciones de pertenencia, las posiciones de los centros y sus salidas estimadas. Las ecuaciones que permiten calcular los valores de los parámetros se obtienen derivando la función de distorsión (ampliada mediante multiplicadores de Lagrange para incorporar las restricciones) con respecto a las variables c,u y o, obteniendo:

$$u_{ik} = \left(\sum_{j=1}^{m} \left(\frac{D_{ikW}}{D_{jkW}} \right)^{\frac{1}{h-1}} \right)^{-1} \tag{3.4}$$

$$\vec{c}_i = \frac{\sum_{k=1}^{n} u_{ik}^h \vec{x}_k w_{ik}^2}{\sum_{k=1}^{n} u_{ik}^h w_{ik}^2} \tag{3.5}$$

$$o_i = \frac{\sum_{k=1}^{n} u_{ik}^h y_k d_{ik}^2}{\sum_{k=1}^{n} u_{ik}^h d_{ik}^2} \tag{3.6}$$

donde d_{ik} es la distancia Euclídea entre $\vec{c}_i$ y $\vec{x}_k$.

Estas ecuaciones sustituyen a las propuestas por el algoritmo CFA (Eq. 2.7), asegurando una convergencia hacia un mínimo de la función de distorsión. Como se mostrará en el esquema general del algoritmo, no hace falta iterar sobre un bucle dentro del bucle principal del algoritmo tal y como se propone en el algoritmo CFA sino que sólo de han de computar estos valores en cada iteración del bucle principal del algoritmo ICFA. Esto supone un gran ahorro en el tiempo de ejecución.

3.1.5. Paso de migración

Tal y como se propuso en el algoritmo CFA incluyendo la idea presentada en (Patanè and Russo, 2001), para poder evitar el estancamiento en mínimos locales de la función de distorsión, se introduce un paso de migración que permite actualizar las posiciones de los centros sin usar el método iterativo. Sin embargo, el proceso de migración del algoritmo CFA ha sido modificado con la intención de obtener la máxima eficiencia y robustez posibles.

En el paso de migración del algoritmo ICFA solo se realiza una única iteración, en vez del bucle que existe en la del CFA, haciendo en primer lugar una preselección de los centros. La preselección de los centros se realiza con el objetivo de descartar para ser migrados los centros que están cumpliendo su función aunque desde el punto de vista de la función de distorsión pueda no parecerlo. Dado que se modifica el criterio de similitud entre los centros y los vectores de entrada, puede darse el caso de que un centro que posee un

gran número de vectores de entrada con una salida similar posea un valor de distorsión pequeño. Este centro podría ser elegido para ser migrado a una zona donde haya mayor distorsión, dejando de cubrir su zona de manera correcta. Por tanto solo se consideran los centros con un valor de distorsión por encima de la media, con la idea de que si migramos un centro que aporta una gran distorsión cerca de otro que también aporta una distorsión grande, entre ambos, sean capaces de disminuir la distorsión total de la partición.

Una vez hecha la preselección, disponemos de los centros que pueden ser destino o fuente de la migración. Para elegir cuál será el centro que migra y el centro de destino se establece un criterio fijo en el cuál se establece que el centro de destino es el que mayor distorsión aporta, y el centro de origen el que menos. Una vez que el centro se ha colocado en su posición de destino siguiendo el mismo procedimiento que en el CFA, se actualizan los valores de las funciones de pertenencia y se calcula la distorsión de la nueva partición. Si la distorsión total es menor, la migración se acepta y si no ha disminuido se rechaza.

En la Figura 3.1 se puede apreciar cómo evoluciona el valor de la distorsión total de la partición a lo largo de las iteraciones del algoritmo. Hay dos líneas, una que corresponde a una ejecución sin emplear la migración de centros y otra utilizándola. Como muestra la figura, gracias al empleo del paso de migración es posible alcanzar mínimos locales menores tras migrar en las iteraciones 2,5 y 6.

3.1.6. Esquema general de ICFA

Una vez expuestos todos los elementos nuevos del algoritmo ICFA con respecto al CFA, el esquema general que sigue el algoritmo propuesto se muestra en la Figura 3.2.

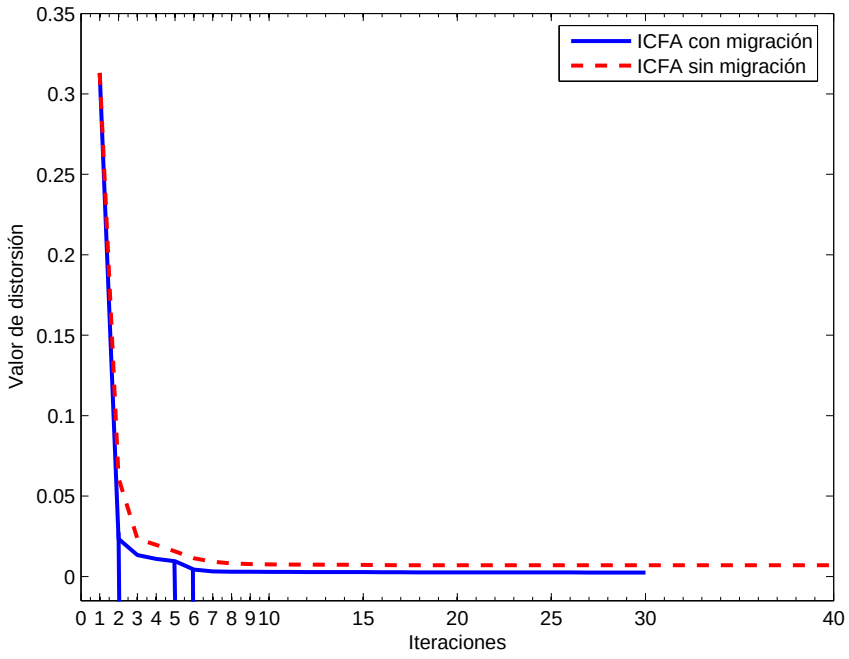


Fig. 3.1: Valores de la distorsión total de la partición a lo largo de las iteraciones del algoritmo, considerando o no la migración.

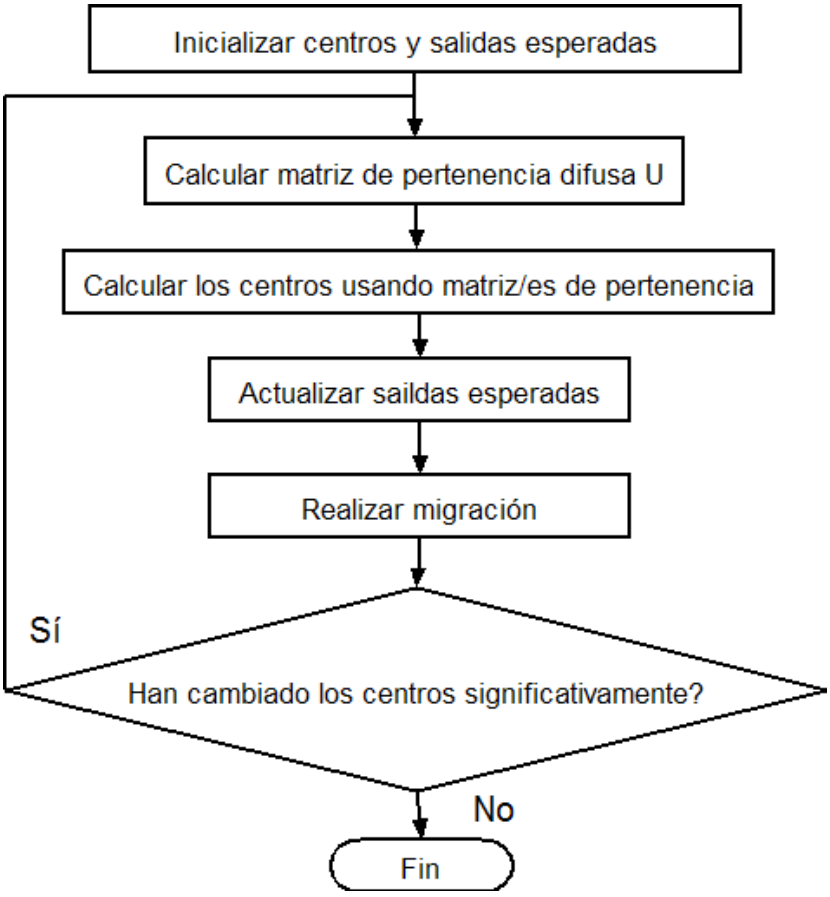


Fig. 3.2: Esquema general del algoritmo ICFA.

En el algoritmo ICFA, el punto de partida no consiste en una inicialización aleatoria de los centros o de la matriz con los valores de pertenencia, sino que se fija un valor concreto para las posiciones de los centros. Éstos son distribuidos uniformemente por todo el espacio de entrada y todas sus salidas estimadas son inicializadas a 1. Dado que se realiza un descenso en gradiente, se alcanza un mínimo local y, aunque la migración ayude a mejorar la convergencia a un mínimo local mejor, no lo garantiza. La consecuencia es una falta de robustez del algoritmo, tal y como ocurre con otros algoritmos. Por tanto se ha decidido dar una inicialización fija de modo que para la misma entrada, el algoritmo devuelve la misma salida siempre. La posición inicial de los centros permite que un vector de entrada, no importa dónde se encuentre, tendrá un centro cercano desde el punto de partida del algoritmo. La razón de fijar las salidas estimadas al mismo valor tiene como motivo ser equitativo a la hora de calcular por primera vez las distancias entre los centros y los vectores de entrada.

3.1.7. Resultados experimentales

Tras describir las diversas mejoras propuestas, esta subsección muestra cómo, efectivamente, los cambios incorporados al nuevo algoritmo ICFA permiten obtener mejores resultados requiriendo un menor tiempo de ejecución.

El criterio establecido para determinar si un resultado es bueno o no es el error de aproximación utilizando su expresión cuadrática media normalizada (Normalized Root Mean Squared Error, NRMSE):

$$\mathbf{NRMSE} = \sqrt{\frac{\sum_{k=1}^{n} \left(F(\vec{x}_k) - \mathcal{F}(\vec{x}_k; C, R, \Omega)\right)^2}{\sum_{k=1}^{n} \left(F(\vec{x}_k) - \bar{F}\right)^2}} \tag{3.7}$$

donde $\bar{F}$ es la media de la salida de la función a ser aproximada.

Una vez los centros han sido inicializados usando los algoritmos de *clustering*, los radios se inicializan usando la heurística de los K vecinos con K=1. Por último el algoritmo de búsqueda local (Levenberg-Marquardt (Marquardt, 1963)) se encarga de hacer un ajuste fino tanto de los centros como de los radios. La tarea principal de los algoritmos de inicialización de los centros es encontrar una posición adecuada para que, al aplicar la búsqueda local, se pueda encontrar un buen mínimo local.

3.1.7.1. Experimento 1

El primer experimento consiste en aproximar datos reales provenientes de la reflectancia para distintas longitudes de onda emitidas por un fragmento de carne de cerdo ibérico. La muestra fue recogida utilizando un espectrómetro portátil Vis/NIR FieldSpec® Pro JR (*Analytical Spectral Devices*, Inc; Boulder, Colorado, USA) de donde se extrajeron las reflectancias para el espectro comprendido en el rango de 350-2500 nm. El espectro de un cerdo es obtenido mediante el cálculo de la media de los valores de reflectancia obtenidos tras realizar 10 escaneos del espectro. En la Figura 3.3 se muestra el espectro para un cerdo ibérico, una vez que sus valores de reflectancia han sido normalizados, donde el conjunto de entrenamiento está formado por las cruces y el de test por puntos. De una medida de reflectancia con 2500 valores de longitudes de onda, se escogieron aleatoriamente 630 para entrenamiento y el resto fue utilizado para test.

Dado que los valores de reflectancia de la carne están relacionados con su ternura, si existe la posibilidad de diseñar modelos que aproximen la reflectancia de un cerdo, éstos podrían analizarse e intentar obtener información a partir de los parámetros de éste sin tener que analizar el espectro

por completo. En la Tabla 3.1 se muestran los errores de aproximación para el conjunto de entrenamiento antes y después de realizar la búsqueda local y en la Tabla 3.2 se muestran los errores de aproximación para el conjunto de test. En la comparativa se han incluido los algoritmos CFA, FCM y Gustafson-Kessel (GK) (Gustafson and Kessel, 1979), estos dos últimos han sido empleados extensamente en la bibliografía para inicializar los centros de las RBF.

Los resultados muestran la gran mejora que se ha obtenido con respecto al algoritmo CFA tanto en la primera inicialización de los centros como tras la aplicación de la búsqueda local. Esto indica que el algoritmo ICFA es capaz de identificar las zonas del espacio de entrada donde se requieren más RBF para poder ser modeladas desde un primer momento y, gracias a esa identificación, al hacer un ajuste de los parámetros, los resultados son también superiores. Tal y como cabía esperar, el algoritmo ICFA también supera notablemente los resultados proporcionados por los otros dos algoritmos dado que no fueron diseñados para la inicialización de centros sino para agrupación en *clusters*.

3.1.7.2. Experimento 2

Los datos utilizados en este segundo experimento proceden de la base de datos conocida como *UCI Machine Learning Repository*(S. Hettich and Merz, 1998). Más concretamente se ha utilizado el conjunto de datos denominado MPG (Miles Per Galon) donde el objetivo es, conocidas las características de unos automóviles, se desea predecir el consumo de combustible en galones. Las variables que identifican cada coche son:

- Desplazamiento
- Caballos de Vapor

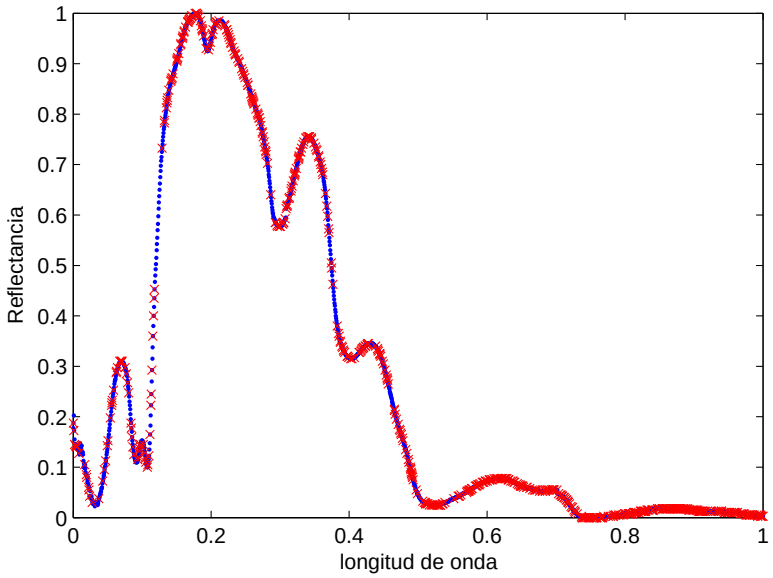


Fig. 3.3: Espectroscopía de una pieza de cerdo ibérico (datos normalizados). Las cruces rojas representan los datos de entrenamiento, los puntos azules, los de test.

RBF	**FCM**	**GK**	**CFA**	**ICFA**
5	0.270(0.015)	0.310(0.000)	0.316(0.029)	0.219(0)
6	0.255(0.013)	0.316(0.092)	0.316(0.004)	0.207(0)
7	0.276(0.040)	0.265(0.044)	0.403(0.139)	0.237(0)
8	0.250(0.034)	0.273(0.049)	0.258(0.007)	0.480(0)
9	0.259(0.062)	0.256(0.029)	0.231(0.043)	0.206(0)
10	0.206(0.051)	0.232(0.022)	0.235(0.011)	0.168(0)
11	0.254(0.059)	0.192(0.041)	0.216(0.033)	0.159(0)

RBF	**FCM**	**GK**	**CFA**	**ICFA**
5	0.145(0.023)	0.182(0.001)	0.173(0.003)	0.167(0)
6	0.160(0.018)	0.141(0.022)	0.172(0.005)	0.085(0)
7	0.133(0.035)	0.096(0.045)	0.181(0.012)	0.090(0)
8	0.108(0.053)	0.070(0.017)	0.163(0.010)	0.046(0)
9	0.072(0.050)	0.065(0.022)	0.102(0.046)	0.045(0)
10	0.044(0.004)	0.088(0.057)	0.110(0.046)	0.041(0)
11	0.059(0.020)	0.061(0.025)	0.063(0.032)	0.030(0)

Tab. 3.1: Media y desviación estándar (en paréntesis) del error de aproximación antes y después de aplicar la búsqueda local para el conjunto de entrenamiento de la reflectancia.

RBF	**FCM**	**GK**	**CFA**	**ICFA**
5	0.154(0.020)	0.190(0.001)	0.184(0.007)	0.174(0)
6	0.163(0.013)	0.151(0.024)	0.180(0.006)	0.092(0)
7	0.139(0.034)	0.102(0.045)	0.187(0.011)	0.094(0)
8	0.115(0.055)	0.078(0.016)	0.167(0.010)	0.050(0)
9	0.078(0.049)	0.072(0.024)	0.111(0.048)	0.050(0)
10	0.050(0.007)	0.097(0.058)	0.120(0.045)	0.046(0)
11	0.075(0.020)	0.071(0.022)	0.075(0.036)	0.034(0)

Tab. 3.2: Media y desviación estándar del error de aproximación para el conjunto de test de la reflectancia.

- Peso
- Aceleración
- Año del modelo

El algoritmo ICFA es comparado en esta ocasión con otros algoritmos presentados en la bibliografía tales como el algoritmo *Modified Gath-Geva fuzzy Clustering* (MGGC) (János Abonyi and Robert Babuska and Ferenc Szeifert, 2002), el algoritmo ANFIS (Jang and Sun, 1995) incluido en la Toolbox difusa de MATLAB y con la *Toolbox* de MATLAB de identificación de modelos difusos (*Fuzzy Model Identification*, FMID) basada en el algoritmo de Gustafson-Kessel. De los 392 datos disponibles, 196 fueron usados como training y el resto como test, siguiendo el mismo criterio que en (János Abonyi and Robert Babuska and Ferenc Szeifert, 2002). En (János Abonyi and Robert Babuska and Ferenc Szeifert, 2002) se diseñan modelos difusos del tipo Takagi-Sugeno-Kang (TSK) con cuatro reglas cuyas funciones de pertenencia son gausianas y los consecuentes lineales, por tanto, dada la equivalencia entre estos sistemas difusos y las RBFNN, las redes generadas con los algoritmos CFA e ICFA poseen 4 neuronas.

La Tabla 3.3 muestra los errores de aproximación para los conjuntos de entrenamiento y test. Aunque el menor error de aproximación para el conjunto de entrenamiento es obtenido por el algoritmo ANFIS, éste es el que peores resultados de test obtiene. El algoritmo ICFA obtiene los segundos mejores resultados en el entrenamiento y además obtiene los errores más bajos a la hora de enfrentar el modelo a los datos de test. Esto indica que, aún obteniendo gran precisión con los datos de entrenamiento, las redes generadas son capaces de interpolar y obtener buenas aproximaciones con datos nuevos.

	Entrenamiento	**Test**
FMID	0.3396	0.3818
ANFIS	0.2506	11.8222
MGGC	0.3459	0.3688
CFA	0.3459	0.3637
ICFA	0.3306	0.3494

Tab. 3.3: Errores de aproximación para el problema MPG.

3.1.7.3. Tiempos de ejecución

Tal y como se ha mostrado, muchos aspectos del algoritmo original han cambiado y el resultado de estos cambios no se refleja únicamente en la disminución de los errores de aproximación obtenidos por las RBFNN generadas, sino por un menor tiempo de ejecución del algoritmo. Las modificaciones en el proceso de actualización de los centros y sus salidas estimadas, eliminando el bucle donde se obtenían los nuevos valores, así como en la migración, donde ahora solo se consideran dos centros en cada paso de migración, permiten obtener unos tiempos de ejecución mucho menores que los que se obtienen con el algoritmo CFA. Para ilustrar esto, en la Figura 3.4 se muestran los pares error de aproximación/ tiempo de ejecución obtenidos en los experimentos realizados para la función de la reflectancia de las longitudes de onda.

3.2. Aplicación de clustering posibilístico para aproximación funcional

En la sección previa se ha descrito el diseño de un algoritmo de *clustering* que, utilizando una partición difusa, realiza un inicialización de los centros de las RBF. En concreto, el algoritmo ICFA tiene un comportamiento bas-

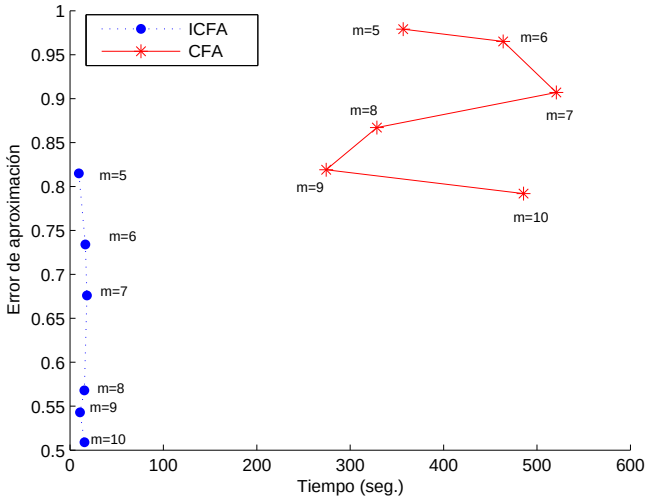


Fig. 3.4: Tiempo de ejecución y error de aproximación para la función de la espectroscopía de varias ejecuciones de los algoritmos CFA e ICFA usando diferentes valores para el número de centros (m).

tante adecuado, no requiriendo la inicialización muchos parámetros para la obtención de buenos resultados. Tal y como se ha mostrado en la sección 3.1, gracias al empleo de una partición difusa, el ICFA es menos sensible cuando hay datos con ruido en comparación con el uso de una partición dura. Sin embargo aunque se puedan asignar distintos grados de pertenencia reduciendo el efecto del ruido, debido a las características de la partición difusa, los datos con ruido pertenecerán siempre a un *cluster*, afectando al comportamiento del algoritmo. Para resolver este problema se han desarrollado varios algoritmos que utilizan particiones posibilísticas (Krishnapuram and Keller, 1993); (Timm and Kruse, 2002) y enfoques mixtos que mezclan particiones difusas con posibilísticas (Zhang and Leung, 2004); (Pal et al., 1997). Dado que el reemplazo de la partición dura por una difusa permitió al ICFA mejorar los resultados, es razonable hacer un estudio del comportamiento del algoritmo cuando se utiliza una partición posibilística pura y difuso-posibilística.

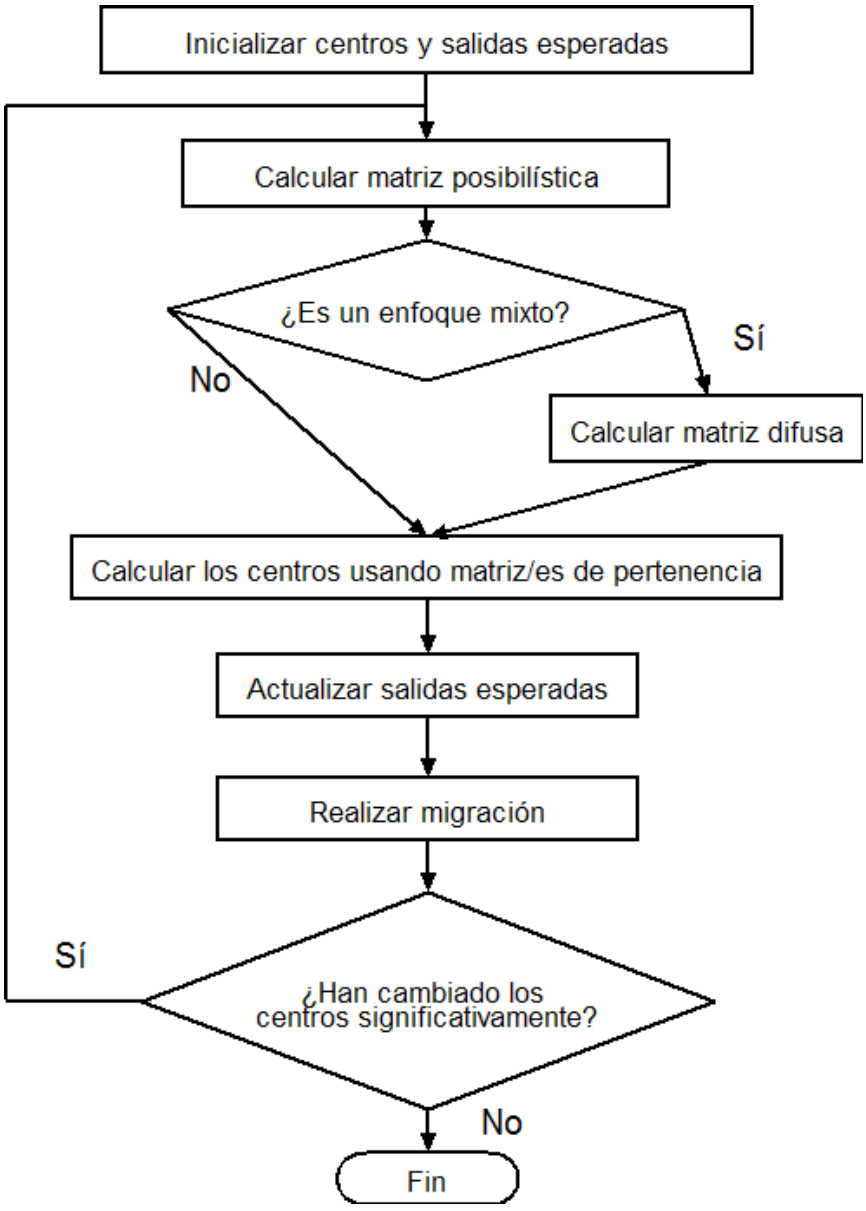


Fig. 3.5: Esquema general para los algoritmos posibilísticos.

Todos los algoritmos que vamos a introducir presentan una estructura muy similar a la que el ICFA posee salvo por ciertos elementos que son añadidos a consecuencia de utilizar particiones posibilísticas (**Guillén** et al., 2007c); (**Guillén** et al., 2005a); (**Guillén** et al., 2006d). Estos cambios se refieren a las matrices que deben ser calculadas así como la inicialización de los distintos parámetros que cada algoritmo posee. En general, la estructura general que los algoritmos siguen se muestra en la Figura 3.5.

En los algoritmos posibilísticos, los centros son pre-inicializados utilizando el algoritmo FCM aunque, para el problema de aproximación funcional, esto puede conducir a distintas configuraciones, especialmente tras la aplicación de la búsqueda local para optimizar el resto de los parámetros de la RBFNN. Para evitar este tipo de inconvenientes y para hacer una justa comparación entre los distintos algoritmos, al igual que en el algoritmo ICFA, los centros partirán de una zona prefijada. Los centros serán distribuidos de forma uniforme a través del espacio de los vectores de entrada y sus salidas estimadas serán todas igual a 1. De este modo, todos los centros serán influidos por la salida del mismo modo en la primera iteración. En dos de los algoritmos que utilizan un enfoque difuso-posibilístico es necesario ejecutar el algoritmo FCM para poder calcular el valor de ciertos parámetros.

Aunque el esquema que sigue la migración es el mismo que en el ICFA, algunas modificaciones han de ser introducidas para adaptar la migración a los enfoques posibilísticos y difuso-posibilísticos. Estas modificaciones residen en la asignación de los valores de distorsión para cada centro. Debido a que las funciones de distorsión son modificadas, el cálculo de la distorsión para cada prototipo debe hacerse según su correspondiente función de distorsión. El esquema general se muestra en la Figura 3.6.

Una vez introducidos de forma general los nuevos algoritmos, en las si-

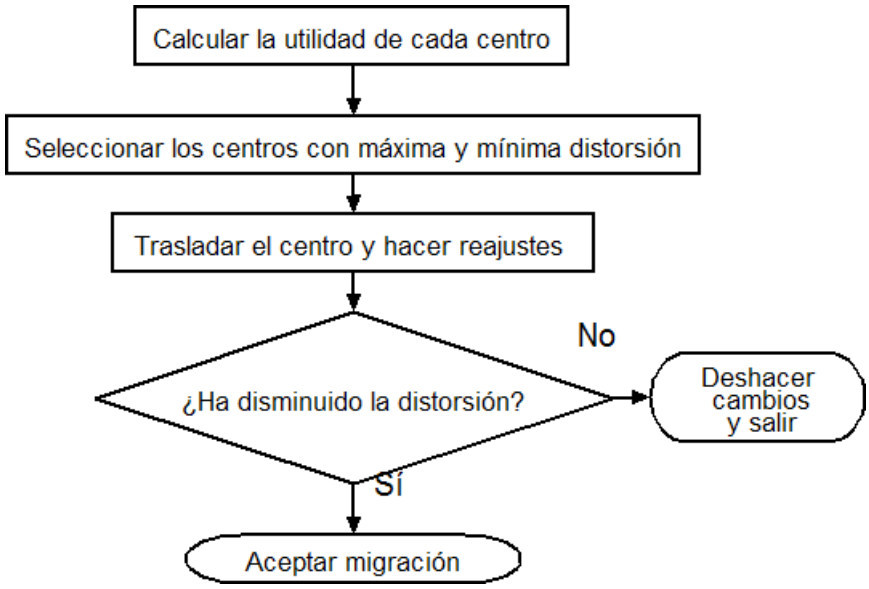


Fig. 3.6: Algoritmo de migración.

guientes subsecciones se describen detalladamente las particularidades de cada uno para, posteriormente, hacer un estudio experimental de su comportamiento y una comparativa entre los algoritmos con diversos ejemplos.

3.2.1. Possibilistic CFA (PCFA)

El algoritmo PCFA es el resultado de reemplazar la partición difusa que se utiliza en el ICFA por la que se utiliza en el Possibilistic C-means (PCM) (Krishnapuram and Keller, 1993). El algoritmo PCM se diseñó con el objetivo de solventar la presencia de *outliers* que pueden afectar a la partición difusa modificando la forma de los *clusters*.

La solución que proponen los autores en (Krishnapuram and Keller, 1993) es la de relajar una de las restricciones que se imponen en el algoritmo FCM para forzar que la partición sea difusa:

- $\sum u_{ik} = 1 \ \forall k = 1...n$

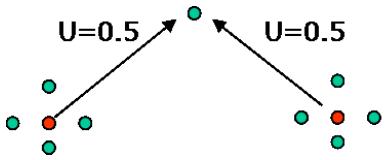


Fig. 3.7: Caso en el que la partición difusa no se comporta adecuadamente en presencia de un *outlier*.

- $0 < \sum u_{ik} < n \ \forall i = 1...m.$

La restricción $\sum_{i=1}^{m} u_{ik} = 1$ puede generar problemas cuando aparecen *outliers* o cuando existen vectores de entrada con ruido dentro del conjunto de datos. Por ejemplo, en la Figura 3.7 se muestra un conjunto de entrada con sus correspondientes *clusters* asignados y dentro de los datos de entrada se ha introducido un *outlier*. Si aplicamos el concepto de partición difusa estrictamente, al estar equidistante el *outlier* de los dos centros del *cluster*, el valor de la matriz de pertenencia debe ser 0.5 para cada *cluster*. En este ejemplo esto sería un error ya que es evidente que el *outlier* no pertenece a ninguno de los dos *clusters*. Este comportamiento es una consecuencia de la restricción anterior que obliga a los centros a compartir todos los puntos de entrada de tal forma que cada punto de entrada debe pertenecer a un *cluster*. Al relajar la restricción responsable de este comportamiento el algoritmo podrá asignar valores de pertenencia independientemente de los valores que tome la función de pertenencia en otros *clusters*. En el ejemplo anterior se pueden obtener unos valores de pertenencia por debajo de 0.5 para cada *cluster*. El problema que se presenta cuando se relaja la citada restricción es que se alcanza el mínimo de la función de distorsión haciendo todos los valores de pertenencia cero. Para evitar este problema es necesario introducir un nuevo elemento a la función de distorsión obteniendo la siguiente formula:

$$J_h(U,C) = \sum_{k=1}^{n}\sum_{i=1}^{m} u_{ik}^h D_{ik}^2 + \sum_{i=1}^{m}\eta_i \sum_{k=1}^{n}(1-u_{ik})^h \tag{3.8}$$

donde η_i es cualquier número positivo. En esta función de distorsión, el primer término requiere que la distancia entre los datos de entrada y los centros de los *clusters* sea la menor posible. El segundo término obliga a u_{ik} tomar los valores más altos que sean posibles. Gracias a este último término, el problema de la obtención del mínimo de la función de distorsión cuando todos los u_{ik} son cero queda solventado. El valor η_i determina a qué distancia el valor de pertenencia de un vector de entrada a un *cluster* es igual a 0.5. Esto implica que este valor debe ser escogido en función de cómo de grandes se desea que sean los *clusters*. En (Krishnapuram and Keller, 1993), los autores proponen la siguiente fórmula para calcularlo:

$$\eta_i = Kp\frac{\sum_{k=1}^{n} u_{ik}^h D_{ik}^2}{\sum_{k=1}^{n} u_{ik}^h} \tag{3.9}$$

donde Kp es un número positivo que usualmente es 1. El enfoque posibilístico se demostró bastante robusto a la hora de identificar *clusters* en condiciones de ruido, aunque presenta un problema descrito en (Barni et al., 1996) el cual consiste en que la función de distorsión puede presentar un mínimo cuando todos los centros de los *clusters* toman el mismo valor. En (Timm and Kruse, 2002) se implementa una modificación del algoritmo PCM que solventa este problema. El enfoque que se utiliza es el de la repulsión de los *clusters*, esta es obtenida añadiendo un nuevo término a la función de distorsión:

$$J_h(U,C;X) = \sum_{k=1}^{n}\sum_{i=1}^{m} u_{ik}^h D_{ik}^2 + \sum_{i=1}^{m}\eta_i \sum_{k=1}^{n}(1-u_{ik})^h + \sum_{i=1}^{m}\gamma_i \sum_{j=1,j\neq i}^{m}\frac{1}{D_{ij}^2} \tag{3.10}$$

donde γ_i es un factor de ponderación definido como $\gamma_i = \gamma \sum_{k=1}^{n} u_{ik}$ y que regula el grado de repulsión de los *clusters*.

Este algoritmo presenta un enfoque interesante aunque esta repulsión de *clusters* no es muy adecuada para el problema de aproximación funcional ya que puede haber centros que estén muy cercanos haciendo muy difícil la elección de un valor adecuado para γ_i. Además la elección de los parámetros Kp y η_i también afecta al comportamiento del algoritmo. Debido a estas razones este enfoque posibilístico no es considerado para el algoritmo ICFA.

Tras aplicar los mismos cambios que se introdujeron a la función de distorsión de algoritmo FCM para obtener la partición posibilística a la función de distorsión definida por el ICFA, se obtiene:

$$J_h(U, C, W) = \sum_{k=1}^{n} \sum_{i=1}^{m} u_{ik}^h D_{ikW}^2 + \sum_{i=1}^{m} \eta_i \sum_{k=1}^{n} (1 - u_{ik})^h \tag{3.11}$$

Tal y como se hizo para el ICFA, se aplicada el mismo método para obtener las ecuaciones que permitirán minimizar la función de distorsión, obteniendo:

$$u_{ik} = \left(\left(1 + \frac{D_{ikW}^2}{\eta_i} \right)^{\frac{1}{h-1}} \right)^{-1} \tag{3.12}$$

$$\vec{c_i} = \frac{\sum_{k=1}^{n} u_{ik}^h \vec{x}_k w_{ik}^2}{\sum_{k=1}^{n} u_{ik}^h w_{ik}^2} \tag{3.13}$$

$$o_i = \frac{\sum_{k=1}^{n} u_{ik}^h y_k d_{ik}^2}{\sum_{k=1}^{n} u_{ik}^h d_{ik}^2} \tag{3.14}$$

3.2.2. Fuzzy-Possibilistic CFA (FPCFA)

En el algoritmo desarrollado en (Pal et al., 1997) se propone una combinación entre una partición difusa y una posibilística. La razón de emplear ambos enfoques es que la partición difusa es importante a la hora de asignar una etiqueta para poder hacer la clasificación, la partición posibilística permite poder mover los centros a través del espacio de los vectores de entrada sin los efectos negativos que se producen cuando hay ruido. Los autores proponen la siguiente función de distorsión para ser minimizada:

$$J_h(U,C,T) = \sum_{k=1}^{n}\sum_{i=1}^{m}(u_{ik}^h + t_{ik}^{h2})D_{ik}^2 \tag{3.15}$$

sujeta a las siguientes restricciones:

$$\sum_{i=1}^{m} u_{ik} = 1 \ \forall k = 1...n \tag{3.16}$$

$$\sum_{k=1}^{n} t_{ik} = 1 \ \forall i = 1...m \tag{3.17}$$

Sea $T = [t_{ik}]$, la restricción que se muestra arriba exige que cada fila de$T$ sume 1 pero se deja libertad en cuanto a que en una columna no haya ninguna entrada con valor 0. Esto significa que puede haber vectores de entrada que no pertenezcan a ningún *cluster*.

La adaptación de este tipo de partición que combina el enfoque posibilístico y el difuso en el algoritmo ICFA es muy directa; es suficiente con reemplazar el valor de la función de pertenencia por la suma de la función de pertenencia y el valor de la partición posibilística:

$$J_h(U,C,T,W) = \sum_{k=1}^{n}\sum_{i=1}^{m}(u_{ik}^h + t_{ik}^{h_p})D_{ikW}^2 \tag{3.18}$$

sujeta a las mismas restricciones que la función definida para el algoritmo FPCM. Como en el caso anterior, las ecuaciones de actualización de los centros, salidas estimadas y los valores de las particiones han de ser recalculados:

$$u_{ik} = \left(\sum_{j=1}^{m} \left(\frac{D_{ikW}}{D_{jkW}} \right)^{\frac{2}{h-1}} \right)^{-1} \tag{3.19}$$

$$t_{ik} = \left(\sum_{j=1}^{n} \left(\frac{D_{ikW}}{D_{ijW}} \right)^{\frac{2}{h_p-1}} \right)^{-1} \tag{3.20}$$

$$\vec{c}_i = \frac{\sum_{k=1}^{n} (u_{ik}^h + t_{ik}^{h_p}) \vec{x}_k w_{ik}^2}{\sum_{k=1}^{n} (u_{ik}^h + t_{ik}^{h_p}) w_{ik}^2} \tag{3.21}$$

$$o_i = \frac{\sum_{k=1}^{n} (u_{ik}^h + t_{ik}^{h_p}) y_k d_{ik}^2}{\sum_{k=1}^{n} (u_{ik}^h + t_{ik}^{h_p}) d_{ik}^2} \tag{3.22}$$

3.2.3. Improved Possibilistic CFA (IPCFA)

Otra modificación del algoritmo PCM fue propuesta en (Zhang and Leung, 2004) donde, al igual que el algoritmo previo, se emplea una combinación de una partición posibilística y una difusa. Para llevar a cabo su propósito, los autores reemplazan el criterio de similitud en la función de distorsión del FCM (la distancia Euclídea) por la función de distorsión definida en el PCM. El resultado de dicha operación es el siguiente:

$$J_h(U^{(p)}, U^{(f)}, C) = \sum_{k=1}^{n}\sum_{i=1}^{m}(u_{ik}^{(f)})^{h_f}(u_{ik}^{(p)})^{h_p}D_{ik}^2 + \sum_{i=1}^{m}\eta_i\sum_{k=1}^{n}(u_{ik}^{(f)})^{h_f}(1-u_{ik}^{(p)})^{h_p} \tag{3.23}$$

donde $u_{ik}^{(p)}$ es el valor de pertenencia posibilístico de $\vec{x}_k$ para el centro i, $u_{ik}^{(f)}$ es el valor de pertenencia difuso de $\vec{x}_k$ para el centro i, h_p y h_f son los exponentes correspondientes a las funciones de distorsión difusa y posibilística y η_i es un parámetro definido como:

$$\eta_i = \frac{\sum_{k=1}^{n}(u_{ik}^{(f)})^{h_f}(u_{ik}^{(p)})^{h_p}D_{ik}^2}{(u_{ik}^{(f)})^{h_f}(u_{ik}^{(p)})^{h_p}} \tag{3.24}$$

con las restricciones siguientes:

$$0 \leq u_{ik}^{(p)} \leq 1 \ \forall i = 1...m, k = 1...n \tag{3.25}$$

$$0 < \sum_{k=1}^{n} u_{ik}^{(p)} \leq n \ \forall i = 1...m \tag{3.26}$$

$$\sum_{i=1}^{m} u_{ik}^{(p)} > 0 \ \forall k = 1...n \tag{3.27}$$

$$0 \leq u_{ik}^{(f)} \leq 1 \ \forall i = 1...m, k = 1...n \tag{3.28}$$

$$0 < \sum_{k=1}^{n} u_{ik}^{(f)} < n \ \forall i = 1...m \tag{3.29}$$

$$\sum_{i=1}^{m} u_{ik}^{(f)} = 1 \ \forall k = 1...n \tag{3.30}$$

Al igual que con los anteriores algoritmos, el ICFA ha sido adaptado a este tipo de partición obteniendo la siguiente función de distorsión:

$$J_h(U^{(p)}, U^{(f)}, C, W) = \sum_{k=1}^{n}\sum_{i=1}^{m}(u_{ik}^{(f)})^{h_f}(u_{ik}^{(p)})^{h_p}D_{ikW}^2 + \sum_{i=1}^{m}\eta_i\sum_{k=1}^{n}(u_{ik}^{(f)})^{h_f}(1-u_{ik}^{(p)})^{h_p} \tag{3.31}$$

Para poder obtener el valor de η_i por primera vez, se requiere el cálculo de la distancia entre los centros y los vectores de entrada. El valor de estas distancias no debe ser ponderado usando w ya que las salidas estimadas de los centros, en la inicialización del algoritmo, no tienen los valores apropiados para medir la influencia de la salida de la función objetivo en la posición de los centros. Por tanto, el cálculo de η_i queda como:

$$\eta_i = \frac{\sum_{k=1}^{n} (u_{ik}^{(f)})^{h_f} d_{ik}^2}{(u_{ik}^{(f)})^{h_f}} \tag{3.32}$$

donde u_{ik} es el valor de pertenencia difusa obtenido previa ejecución del algoritmo FCM. La necesidad de ejecutar el FCM para poder calcular η_i hace que, al contrario que con las otras adaptaciones, esta no tenga desviación estándar igual a cero.

Como en todos los algoritmos previos basados en una partición difusa o posibilística, la solución se alcanza cuando todos los elementos definidos en la función de distorsión son minimizados empleando un proceso iterativo. En este caso concreto las ecuaciones para actualizar las variables son:

$$u_{ik}^{(p)} = \frac{1}{1 + \left(\frac{D_{ikW}}{\eta_i}\right)^{\frac{1}{h_p - 1}}} \tag{3.33}$$

$$u_{ik}^{(f)} = \sum_{j=1}^{m} \left(\frac{(u_{ik}^{(p)})^{(h_p-1)/2} D_{ikW}}{(u_{jk}^{(p)})^{(h_p-1)/2} D_{jkW}} \right)^{\frac{-2}{h_f - 1}} \tag{3.34}$$

$$\vec{c}_i = \frac{\sum_{k=1}^{n} (u_{ik}^{(p)})^{(h_p)} (u_{ik}^{(f)})^{(h_f)} \vec{x}_k w_{ik}^2}{\sum_{k=1}^{n} (u_{ik}^{(p)})^{(h_p)} (u_{ik}^{(f)})^{(h_f)} w_{ik}^2} \tag{3.35}$$

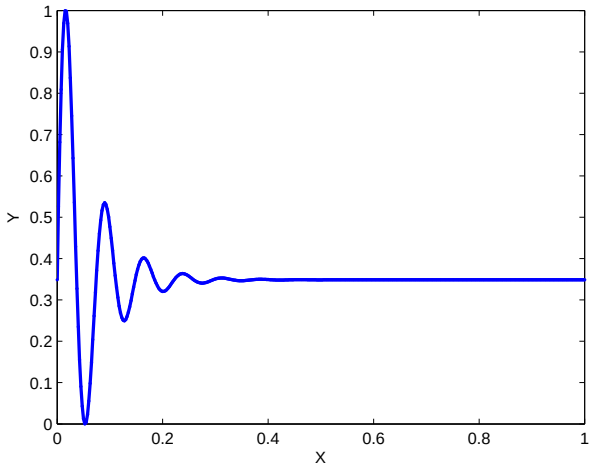


Fig. 3.8: Función objetivo f_1 con sus valores normalizados en el intervalo [0,1].

$$o_i = \frac{\sum_{k=1}^{n} (u_{ik}^{(p)})^{(h_p)} (u_{ik}^{(f)})^{(h_f)} y_k d_{ik}^2}{\sum_{k=1}^{n} (u_{ik}^{(p)})^{(h_p)} (u_{ik}^{(f)})^{(h_f)} d_{ik}^2} \tag{3.36}$$

3.2.4. Análisis experimental de los resultados

A continuación se realizará un análisis del comportamiento de los algoritmos en función de sus distintos parámetros. Para poder tener más claridad, los experimentos se realizarán con la función unidimensional f_1 (Fig. 3.8) que está definida como:

$$f_1 = \frac{\sin(25X)}{e^{(7X)}}, \tag{3.37}$$

Debido a que el número de centros exacto para modelar la función es desconocido, los algoritmos serán ejecutados utilizando distintos valores para el número de centros. Tras el completo diseño de cada RBFNN, se empleará

el NRMSE para determinar la calidad de la aproximación.

3.2.4.1. PCFA

El comportamiento del algoritmo PCM depende de la elección del valor para del parámetro η para cada centro. Este parámetro determina la distancia a la cual el valor de pertenencia de un vector de entrada para un centro toma el valor 0.5. El PCM también necesita la inicialización de otro parámetro,Kp, que es un número real positivo y que está presente en la formula para calcular η_i. El valor de η_i determina la anchura del *cluster* que está centrado en $\vec{c}_i$ de tal forma que si toma un valor muy grande, el *cluster* cubrirá todos los puntos de entrada, haciendo que se coloquen todos los centros en la misma posición (Barni et al., 1996). Por ejemplo, utilizando la función f_1, si Kp es pequeño, los centros serán colocados de forma equidistribuida a lo largo del espacio de entrada debido a que los puntos de entrada están equidistribuidos. Si el valor de Kp se incrementa, todos los vectores de entrada serán identificados como un único *cluster* y todos los centros se colocan en el centro de los puntos de entrada.

El algoritmo PCFA sufre el mismo problema que se presenta en el algoritmo PCM. Si los valores para η_i y Kp son calculados siguiendo las recomendaciones de los autores (Eq. 3.9), tenderá a identificar el mismo *cluster* colocando todos los centros en la misma posición. A diferencia del PCM, los centros, en vez de ser colocados en la mitad de los vectores de entrada, son colocados más cerca de la zona donde la función es más variable debido al parámetro w.

Debido a que los valores que proponen los autores para inicializar los parámetros Kp y η_i no son muy apropiados para el problema de aproximación funcional, se presenta la dificultad para elegir adecuadamente estos

valores. Para el problema de aproximación funcional no es posible definir una anchura predeterminada para un *cluster*, por tanto, el cálculo de este parámetro basándose en el resultado de la ejecución del FCM no es muy adecuado. El parámetro Kp determina la importancia de η_i haciéndolo más o menos grande. Debido a que η_i se puede calcular de forma automática, el problema real se centra en la elección del valor para Kp.

Conforme el valor de Kp decrece, los centros se situan de forma más distribuida. La razón de este comportamiento es debido a que un valor pequeño para Kp hará que el algoritmo converja rápidamente, haciendo que se mantenga la primera inicialización obtenida tras aplicar el FCM. Si, por el contrario, Kp toma valores grandes, los centros serán colocados en la misma posición, cerca de la zona donde la función es más variable. El equilibrio se encuentra cuando Kp es lo suficientemente grande como para permitir que los centros se muevan a la zona donde la función es más variable y lo suficientemente pequeño como para que converja antes de que los centros se sitúen en la misma posición.

En la Figura 3.9 se muestran varias ejecuciones con 5 centros variando el valor de Kp. En las primeras ejecuciones, no es posible distiguir los 5 centros ya que se encuentran aglutinados en la misma zona. A medida que el valor de Kp va disminuyendo, los centros comienzan a dispersarse.

En la Tabla 3.4 se muestran los errores de aproximación obtenidos tras el diseño completo de las RBFNN. Estos resultados reflejan la influencia dramática que Kp tiene en los resultados haciendo que, en general, los errores de aproximación no sean lo suficientemente pequeños.

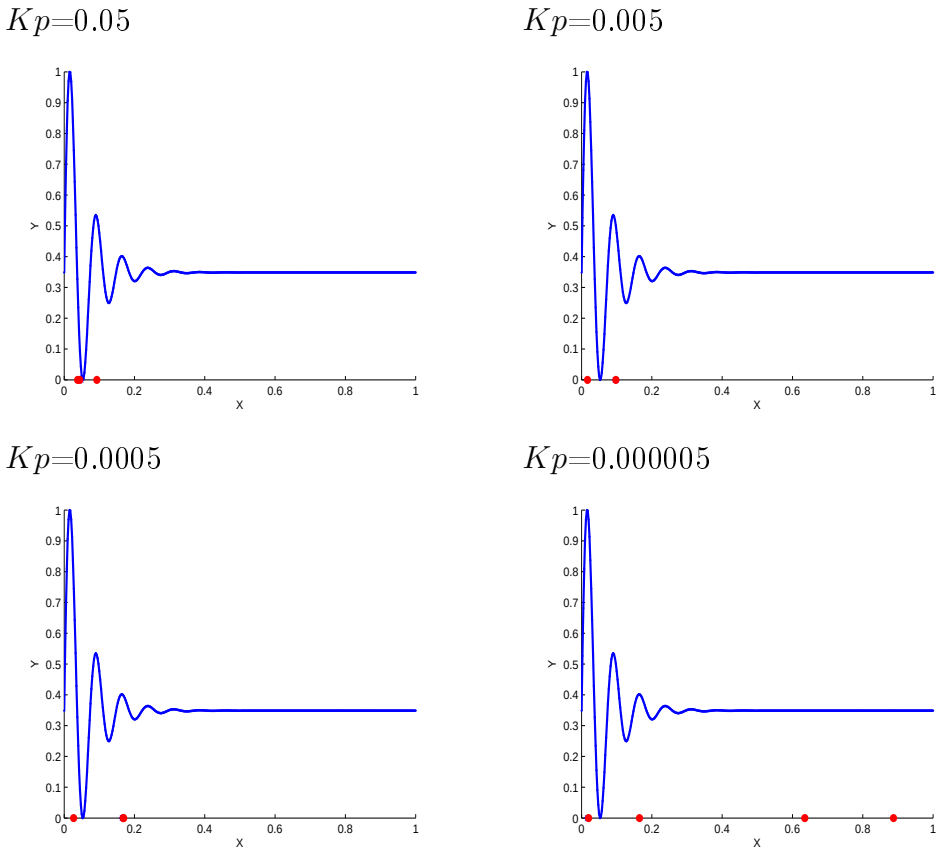


Fig. 3.9: Inicialización de los centros utilizando el algoritmo PCFA en función de varios valores de Kp.

		PCFA
Centros	Kp	Error de aproximación
6	0.000005	0.392(0.007)
	0.00005	0.381(0.112)
7	0.00005	0.473(0.226)
	0.0005	0.459(0.260)
8	0.00005	0.089(0.014)
	0.0005	0.276(0.204)
9	0.00005	0.430(0.004)
	0.0005	0.399(0.296)

Tab. 3.4: Media y desviación típica del error de aproximación (NRMSE) para la función f_1 usando el algoritmo PCFA.

3.2.4.2. FPCFA

El algoritmo FPCM resuelve de manera satisfactoria los problemas que se presentan cuando se utiliza el *clustering* posibilístico, evitando que los centros se coloquen en la misma posición. Cuando el algoritmo FPCM es ejecutado utilizando la función f_1, coloca los centros de manera uniforme a lo largo del espacio de los vectores de entrada.

La adaptación del algoritmo ICFA a este tipo de enfoque mixto tiene un comportamiento bastante bueno, situando los centros donde la función es más variable pero manteniendo un compromiso entre las distancias que separan a los centros. Hay dos parámetros que pueden ser ajustados para poder obtener mejores resultados, h y h_p.

El parámetro h controla cómo de difusa la partición será, es decir, el grado en el que un vector de entrada puede ser compartido entre varios centros. Si toma valores cercanos a uno, los centros compartirán en menor medida los vectores de entrada, obteniendo particiones prácticamente duras.

Sin embargo, si h aumenta, los vectores de entrada seran compartidos en mayor medida por los centros. En este caso, cuando h es grande, los centros podrán compartir entre todos la zona más variable, concentrándose en mayor medida en esa región. Cuando, h decrece, debido a que cada centro deberá poseer sus propios vectores de entrada, los centros tenderán a expandirse y separarse más.

El parámetro h_p tiene el mismo comportamiento que el parámetro h aunque su influencia es menor. La razón por la cual su influencia es menor es debido a que la restricción sobre la pertenencia posibilística (Eq. 3.17) afecta únicamente a los centros mientras que la restricción sobre la pertenencia difusa (Eq. 3.16) referencia a los vectores de entrada. Debido a que el número de vectores de entrada es considerablemente mayor que el número de centros, es lógico que h modifique en mayor medida el comportamiento del algoritmo.

Para ilustrar este hecho, se realizaron varias ejecuciones utilizando distintos valores para h y h_p, los resultados se muestran en la Figura 3.10. Tal y como ocurrió en la anterior subsección, en la figura, hay ocasiones en las que no se pueden distinguir los 5 centros debido a que están muy cercanos entre sí. Cuando h y h_p toman valores cercanos a uno, los centros se separan más, manteniendo un compromiso entre las áreas más y menos variables de la función. Cuando h toma valores cercanos a uno y h_p incrementa su valor, los centros se sitúan en mayor medida en la zona donde la función es más variable, dejando las zonas menos variables sin ningún centro. Si h es grande y h_p también, los centros se aglutinarán en las zonas más variables, llegando a solaparse entre ellos. Al disminuir el valor de h_p, los centros se separan un poco en comparación a la situación anterior aunque, como se comentó previamente, la influencia del parámetro h es mayor, haciendo que los centros todavía se solapen en un caso.

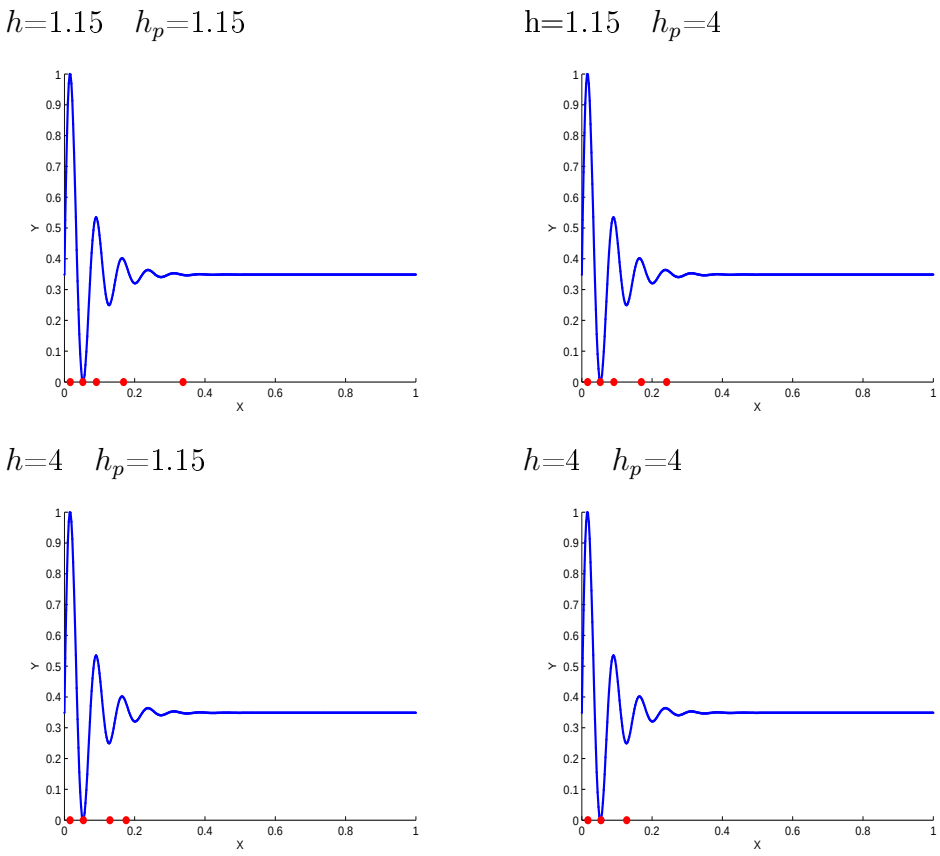


Fig. 3.10: Inicialización de los centros empleando el algoritmo FPCFA con distintos valores para los parámetros h y h_p.

En la Tabla 3.5 se muestran los errores de aproximación para la función f_1 utilizando varias combinaciones de valores para los parámetros h y h_p. El algoritmo FPCFA proporciona resultados bastante satisfactorios, no solo en lo que al error de aproximación se refiere sino a la robustez que estos resultados presentan. Un hecho a destacar es que, independientemente de los valores para h y h_p elegidos, el algoritmo proporciona buenas aproximaciones para este ejemplo.

FPCFA			
Centros	h	h_p	Error de aproximación
	1.5	1.5	0.244(0)
	1.5	2	0.043(0)
6	2	1.5	0.014(0)
	2	2	0.025(0)
	1.5	1.5	0.019(0)
	1.5	2	0.013(0)
7	2	1.5	0.092(0)
	2	2	0.020(0)
	1.5	1.5	0.004(0)
	1.5	2	0.067(0)
8	2	1.5	0.005(0)
	2	2	0.066(0)
	1.5	1.5	0.024(0)
	1.5	2	0.015(0)
9	2	1.5	0.004(0)
	2	2	0.021(0)

Tab. 3.5: Media y desviación típica del error de aproximación (NRMSE) para la función f_1 usando el algoritmo FPCFA.

3.2.4.3. IPCFA

El algoritmo IPCM, al igual que el FPCM, presenta un enfoque híbrido entre una partición difusa y una posibilística. Cuando este algoritmo es ejecutado sobre la función f_1, al igual que en el caso anterior, reparte los centros de manera uniforme por el espacio de entrada demostrando que no padece los problemas que presenta el PCM.

Al provenir directamente del enfoque posibilístico, es necesario dar valor a los mismos parámetros que requería el PCM además de al exponente para la función de pertenencia que requiere el enfoque difuso.

La función de pertenencia difusa afecta más al comportamiento del algoritmo ya que, en la función de distorsión (Eq. 3.31), el resto de elementos se encuentran multiplicados por ella. La consecuencia de esto es que cuando el exponente h_f que modifica el valor de pertenencia difuso es pequeño, las variaciones en el exponente que modifica la pertenencia posibilistica, h_p, no son muy significativas. Esto es debido a que al decrecer el valor de h_f, los valores de pertenencia tienden a ser más grandes haciendo que la función de distorsión se vea modificada por estos en mayor medida. Al igual que en el algoritmo anterior, conforme el valor de h_f se acerca a uno, más distribuidos estarán los centros a lo largo del espacio de entrada y se irán concentrando en las zonas más variables a medida que el valor de h_f crezca. Por el contrario, cuanto mayor sea el valor de h_f, más concentrados estarán los centros en las zonas variables.

Conforme el valor de h_f aumenta, la importancia del parámetro h_p se incrementa ya que los valores de la función de pertenencia difusa serán más pequeños haciendo que la función de distorsión tienda a cero y gracias a h_p se puede incrementar o decrementar el valor de la distorsión. Al igual que con h_f, si h_p es pequeño, los centros estarán más dispersos al no compartirse en

IPCFA			
RBF	h_f	h_p	Error de aproximación
	1.5	2	0.038(0.030)
6	2	2	0.265(0.289)
	2	1.5	0.082(0.059)
	1.5	2	0.102(0.064)
7	2	2	0.070(0.090)
	2	1.5	0.075(0.043)
	1.5	2	0.018(0.012)
8	2	2	0.045(0.021)
	2	1.5	0.030(0.042)
	1.5	2	0.095(0.058)
9	2	2	0.013(0.015)
	2	1.5	0.073(0.089)

Tab. 3.6: Media y desviación típica del error de aproximación (NRMSE) para la función f_1 usando el agoritmo IPCFA.

gran medida los vectores de entrada y, si aumenta su valor, el grado en el que los centros compartirán los vectores de entrada será mayor y se colocarán en las zonas más variables.

Tal y como se hizo para los anteriores algoritmos, en la Figura 3.11 se muestran varias ejecuciones con distintos valores para los parámetros h_f y h_p, los correspondientes errores de aproximación se muestran en la Tabla 3.6.

La elección del valor para el parámetro η_i también influencia el comportamiento del algoritmo aunque en esta ocasión, la inicialización propuesta por los autores del IPCM es lo suficientemente buena como para obtener configuraciones finales sin excesivo solapamiento entre los centros.

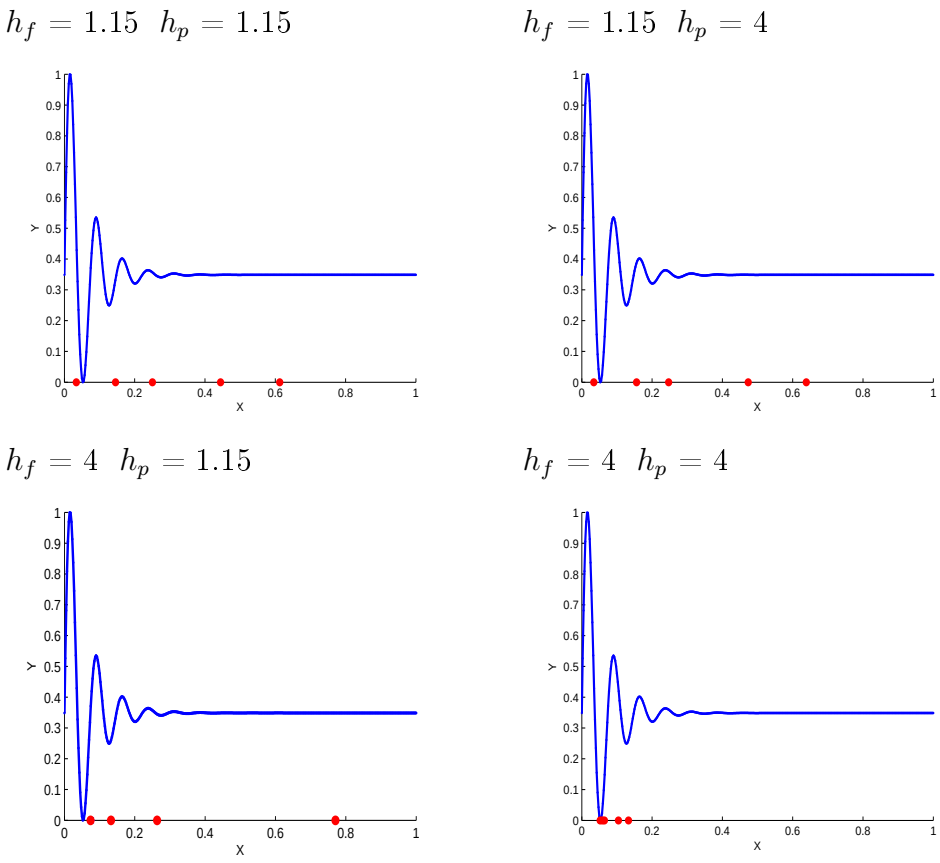


Fig. 3.11: Inicialización de los centros empleando el algoritmo IPCFA con distintos valores para los parámetros h_f y h_p.

3.2.5. Resultados experimentales

A continuación se realizará una comparativa de los algoritmos descritos en las subsecciones previas. El ICFA con la partición difusa original será comparado con el IPCFA y el FPCFA, el algoritmo PCFA no será considerado debido a su irregular comportamiento en función de los parámetros necesarios para su ejecución.

Los experimentos consisten en primer lugar, una aproximación de una función unidimensional en situación normal y en presencia de ruido. En el segundo experimento, se ha diseñado una función de dos dimensiones cuyas posiciones originales de los centros son conocidas así que es posible comparar los algoritmos para ver cuál posiciona los centros más cerca de los originales. Tras esto, dos funciones sintéticas en presencia de ruido y un problema real serán empleados para realizar la comparativa.

3.2.5.1. Función unidimensional

En este experimento se aproximará la función de una dimensión presentada en la subsección anterior, primero en condiciones normales y después, con la adición de ruido.

3.2.5.1.1. Ausencia de ruido

Los algoritmos fueron ejecutados buscando la mejor combinación de los parámetros que necesitan ser definidos para cada uno de ellos, los resultados se muestran en la Tabla 3.7. Estos resultados muestran cómo el FPCFA supera a los otros dos algoritmos de manera muy significativa. El error de aproximación obtenido es menor y, al igual que el ICFA, presenta una gran robustez. El IPCFA también presenta un buen comportamiento aunque carece de robustez en comparación con los otros dos algoritmos.

RBF	**ICFA**	**FPCFA**	**IPCFA**
6	0.280(0)	0.014(0)	0.265(0.289)
7	0.073(0)	0.092(0)	0.070(0.090)
8	0.033(0)	0.005(0)	0.045(0.021)
9	0.045(0)	0.004(0)	0.013(0.015)

Tab. 3.7: Media y desviación típica del error de aproximación (NRMSE) para la función f_1.

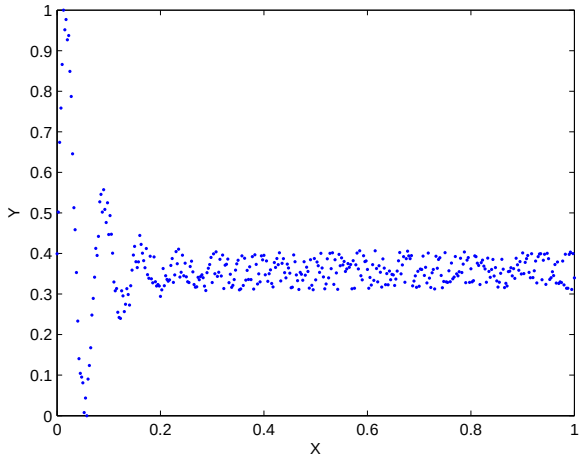


Fig. 3.12: Función objetivo f_1 con ruido añadido.

3.2.5.1.2. Presencia de ruido

La función fue distorsionada mediante la adición de ruido uniforme en el intervalo [0,1] para cada punto:

$$y_k^{noise} = y_k + 0{,}1 \cdot rand \tag{3.38}$$

donde rand es un numero aleatorio en una distribución uniforme.

El conjunto de datos de entrenamiento se muestra en la Figura 3.12 donde, aunque la función ha sido bastante distorsionada, todavía es posible distin-

RBF	**ICFA**	**FPCFA**	**IPCFA**
6	0.141(0)	0.141(0)	0.125(0.018)
7	0.120(0)	0.079(0)	0.103(0.043)
8	0.089(0)	0.077(0)	0.100(0.054)
9	0.075(0)	0.049(0)	0.078(0.035)

Tab. 3.8: Media y desviación típica del error de aproximación (NRMSE) para la función f_1 usando el conjunto de datos de test.

$\vec{c}_i$	r_i	Ω_i
7.0026 3.7539	0.8687	0.0377
8.8390 1.9328	4.4933	0.7343
9.4336 20.5478	3.7750	0.3106
15.3469 12.7493	2.6281	0.8203
24.8801 10.8553	4.8836	0.8785

Tab. 3.9: Parámetros para la función bidimensional.

guir una zona donde la función es más variable. Una vez generadas las correspondientes RBFNN, se empleó como datos de test las muestras originales en ausencia de ruido. Los resultados se muestran en la Tabla 3.8. En esta ocasión también se puede apreciar cómo el FPCFA supera a los otros algoritmos indicando que las RBFNN que genera tienen mayor capacidad de interpolación.

3.2.5.2. Función bidimensional

Para este experimento se ha diseñado una función de dos dimensiones utilizando una RBFNN gausiana sobre una malla de 25×25 puntos utilizando los parámetros que se muestran en la Tabla 3.9 que fueron posteriormente normalizados entre 0 y 1.

Los errores de aproximación obtenidos tras diseñar las RBFNN se mues-

	NRMSE
ICFA	0.0077 (0)
FPCFA	1.7301e-008 (0)
IPCFA	0.0062 (0.0035)

Tab. 3.10: Media y desviación típica del error de aproximación (NRMSE) para la función bidimensional.

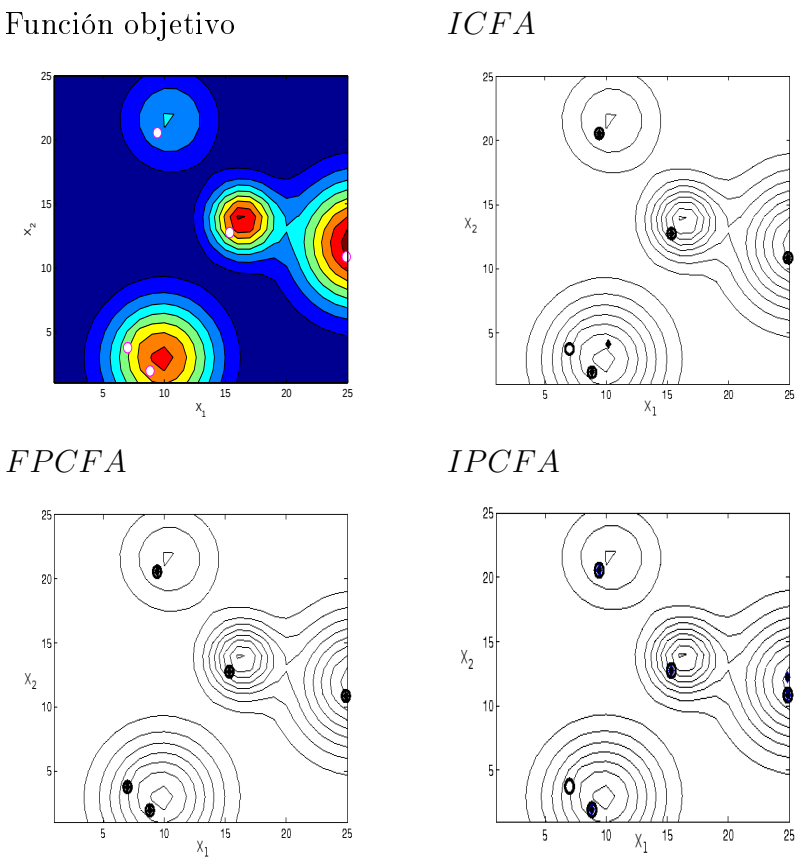


Fig. 3.13: Posiciones originales de los centros (círculos) y las diferentes inicializaciones realizadas por los algoritmos ICFA,FPCFA e IPCFA (diamantes).

tran en la Tabla 3.10 donde se puede observar que el algoritmo FPCFA proporciona unos resultados óptimos. En la Figura 3.13 a) se muestra el contorno de la función bidimensional con las posiciones de los centros originales. En la misma figura en los apartados b), c) y d) están representados los resultados de las ejecuciones de los algoritmos junto con los centros originales. Las posiciones donde los algoritmos han situado los centros están marcadas con un diamante mientras que las posiciones de los centros originales con un círculo sin relleno. En los casos en los que los algoritmos han colocado un centro en la misma posición que el centro original, se puede apreciar cómo el círculo sin relleno aparece ahora prácticamente rellenado por completo.

El algoritmo FPCFA, tal y como indicaba el error de aproximación, es capaz de identificar las posiciones originales de todos los centros, proporcionando la solución óptima. Tanto el algoritmo ICFA como el IPCFA sitúan cuatro de los cinco centros en las posiciones correctas. Sin embargo, el IPCFA obtiene mejores resultados que el ICFA ya que sitúa dos centros en la zona donde la función es más variable.

3.2.5.3. Benchmarks bidimensionales

En esta subsección se presentan los resultados que proporcionan los tres algoritmos comparados empleando dos funciones de dos dimensiones en presencia de ruido.

3.2.5.4. Función sintética f_3

La función objetivo para este experimento fue presentada en (Cherkassky and H.Lari-Najafi, 1991) y ha sido utilizada como *benchmark* en (Pomares, 2000); (Friedman, 1981); (Friedman, 1991). La función está definida como:

	ICFA			
RBF	h=**1.5**	h=**2**	h=**2.5**	Error medio
7	0.522(0)	0.575(0)	0.575(0)	0.557(0.030)
9	0.349(0)	0.435(0)	0.447(0)	0.410(0.053)
11	0.256(0)	0.340(0)	0.244(0)	0.280(0.052)
13	0.164(0)	0.197(0)	0.216(0)	0.192(0.026)
15	0.158(0)	0.148(0)	0.196(0)	0.167(0.025)

Tab. 3.11: Media y desviación estándar del error de aproximación (NRMSE) para la función f_3 usando el algoritmo ICFA.

$$f_3(x_1, x_2) = \frac{1 + sin(2x_1 + 3x_2)}{3{,}5 + sin(x_1 - x_2)} \; x_1, x_2 \in [-2, 2] \tag{3.39}$$

En la Figura 3.14 se muestran la función original y el conjunto de datos utilizado en el experimento. Este conjunto de datos se obtuvo seleccionando aleatoriamente puntos de la función original y añadiéndoles ruido utilizando la siguiente ecuación:

$$y_k^s = f_3(x_1, x_2) + (0{,}1(0{,}5 - randn)); \tag{3.40}$$

donde randn es un número aleatorio en una distribución normal de media 0 y varianza 1. Se han empleado 400 vectores de entrada obtenidos de forma aleatoria para el entrenamiento y 1936 para realizar el test escogidos de forma uniforme y equidistribuida.

La Tablas 3.11, 3.12, y 3.13 muestran los resultados de las ejecuciones de los algoritmos ICFA, FPCFA e IPCFA respectivamente utilizando varios valores para los parámetros que cada uno requiere. Los resultados muestran cómo todos los algoritmos tienen un comportamiento bastante similar, proporcionando errores de aproximación similares. Sin embargo, el ICFA puede destacarse levemente sobre los otros ya que es más independiente del valor

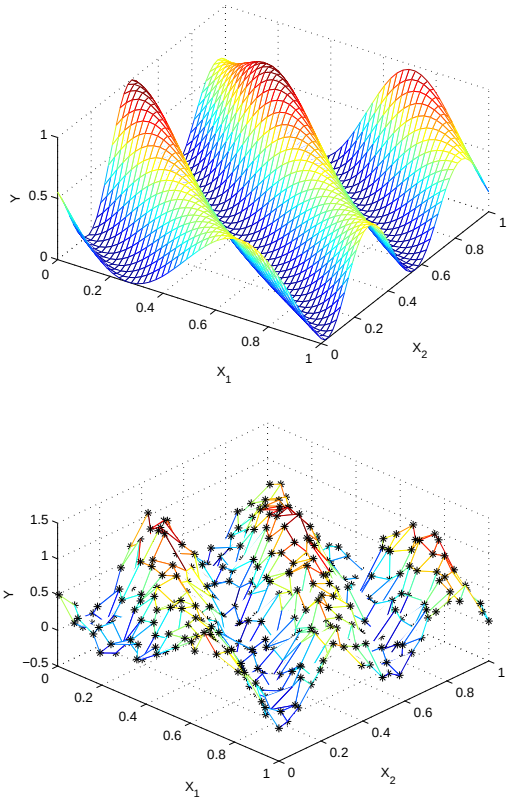


Fig. 3.14: Función objetivo f_3 y conjunto de datos de entrenamiento con ruido añadido.

		FPCFA			
RBF	h=**1.5** h_p=**1.5**	h=**1.5** h_p=**2**	h=**2** h_p=**1.5**	h=**2** h_p=**2**	Error medio
7	0.522(0)	0.522(0)	0.592(0)	0.575(0)	0.552(0.036)
9	0.801(0)	0.572(0)	0.333(0)	0.654(0)	0.590(0.195)
11	0.237(0)	0.243(0)	0.198(0)	0.332(0)	0.250(0.057)
13	0.156(0)	0.358(0)	0.228(0)	0.231(0)	0.241(0.086)
15	0.222(0)	0.161(0)	0.127(0)	0.256(0)	0.191(0.057)

Tab. 3.12: Media y desviación estándar del error de aproximación (NRMSE) para la función f_3 usando el algoritmo FPCFA.

	IPCFA			
RBF	h=**1.5** h_p=**2**	h=**2** h_p=**1.5**	h=**2** h_p=**2**	Error medio
7	0.589(0.073)	0.476(0.100)	0.540(0.148)	0.535(0.056)
9	0.469(0.110)	0.467(0.157)	0.412(0.167)	0.449(0.032)
11	0.310(0.107)	0.275(0.037)	0.270(0.036)	0.285(0.021)
13	0.198(0.037)	0.256(0.066)	0.185(0.062)	0.213(0.037)
15	0.183(0.024)	0.150(0.037)	0.148(0.022)	0.160(0.019)

Tab. 3.13: Media y desviación estándar del error de aproximación (NRMSE) para la función f_3 usando el algoritmo IPCFA.

de h, obteniendo aproximaciones muy similares para cualquier valor de este parámetro. No obstante, para ciertos valores concretos de los parámetros, el IPCFA y el FPCFA obtienen menor error de aproximación que el ICFA.

3.2.5.5. Función Sintética f_4

Al igual que en la función anterior, la función que se empleará en este experimento fue presentada en (Cherkassky and H.Lari-Najafi, 1991) y está definida como:

$$f_4(x_1, x_2) = 1{,}9[1{,}35 + e^{x_1} sin(13(x_1 - 0{,}6)^2) e^{-x_2} sin(7x_2)] \; x_1, x_2 \in [0, 1] \tag{3.41}$$

Los conjuntos de entrenamiento y test fueron generados del mismo modo que en el experimento anterior. En la Figura 3.15 se muestra la función original y el conjunto de datos de entrenamiento empleado en el experimento y las Tablas 3.14, 3.15, y 3.16 muestran los correspondientes errores de aproximación. Tal y como ocurrió en el ejemplo anterior, los tres algoritmos presentan un comportamiento muy similar.

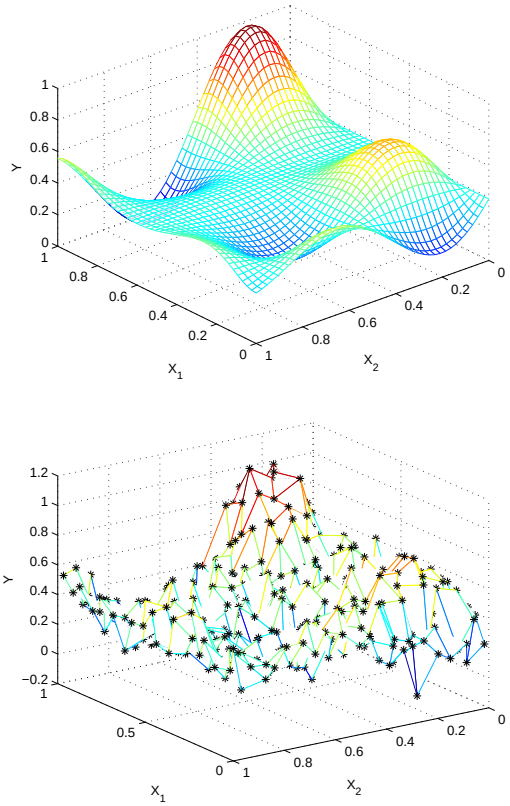


Fig. 3.15: Función objetivo f_4 y conjunto de entrenamiento con ruido.

		ICFA		
RBF	h=**1.5**	h=**2**	h=**2.5**	Error medio
7	0.317(0)	0.342(0)	0.283(0)	0.314(0.029)
9	0.181(0)	0.185(0)	0.307(0)	0.224(0.071)
11	0.241(0)	0.170(0)	0.207(0)	0.206(0.035)
13	0.194(0)	0.188(0)	0.157(0)	0.179(0.019)
15	0.164(0)	0.204(0)	0.176(0)	0.181(0.020)

Tab. 3.14: Media y desviación estándar del error de aproximación (NRMSE) para la función f_4 usando el algoritmo ICFA.

	FPCFA				
RBF	h=**1.5** h_p=**1.5**	h=**1.5** h_p=**2**	h=**2** h_p=**1.5**	h=**2** h_p=**2**	Error medio
7	0.234(0)	0.191(0)	0.371(0)	0.981(0)	0.444(0.366)
9	0.195(0)	0.186(0)	0.231(0)	0.229(0)	0.210(0.023)
11	0.217(0)	0.183(0)	0.186(0)	0.217(0)	0.200(0.018)
13	0.167(0)	0.183(0)	0.200(0)	0.160(0)	0.177(0.017)
15	0.196(0)	0.177(0)	0.152(0)	0.220(0)	0.186(0.028)

Tab. 3.15: Media y desviación estándar del error de aproximación (NRMSE) para la función f_4 usando el algoritmo FPCFA.

	IPCFA			
RBF	h=**1.5** h_p=**2**	h=**2** h_p=**1.5**	h=**2** h_p=**2**	Error medio
7	0.300(0.090)	0.283(0.076)	0.321(0.070)	0.301(0.019)
9	0.231(0.080)	0.262(0.057)	0.216(0.044)	0.236(0.023)
11	0.206(0.009)	0.293(0.177)	0.244(0.057)	0.247(0.043)
13	0.196(0.033)	0.166(0.006)	0.186(0.031)	0.182(0.015)
15	0.205(0.013)	0.191(0.019)	0.188(0.007)	0.194(0.009)

Tab. 3.16: Media y desviación estándar del error de aproximación (NRMSE) para la función f_4 usando el algoritmo IPCFA.

3.2.5.6. Problemas reales

3.2.5.7. Tiempo de respuesta de un servo-motor

Este ejemplo ha sido extraido del University of California Irvine Machine Learning Repository (UCI) (S. Hettich and Merz, 1998) y consiste, tal y como es descrito en la base de datos de la UCI, en:

> Un sistema que envuelve un servo amplificador, un motor, un tornillo/tuerca y una plataforma de transporte deslizante. Puede haber pertenecido a los ejes de traslación de un robot en la novena planta del laboratorio de inteligencia artificial. En cualquier caso, el valor de salida es, casi con certeza, un tiempo de elevación o el tiempo requerido por el sistema para responder a un cambio de paso en una posición dada.
>
> Esta es una colección interesante de datos proporcionada por Karl Ulrich. Cubre un fenómeno extremadamente no lineal - la predicción del tiempo de elevación de un servomecanismo en términos de dos (continuos) valores de ganancia y dos (discretas) opciones de enlaces mecánicos.

El conjunto de datos contiene 167 vectores de entrada con 4 variables y una salida:

- motor: A,B,C,D,E
- tornillo: A,B,C,D,E
- ganacia p: 3,4,5,6
- ganacia v: 1,2,3,4,5

RBF	**ICFA**	**FPCFA**	**IPCFA**
5	0.378(0.091)	0.419(0.046)	0.373(0.101)
6	0.403(0.127)	0.427(0.068)	0.404(0.101)
7	0.377(0.015)	0.353(0.060)	0.369(0.082)
8	0.377(0.056)	0.296(0.063)	0.284(0.021)
9	0.972(0.876)	0.420(0.102)	0.620(0.390)

Tab. 3.17: Media y desviación estándar del error de aproximación (NRMSE) para el conjunto de test del problema del servo-motor utilizando distintos valores de los parámetros que requiere cada algoritmo.

- clase: 0.13 hasta 7.10

Las variables motor y tornillo fueron mapeadas a los números enteros del 1 al 5. De los 167 datos disponibles, 126 han sido utilizados para entrenamiento y los restantes para test. Los resultados se muestran en la Tabla 3.17 donde se puede observar la media de los errores de test tras diversas ejecuciones de los algoritmos variando sus distintos parámetros. En general, los tres algoritmos tienen un comportamiento similar, demostrando lo complicado que puede resultar aproximar funciones con datos reales y poco muestreados.

3.3. Algoritmo para la inicialización de los centros de los radios (OVI: Ouput Value-Based Initializer)

Tras haber analizado diversos algoritmos cuya finalidad es únicamente la inicialización de los centros, esta sección describirá un algoritmo que tiene la capacidad de inicializar tanto los centros como los radios de las funciones de base radial. La mejora de los resultados a través de una correcta inicialización de los centros tiene sus limitaciones y es necesario seguir buscando una disminución del error de aproximación mediante la búsqueda de métodos más

adecuados para la inicialización de los radios. En los algoritmos mostrados hasta ahora, al introducir el elemento de supervisión, la interpretabilidad de las funciones de pertenencia tanto difusas como posibilísticas se distorsionaba, perdiendo totalmente su significado. Esto es lo que ha motivado el diseño de este nuevo algoritmo que ha sido denominado como OVI (Output Value-Based Initializer) (**Guillén** et al., 2007b); (**Guillén** et al., 2006b)ya que emplea los valores de la salida de la función objetivo para ser modelada como elemento supervisor tanto en la inicialización de los centros como en la de los radios.

3.3.1. Esquema general del algoritmo OVI

Del mismo modo que en los algoritmos mostrados anteriormente, el OVI define una función de distorsión que es minimizada mediante un proceso de optimización que alterna el cálculo de las posiciones de los centros y los grados de activación para cada vector de entrada. En cada iteración se actualizan las posiciones de los centros y los grados de activación de las neuronas de modo que el algoritmo se detendrá cuando estos elementos no cambien significativamente. Tras la convergencia del algoritmo, la matriz A contendrá el grado en el que cada neurona es activada por cada vector de entrada y, gracias a esto, será posible calcular la longitud del radio para cada RBF. El esquema general propuesto para el algoritmo se muestra en la Figura 3.16.

El algoritmo comienza inicializando los centros empleando el algoritmo FCM tal y como se ha hecho en otros algoritmos presentados en la literatura (Zhang and Leung, 2004) teniendo como objetivo el alcanzar la mayor robustez posible. Debido al carácter de búsqueda local del algoritmo que se aplica para el ajuste fino de los centros y los radios, una pequeña variación

en el punto de partida de estos puede hacer que los resultados sean muy dispares.

3.3.2. Función de distorsión

Ya es conocida la importancia de tener en cuenta el valor de la salida a la hora de inicializar los centros, debido a ello, en la función de distorsión debe haber un elemento que permita al algoritmo identificar las zonas de la función objetivo donde las neuronas deben ser más activadas. La función de distorsión es definida como:

$$J_{l,p}(A,C) = \sum_{k=1}^{n}\sum_{i=1}^{m} D_{ik}^2 a_{ik}^l |Y_k^p| \tag{3.42}$$

donde D_{ik} representa la distancia Euclídea desde un centro $\vec{c}_i$ a un vector de entrada $\vec{x}_k$, a_{ik} es el grado en el que la neurona $\vec{c}_i$ es activada por el vector de entrada $\vec{x}_k$, l es un parámetro que controlará el grado en solapamiento de las neuronas, Y_k es la salida preprocesada del vector de entrada $\vec{x}_k$ y p es un parámetro que permite modificar la influencia de la salida a la hora de calcular los grados de activación, haciéndola mayor o menor.

Esta función de distorsión combina la información de la salida de la función objetivo con los valores de las coordenadas de las posiciones de las neuronas. Esta combinación permite que cuando un vector de entrada con un valor alto en la salida esté cerca de una neurona, ésta se active en mayor medida.

La salida de la función objetivo debe ser preprocesada antes de comenzar la ejecución del algoritmo. El planteamiento que se hace es el siguiente: la salida de la función objetivo es una superficie plana e igual a cero ($Y(\vec{x}_k) = 0$), donde algunos valores de esta superficie han sido modificados por distintos elementos n-dimensionales, generando y_k. Partiendo de esta afirmación se

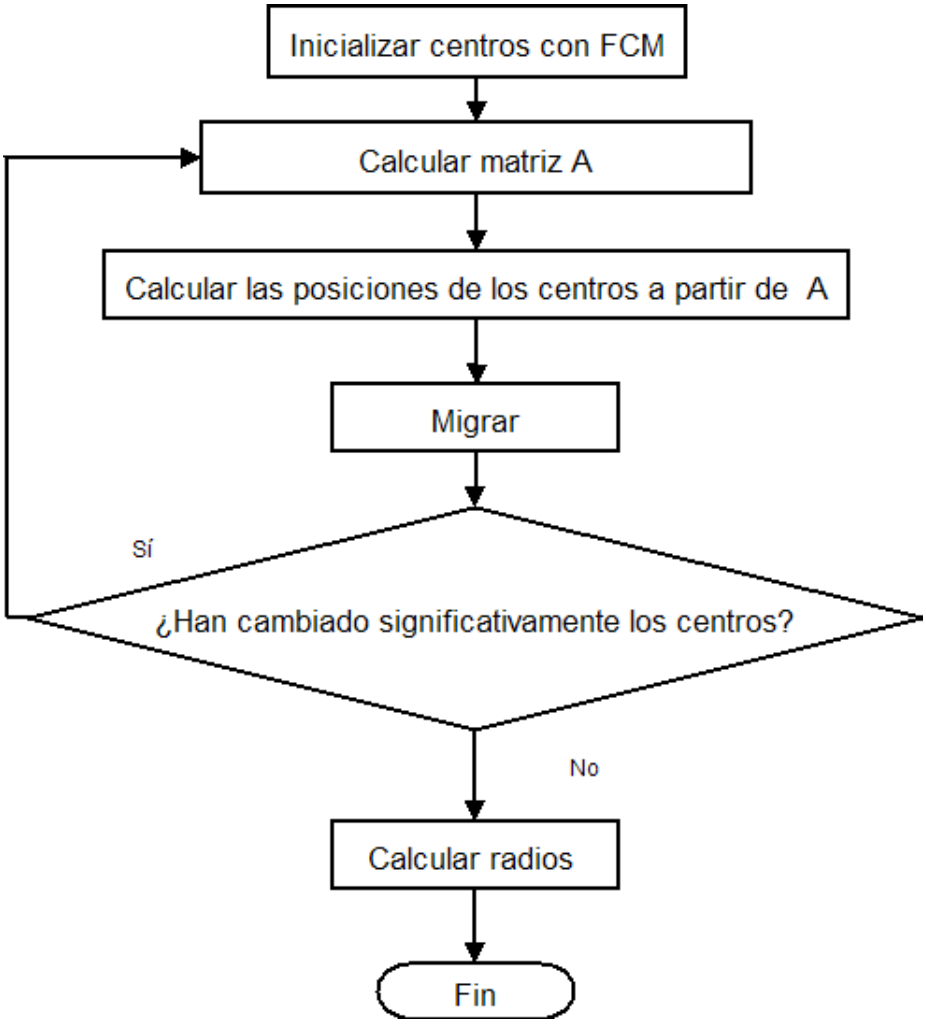


Fig. 3.16: Esquema general del algoritmo OVI.

puede calcular el valor más frecuente en la salida de la función objetivo y hacerlo igual a 0. El valor más frecuente debe ser calculado empleando el concepto de moda difusa para que, en situaciones de ruido, este valor siga siendo representativo. Una vez que se ha preprocesado la salida, tendremos que la función de distorsión solamente es alterada por los valores que tienen una salida significativa haciendo que los cercanos a cero, apenas la modifiquen.

Sea $A = [a_{ik}]$ la matriz de activación con las siguientes restricciones:

- $\sum a_{ik} = 1 \ \forall k = 1...n$
- $0 < \sum a_{ik} < n \ \forall i = 1...m.$

La primera restricción fuerza a cada vector de entrada a activar como mínimo a una neurona siendo 1 el valor máximo de activación. Gracias a la segunda, se puede garantizar que las neuronas serán activadas por al menos un vector de entrada. Esta restricción también nos permite mantener un solapamiento entre las neuronas permitiendo una interpolación más uniforme de la función.

El algoritmo trata de calcular un mínimo de la función de distorsión de modo que las neuronas sólo sean activadas cuando sea necesario, es decir, cuando esté cerca un vector de entrada que produzca un valor alto en la salida. Por tanto, el mínimo implicará que el valor de activación a_{ik} de una neurona con respecto a un vector sea alto si este vector produce una salida alta. Para poder llegar a esta situación de equilibrio, se emplea el procedimiento de optimización alterna usado en los anteriores algoritmos dado que se garantiza la convergencia a un mínimo de la función. Las ecuaciones que permiten calcular las posiciones de las neuronas y sus grados de activación son:

$$a_{ik} = \left(\sum_{j=1}^{m} \left(\frac{D_{ik}}{D_{jk}} \right)^{\frac{2}{l-1}} \right)^{-1} \tag{3.43}$$

$$\vec{c}_i = \frac{\sum_{k=1}^{n} a_{ik}^{l} \vec{x}_k |Y_k^p|}{\sum_{k=1}^{n} a_{ik}^{h} |Y_k^p|} \tag{3.44}$$

donde D_{ik} es la distancia euclidea entre $\vec{x}_k$ y $\vec{c}_i$, y Y_k es la salida preprocesada de la función a aproximar.

3.3.3. Paso de migración

Tal y como se mostró con previos algoritmos de descenso en gradiente, existe la posibilidad de caer en un mínimo local. Por tanto, se ha incorporado a este algoritmo un paso de migración similar al incorporado en el algoritmo ICFA aunque se ha considerado el aspecto de evaluar la posibilidad de migrar varios centros dentro del mismo paso de migración. En este caso la eficiencia no es tan crítica puesto que no se ha de actualizar ningún valor de salida estimada para un centro tal y como ocurre en el algoritmo ICFA.

3.3.4. Cálculo del radio de las RBF

Como se ha comentado en varias ocasiones a lo largo de esta memoria, una vez fijado la posición de los centros, debe realizarse la asignación de los valores de los radios. Aunque en (Park and Sandberg, 1993a); (Park and Sandberg, 1991b) se demostró que una RBFNN en la que todas las neuronas posean el mismo radio puede aproximar cualquier función, en (Benoudjit and Verleysen, 2003); (Musavi et al., 1992a) se muestra cómo es posible mejorar la aproximación si cada RBF posee un valor propio para su radio. Por tanto, el valor del radio es un elemento importante que puede mejorar notablemente el comportamiento de la red.

El algoritmo OVI, gracias a el cálculo de los grados de activación basándose en la salida preprocesada, es capaz de determinar un valor para el radio

de cada neurona. Este valor es igual a la distancia del vector más lejano que activa a la neurona. Para poder decidir si un vector de entrada activa una neurona o no, debe definirse un umbral de activación que controlará la longitud del radio. Para ilustrar los posibles casos se utilizará una función de ejemplo f_5 (Figure 3.21) definida como:

$$f_5(x) = \frac{\sin(2\pi x)}{e^x},\ x \in [0, 10] \tag{3.45}$$

Esta función ha sido diseña con el objetivo de mostrar la importancia de elegir un buen valor para el radio de las neuronas. En unas zonas, dado que la variabilidad de la salida es muy alta en una zona muy concreta, se necesitarán valores de radio pequeños mientras que en zonas más extensas con menos variabilidad, los radios deberán ser más grandes.

Tras la ejecución del algoritmo, los valores de activación de las neuronas están almacenados en la matriz A. La Figura 3.18 muestra estos valores de activación pudiendo apreciarse qué vectores de entrada activan a cada neurona. Se ha de definir un umbral ($\vartheta_{overlap}$) para determinar qué valor de activación puede considerarse valido puesto que si un vector de entrada activa la neurona con un valor muy pequeño, no es significativo. Una vez definido el umbral, se puede calcular la longitud del radio haciéndola igual a la distancia Euclídea entre la posición de la neurona y la posición del vector de entrada con un valor de activación para esa neurona igual al valor del umbral:

$$r_i = \text{máx}\{\ D_{ik}\ /\ a_{ik} > \vartheta_{overlap}\ ,\ 1 \leq i \leq m\ ,\ 1 \leq k \leq n\ \} \tag{3.46}$$

Este método hace el cálculo del radio de manera independiente a la posición de las otras neuronas, al contrario que la heurística KNN, pero manteniendo el solapamiento entre los radios de unas neuronas con otras y no

como en la heurística CIV. Lo más significativo de esta nueva técnica es que inicializa los radios teniendo en cuenta la salida de la función a aproximar puesto que los valores de activación son proporcionales a los valores de salida de los vectores de entrada.

El tener que fijar un valor umbral puede parecer a priori un inconveniente aunque hace al algoritmo más flexible en caso de poseer un experto supervisando el procedimiento de aprendizaje. Además el valor del umbral podría optimizarse mediante un procedimiento automático en el que se vaya incrementando/disminuyendo su valor en intervalos fijos hasta obtener el que mejores resultados proporcione. En comparación con las otras técnicas, el algoritmo KNN también requiere la elección de un valor para k. En la Figura 3.19 se representan los valores de los radios en función de distintos valores del umbral $\vartheta_{overlap}$. Se puede observar cómo para pequeños valores del umbral se obtiene un gran solapamiento de las neuronas y conforme el valor del umbral crece, este solapamiento disminuye.

3.3.5. Experimentos

En esta subsección se muestra el comportamiento del algoritmo OVI en función de sus parámetros además de la comparación del algoritmo OVI con otros algoritmos dedicados a la inicialización de los parámetros de una RBFNN o al diseño integral de la RBFNN.

En primer lugar, se analiza el efecto del parámetro p a la hora de posicionar las neuronas en el espacio de entrada. Además, se realiza una comparación de las distintas heurísticas usadas tradicionalmente para inicializar los radios con la nueva metodología propuesta por OVI. En el segundo experimento, el algoritmo OVI es comparado con distintas metodologías para la inicialización de los centros de las RBF. En el tercer experimento se aproxima

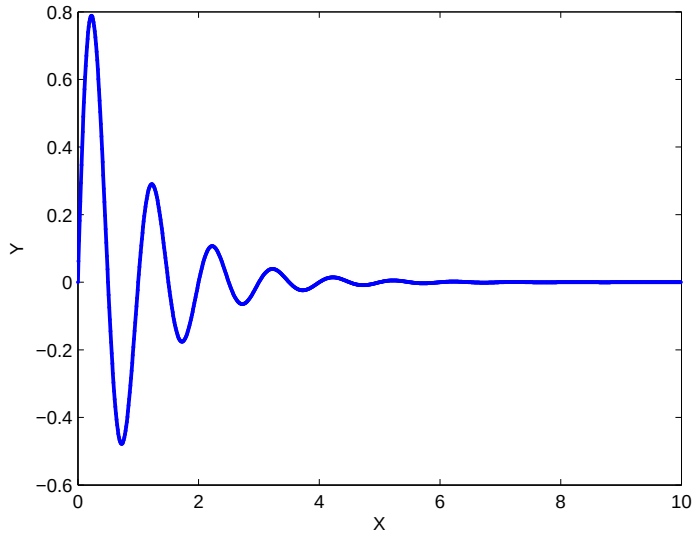


Fig. 3.17: Función objetivo f_1.

una función bidimensional que ha sido usada como benchmark en distintos trabajos de la bibliografía y se compara con los distintos algoritmos que la aproximan. Por último, se compara el algoritmo con otros presentados en la bibliografía para realizar la aproximación de la serie temporal Box-Jenkins presentada en (Box and Jenkins, 1976a).

3.3.5.1. Experimento 1: Influencia de p

La función que propone minimizar el algoritmo posee un parámetro$p$ que permite regular la influencia de la salida de la función a aproximar sobre la posición final de las neuronas. En este experimento se han realizado diversas ejecuciones utilizando distintos valores de este parámetro con el objetivo de analizar su influencia. La función a aproximar es f_1, la cual fue presentada en la subsección anterior para ver el efecto del cálculo de los radios. Como ya se comentó, esta función requiere un diseño adecuado para obtener buenos

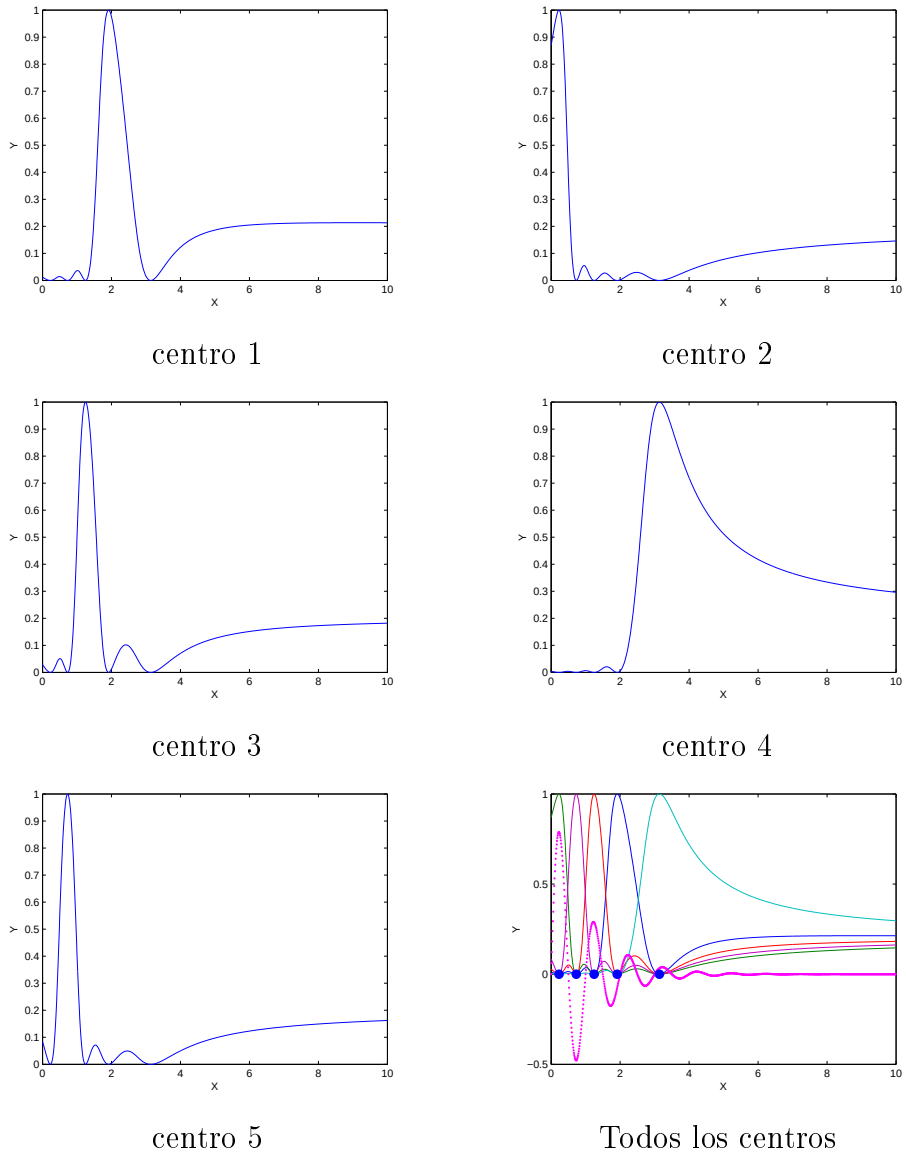


Fig. 3.18: Grados de activación para 5 neuronas tras la ejecución del algoritmo OVI usando como entrada la función f_1.

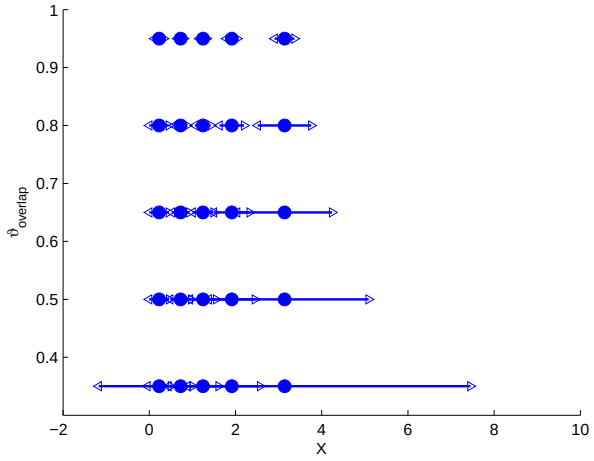


Fig. 3.19: Posiciones de las neuronas y sus valores correspondientes para los radios en función de $\vartheta_{overlap}$.

resultados de aproximación puesto que presenta zonas difíciles de modelar debido a su alta variabilidad en la salida. Gracias al parámetro p, es posible concentrar más centros en el intervalo donde la función es más variable tal y como se muestra en la Figura 3.20 donde se representan las posiciones de los centros de las RBF tras la ejecución del algoritmo usando distintos valores para p.

Los errores de aproximación obtenidos de las distintas ejecuciones tras generar las correspondientes RBFNN se muestran en la Tabla 3.18. Los datos en la primera columna corresponden a las redes generadas utilizando los valores de los centros y de los radios proporcionados por el algoritmo OVI. En las columnas segunda y tercera, los errores corresponden a las RBFNN generadas usando las posiciones de los centros son las proporcionadas por el algoritmo OVI y los radios inicializados mediante la heurística KNN y CIV. De este modo es posible comparar cómo el nuevo método propuesto para inicializar los radios mejora considerablemente los resultados obtenidos

p	**OVI**	**OVI-KNN**	**OVI-CIV**
1	0.044 (0.000)	0.167 (0.021)	0.107 (0.017)
1.5	0.041 (0.000)	0.131 (0.001)	0.153 (0.000)
2.5	0.105 (0.000)	0.151 (0.000)	0.151 (0.000)
4.5	0.102 (0.000)	0.155 (0.000)	0.231 (0.000)

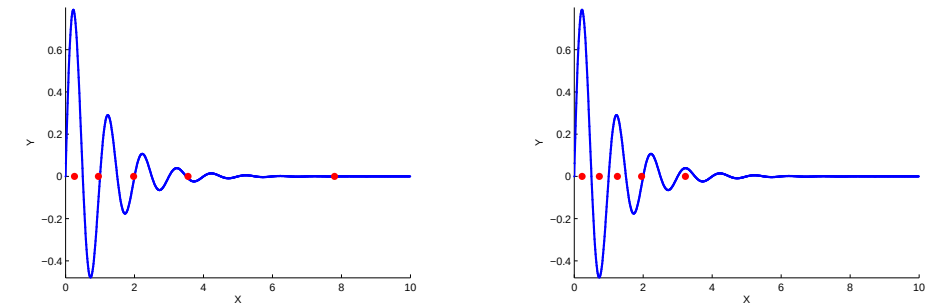
Tab. 3.18: Media y desviación estándar del error de aproximación (NRMSE) para la función f_1 usando el algoritmo OVI con varios valores para p y distintas heurísticas para el cálculo de los radios.

aplicando los métodos clásicos, independientemente del valor de p.

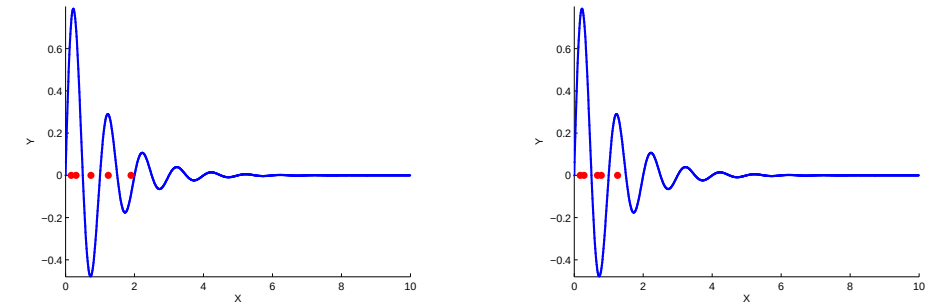
Tal y como se muestra en la Figura 3.20, cuanto más grande es el valor de p, más se aproximan los centros a las zonas más variables. Los resultados mostrados en las tablas indican que para valores grandes de p, los resultados empiezan a empeorar debido a que los centros se agrupan tanto en la zona más variable, que luego no es posible modelar con precisión el resto de zonas. Se ha determinado empíricamente que un valor adecuado para p debe estar comprendido entre 1 y 1.5 de modo que la salida de la función a aproximar influya razonablemente en el grado de activación de las neuronas pero sin provocar efectos perniciosos.

3.3.5.2. Experimento 2: Función unidimensional

La función f_1 empleada anteriormente ha sido utilizada en diversos artículos de la bibliografía para comparar distintas técnicas de inicialización de los parámetros que definen una RBFNN. En la Tabla 3.19 se muestran los errores de aproximación al diseñar las correspondientes redes utilizando el procedimiento clásico donde los radios han sido inicializados usando el algoritmo KNN con el parámetro K igual a 1 excepto para el algoritmo OVI que utiliza su propia metodología. El algoritmo OVI obtiene los menores errores



$p = 1$ $p = 1{,}5$



$p = 2{,}5$ $p = 4{,}5$

Fig. 3.20: Inicialización de los centros para varios valores de p.

m	HCM	FCM	ELBG	CFA	FCFA	OVI
6	0.759 (0.263)	0.783 (0.317)	0.152 (0.122)	0.090 (0.008)	0.103 (0.060)	0.041 (0.031)
7	0.344 (0.281)	0.248 (0.115)	0.111 (0.072)	0.081 (0.016)	0.069 (0.017)	0.029 (0.012)
8	0.227 (0.368)	0.150 (0.162)	0.093 (0.064)	0.053 (0.028)	0.048 (0.031)	0.014 (0.001)
9	0.252 (0.386)	0.300 (0.160)	0.073 (0.057)	0.056 (0.027)	0.027 (0.024)	0.010 (0.001)
10	0.087 (0.100)	0.285 (0.334)	0.064 (0.039)	0.047 (0.015)	0.011 (0.015)	0.007 (0.003)

Tab. 3.19: Error medio y desviación estándar del error de aproximación para la función f_1 utilizando distintos valores para el número de RBF (m).

de aproximación para cualquier número de centros pudiendo observarse cómo el error de aproximación para el menor número de centros es menor que los errores de aproximación de los otros algoritmos para el mayor número de centros.

3.3.5.3. Experimento 3: Función de dos dimensiones

A continuación se compara el algoritmo OVI con otros algoritmos de diseño de RBFNN así como de otros modelos para aproximación funcional. La función objetivo f_5 (Figura 3.21) fue utilizada por primera vez en (Cherkassky and H.Lari-Najafi, 1991) y se define como:

$$f_5(x_1, x_2) = 42{,}659(0{,}1 + x_1(0{,}05 + x_1^4 - 10x_1^2x_2^2 + 5x_2^4)) \quad x_1, x_2 \in [-0{,}5, 0{,}5] \tag{3.47}$$

En (Cherkassky et al., 1996) la aproximación de esta función se realizaba mediante el perceptrón multicapa y en (Pomares, 2000) se empleaba un sistema de reglas difusas. Otros modelos como los presentados en (Friedman, 1981),(Friedman, 1991) también usan esta función para medir la calidad de las aproximaciones. En (González et al., 2003) se utiliza un algoritmo genético multiobjetivo para diseñar una RBFNN que aproxime esta función, coincidiendo plenamente con el modelo que diseña el algoritmo OVI.

Todos los algoritmos utilizaron un conjunto de entrenamiento de 400 vectores de entrada generados aleatoriamente dentro de una malla de 20×20

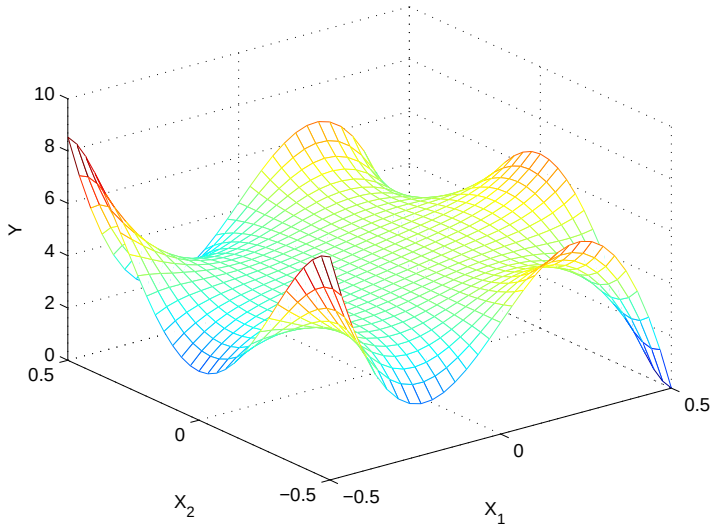


Fig. 3.21: Función objetivo f_5.

(Figura 3.22). El conjunto de test está formado por 961 datos generados aleatoriamente en una malla de 31×31. Los resultados proporcionados por los algoritmos citados previamente se muestran en la Tabla 3.20. El algoritmo OVI mejora notablemente las aproximaciones obtenidas por los otros algoritmos demostrando la efectividad de la inicialización tanto de los centros como de los radios.

3.3.5.4. Experimento 4: Predicción de series temporales

En esta ocasión el objetivo será realizar la predicción de una serie temporal presentada por Box y Jenkins en (Box and Jenkins, 1976b). Los datos se tomaron de un experimento real con el objetivo de predecir la concentración de CO_2 en un horno de gas teniendo en cuenta la concentración actual y el flujo de gas hacia el horno. Originariamente hay 296 pares de datos$y(t), u(t)$ con t=1...296, siendo el objetivo predecir el valor de$y(t)$ basándose en los valores

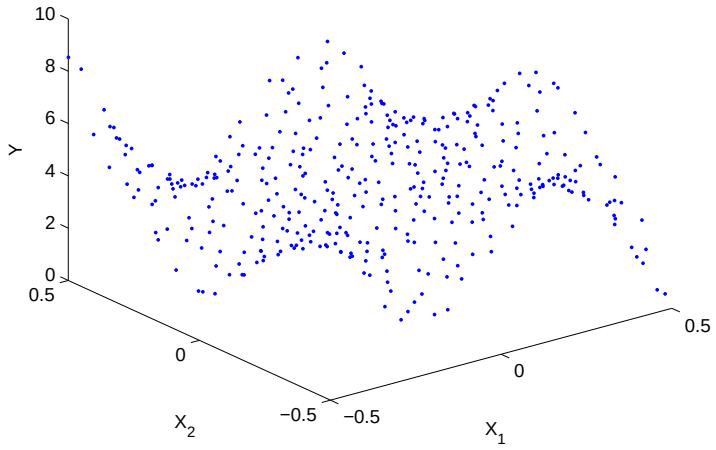


Fig. 3.22: Conjunto de entrenamiento para la función f_5.

Algorithm	m	Test NRMSE
MLP	15	0.308
PP	–	0.504
CTM	–	0.131
MARS	–	0.190
Cherkassky *et al*, 1996	40	0.038
González,2003	13	0.023 (0.001)
	14	0.021 (0.005)
	15	0.017 (0.005)
	16	0.016 (0.009)
	17	0.015 (8.5E-5)
Algoritmo Propuesto	13	0.004 (0.002)
	14	0.002 (0.001)
	15	0.002 (0.001)
	16	0.002 (0.001)
	17	0.001 (0.000)

Tab. 3.20: Error medio y desviación estándar del error de aproximación para la función f_5 utilizando distintos valores para el número de RBF (m).

$\{y(t-1), y(t-2), y(t-3), y(t-4), u(t-1), u(t-2), u(t-3), u(t-4), u(t-5), u(t-6)\}$ de modo que se poseen 10 variables de entrada y se disponen de 290 datos de entrada.

El algoritmo OVI es comparado con el algoritmo propuesto en (Leski, 2003) (Generalized Weighted Conditional Fuzzy *clustering*, GWCFC) el cual es una generalización del propuesto en (Pedrycz, 1998) (Conditional Fuzzy *clustering*, GWCFC), los cuales se basan en la definición de etiquetas lingüísticas para distintos contextos. Para el experimento, el algoritmo GWCFC diseña una RBFNN con seis neuronas utilizando los contextos*grande*,*mediano* y *pequeo*. En el trabajo presentado en (Leski, 2003), el error de predicción se mide usando el error cuadrático medio (*Root Mean Squared Error*, RMSE) definido como:

$$RMSE = \sqrt{\frac{\sum_{k=1}^{n} (y_k - \mathcal{F}(\vec{x}_k))^2}{n}} \tag{3.48}$$

En la Tabla 3.21 se muestran los errores de predicción obtenidos por los algoritmos GWCFC, FCM y OVI. El algoritmo OVI obtiene mejores resultados que los otros dos algoritmos demostrando nuevamente cómo las redes que puede diseñar tiene mejor capacidad de aproximación. En la Figura 3.23 se muestra la función a aproximar y el resultado que proporciona el algoritmo OVI.

3.4. Conclusiones

En el Capítulo 2 se presentó un algoritmo que fue diseñado específicamente para realizar una correcta inicialización de los centros de las RBF en problemas de aproximación funcional. En este capítulo se han descrito las carencias y deficiencias del algoritmo CFA y se han propuesto soluciones

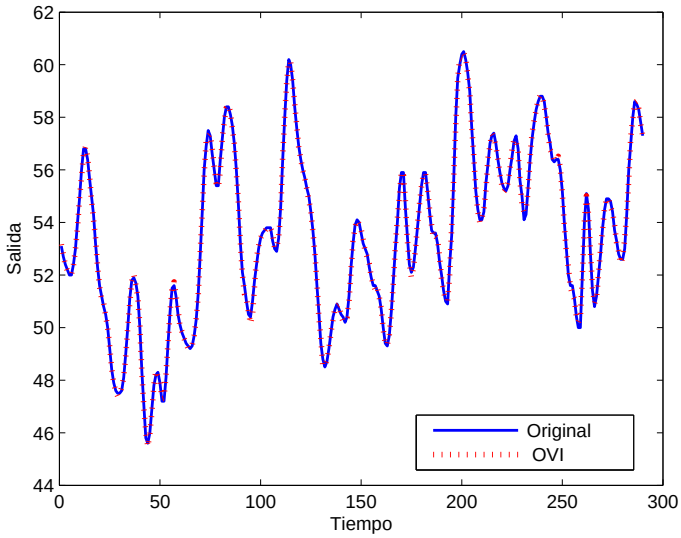


Fig. 3.23: Valores de salida para el conjunto de datos Box-Jenkins obtenidos usando el algoritmo OVI.

Algorithm	RMSE
GWFCM	0.187
FCM	0.204
OVI	0.1750

Tab. 3.21: Errores de aproximación (RMSE) para el conjunto de datos Box-Jenkins utilizando los algoritmos FCM, GWCM y OVI.

que, tal y como han mostrado los resultados, son efectivas. Este capítulo ha puesto de relieve lo importante que es partir de una buena localización antes de realizar la minimización del error mediante un algoritmo de búsqueda local tal y como se propone en una de las metodologías de diseño de RBF. En este sentido, el algoritmo ICFA representa un gran avance puesto que realiza una inicialización adecuada de los centros y, además, lo hace en un tiempo bastante reducido en comparación con su antecesor. Este factor es importante puesto que el posterior algoritmo de búsqueda local suele requerir mucho tiempo de cómputo.

Uno de los cambios más significativos del ICFA con respecto al CFA es el modo en el que la partición del conjunto de datos de entrada se realiza. En el ICFA se ha propuesto el uso de una partición difusa. Dado que uno de los elementos más importantes en un algoritmo de *clustering* es el tipo de partición de los datos que realiza, se ha hecho un estudio en profundidad de cómo el algoritmo se comporta cuando utiliza los distintos tipos de particiones propuestas en la literatura. Los resultados han mostrado cómo el enfoque difuso tiene un comportamiento excepcional si bien, en ciertas circunstancias, la hibridación con el enfoque posibilístico, puede mejorar los resultados a costa de un mayor tiempo de ejecución y la necesidad de inicializar más parámetros.

Tras haber estudiado la inicialización de los centros, se ha observado como también es necesario asignar un valor adecuado a los radios de cada RBF. Con el objetivo de mantener la interpretabilidad durante la inicialización de los centros, se ha propuesto un nuevo algoritmo que es capaz de situar los centros en zonas adecuadas y además permite dar un valor inicial adecuado también para los radios. Tal y como se ha mostrado en el capítulo, este nuevo algoritmo proporciona unos resultados bastante satisfactorios.

4. ALGORITMO GENÉTICO MULTIOBJETIVO PARALELO PARA EL DISEÑO DE RBFNN Y SELECCIÓN DE VARIABLES

Este capítulo presenta los componentes de un algoritmo evolutivo multiobjetivo distribuido heterogéneo para la optimización de los parámetros de una red de funciones base radiales, incluyendo en éstos la selección de las variables que son consideradas por la red.

El diseño del algoritmo ha requerido la adaptación, hibridación e integración de diversas técnicas y paradigmas de programación tales como, algoritmos de búsqueda local, algoritmos evolutivos multiobjetivo, algoritmos evolutivos paralelos y heterogéneos y redes neuronales de funciones de base radial.

La razón para integrar las diversas técnicas es el poder combinar las ventajas que posee cada una, supliendo sus desventajas con las demás. Por ejemplo, dado que la búsqueda local puede dejar de explorar zonas del espacio solución

debido a su carácter local, los algoritmos evolutivos permitirán realizar esa tarea, supliendo la posible falta de precisión en el ajuste de las soluciones de éstos mediante la búsqueda local. El aplicar la evolución multiobjetivo es lo que permite al algoritmo mantener un compromiso entre la complejidad de las redes y la calidad de las soluciones a costa de un coste computacional mayor. Este último punto es lo que motiva a introducir el paralelismo dentro del algoritmo dado que los algoritmos evolutivos son fácilmente paralelizables y esto permite reducir el coste computacional. Gracias a la sinergia de estas técnicas, el paralelismo no sólo se puede orientar a la reducción en el tiempo de computación sino en la distribución de tareas entre procesos, dando lugar a un algoritmo distribuido heterogéneo que, gracias a la especialización de sus tareas, permite mejorar los resultados proporcionados por un algoritmo secuencial homogéneo.

Dentro del presente capítulo se detalla paso a paso, con un diseño incremental, cómo se han ido integrando las diversas técnicas así como las modificaciones y adaptaciones necesarias para resolver el problema en cuestión del diseño de RBFNN.

4.1. Evolución Multiobjetivo de RBFNN

En el Capítulo 3 se han presentado una serie de algoritmos que proporcionan puntos de partida aceptables para, a partir de ahí, ajustar los parámetros de la red de modo que el error de aproximación sea lo más pequeño posible. Aunque el comportamiento de los algoritmos es bastante adecuado, como mostraron los resultados experimentales, se basan en técnicas de descenso en gradiente que no garantizan la convergencia a un mínimo global .

Tal y como se mostró en el Capítulo 2, para suplir los procedimientos de búsqueda local, se han adaptado algoritmos genéticos (*Genetic Algorithms*, GA) (Bounds, 1987); (Buckles and Petry, 1993); (Davis, 1991); (Forrest, 1993a); (Goldberg, 1994); (Rojas et al., 2001); (Schwefel, 1981); (Srinivas and Patnaik, 1994b); (Stender, 1893) que evolucionan RBFNN e intentan explorar de forma global el espacio solución. Esta exploración también considera la optimización del número de neuronas que debe tener la red. Dado que se desea diseñar una RBFNN que proporcione un error de aproximación lo más bajo posible, éste tenderá a disminuir conforme aumente el número de neuronas, siendo el caso extremo el que asigne una neurona por vector de entrada de modo que ésta sólo se active para ese vector. De este modo se obtendría un error de aproximación igual a 0. Esta situación no es adecuada debido a que aunque el conjunto de entrenamiento se aprenda a la perfección, la capacidad de generalización de las RBFNN se perdería al sobreajustarse la red, incluyendo efectos colaterales como el aprendizaje del posible ruido en el conjunto de entrenamiento. Para tener en cuenta este hecho, se ha considerado la evolución multiobjetivo de las redes de modo que se pueda mantener un compromiso entre el error de aproximación y el tamaño de la red como ya se hiciera en (González et al., 2003); (García-Pedrajas et al., 2002); (Hatanaka et al., 2003); (Valdés and Barton, 2006); (Valdés and Barton, 2000).

Uno de los aspectos que ha influido notablemente en el desarrollo de los GA es la aparición de problemas multiobjetivo y su posible solución mediante modificaciones de los GA clásicos. Un problema multiobjetivo es aquel en el que existen soluciones que no se pueden considerar ni mejores ni peores que otras ya que éstas encuentran un compromiso distinto pero igual de válido entre los objetivos a satisfacer. Formalmente, la solución a un problema multiobjetivo no consiste en un único valor sino en un vector $f : D \rightarrow \mathbb{R}^{N_{obj}}$ dentro del espacio de soluciones D con N_{obj} funciones a optimizar.

Dentro de los algoritmos evolutivos multiobjetivo, se propuso en (Deb et al., 2002) el algoritmo denominado *Non-dominated Sorting Genetic Algorithm II* (NSGAII), que es considerado como uno de los mejores algoritmos de este tipo. Debido a esta circunstancia se ha elegido como algoritmo de base a partir del cual se adaptarán los distintos elementos que componen el nuevo algoritmo propuesto para diseñar RBFNN. Otros factores que apoyan esta decisión son:

- su política de reemplazo de individuos considera el elitismo de un modo que se puede adaptar fácilmente a la hora de integrar individuos que migran desde otras islas, tal y como se mostrará en el siguiente capítulo.

- su eficiciencia a la hora de generar la siguiente población

- su sencillez y claridad a nivel de diseño.

A continuación se mostrará en qué consiste el funcionamiento del algoritmo NSGAII y en la sección 5.1 se describirán los elementos que constituyen el algoritmo propuesto para el diseño de RBFNN.

4.1.1. Non-dominated Sorting Genetic Algorithm II (NSGAII)

Este algoritmo es una versión evolucionada de el algoritmo*Non-dominated Sorting Genetic Algorithm* (NSGA) propuesto en (Deb and Goel, 2001). El aspecto que mejora la segunda versión es el procedimiento mediante el cual se van obteniendo poblaciones de soluciones no dominadas, haciéndolo más eficiente. Para poder comprender esto, se introduce a continuación el concepto de dominancia y Pareto. En los problemas de optimización multiobjetivo existen soluciones $\iota_j \in D/f(\iota_j) = \Big(f_1(\iota_j), f_2(\iota_j), ..., f_{N_{obj}}(\iota_j)\Big)$ que no pueden ser ordenadas utilizando las relaciones clásicas que se establecen en los problemas con un único objetivo f, las cuales son:

$$f(\iota_1) \leq f(\iota_2) \qquad \text{ó} \qquad f(\iota_2) \leq f(\iota_1). \tag{4.1}$$

Al ser cada solución un vector, las anteriores comparaciones se establecen así:

$$\begin{aligned} f(\iota_1) = f(\iota_2) &\iff f_i(\iota_1) = f_i(\iota_2) \qquad \forall i \in 1, 2, ..., N_{obj} \\ f(\iota_1) \leq f(\iota_2) &\iff f_i(\iota_1) \leq f_i(\iota_2) \qquad \forall i \in 1, 2, ..., N_{obj} \\ f(\iota_1) < f(\iota_2) &\iff (f(\iota_1) \leq f(\iota_2)) \wedge (f(\iota_1) \neq f(\iota_2)) \end{aligned} \tag{4.2}$$

siendo imposible establecer un orden estricto entre las soluciones ya que, por ejemplo, puede darse el caso de que $f_1(\iota_2) < f_1(\iota_3)$ y $f_2(\iota_2) < f_2(\iota_3)$. Debido a esto, se obtienen conjuntos de soluciones parcialmente ordenados (Pareto, 1896) donde ninguna solución es mejor que otra. Por tanto, podemos definir la relación de dominancia entre soluciones como:

$$\iota_1 \prec \iota_2 \qquad (\iota_1 \text{ domina a } \iota_2) \qquad \iff \qquad f(\iota_1) < f(\iota_2)$$

$$\iota_1 \preceq \iota_2 \quad (\iota_1 \text{ domina débilmente a } \iota_2) \iff f(\iota_1) \leq f(\iota_2) \tag{4.3}$$
$$\iota_1 \sim \iota_2 \quad (\iota_1 \text{ es indiferente a } \iota_2) \iff f(\iota_1) \not\leq f(\iota_2) \wedge f(\iota_2) \not\leq f(\iota_1)$$

Utilizando esta relación de dominancia se puede, dado un conjunto de soluciones, obtener un subconjunto de éstas donde todas las soluciones no sean dominadas por ninguna otra. Este subconjunto se denomina Pareto y el Pareto óptimo es el Pareto del conjunto de todas las soluciones posibles del espacio solución, es decir, el conjunto de soluciones que no pueden ser dominadas por ninguna otra (Hans, 1988); (Chankong and Haimes, 1983); (Hwang and Masud, 1979). Así podemos afirmar que la solución ι_j es Pareto óptima (pertenece al Pareto óptimo) si y sólo si:

$$\nexists \iota_i \in D : \iota_i \prec \iota_j \tag{4.4}$$

donde D es el conjunto de todas las soluciones posibles y $\prec$ es el criterio que determina si una solución domina a otra.

El funcionamiento a grandes rasgos del NSGAII consiste en, partiendo desde el punto de vista de un GA estacionario, ya que tras generar los descendientes sólo algunos de ellos entran a formar parte de la población, se establece una estrategia de reemplazo basándose en la obtención de Paretos no dominados sucesivamente. Más concretamente, se realiza el cruce de los individuos y se unen las dos poblaciones: ancestros y descendientes. A partir de este conjunto de individuos, se van obteniendo Paretos que pasarán a formar la población para la siguiente generación, procediendo de este modo hasta que el tamaño de la población se haya completado. Este proceso muestra gráficamente en la Figura 4.1.

Otra característica de este algoritmo es la asignación un un ranking para cada individuo de modo que, de entre el último Pareto que completa la

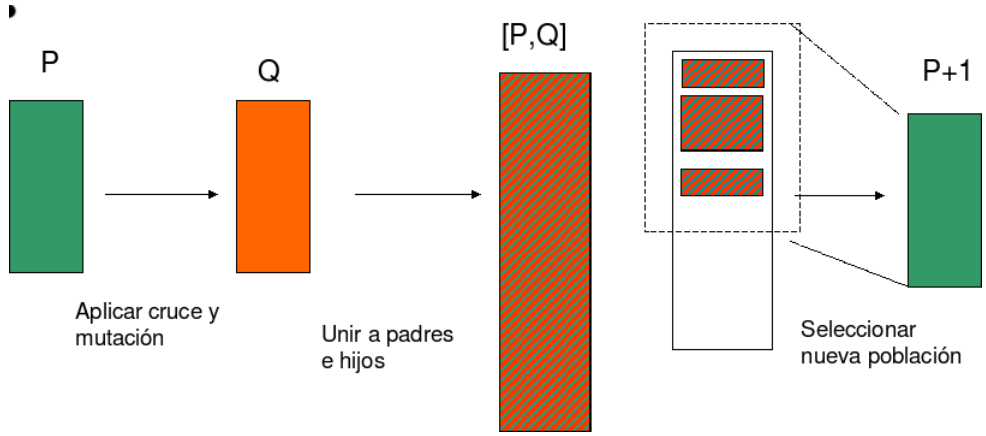


Fig. 4.1: Esquema de ejecución del algoritmo NSGAII. P representa a la población de la generación actual y Q los descendientes generados tras aplicar el cruce y mutación.

población, sólo se seleccionan los que están más dispersos en la frontera del Pareto.

4.2. Algoritmo Memético Paralelo para el Diseño de RBFNN

Tras haber motivado el diseño de un algoritmo de optimización global y ver cómo el planteamiento del problema es adecuado para tratarlo como multiobjetivo, de forma que se reduzca tanto la complejidad de la red como su error de aproximación, esta sección presenta todos los elementos del algoritmo propuesto en esta memoria.

4.2.1. Estructura general

Antes de concretar cada uno de los elementos del algoritmo, se describe el esquema general del algoritmo, puesto que no es un algoritmo evolutivo común. La peculiaridad que lo distingue de los algoritmos genéticos clásicos es la hibridación con técnicas de optimización local, haciendo que entre dentro

de la categoría de algoritmo memético.

Los algoritmos meméticos fueron presentados en (Moscato, 1992) y combinan las cualidades de los algoritmos genéticos como optimizadores globales y las de las técnicas de búsqueda local que permiten obtener una mayor precisión de los resultados. Los algoritmos meméticos están cobrando interés recientemente dada su superioridad en comparación con las dos técnicas utilizadas por separado (Ishibuchi and Kaige, 2004); (Murata et al., 2003); (Martínez-Estudillo et al., 2005); (Mártínez et al., 2005); (Wang et al., 1998); (Radtke et al., 2005); (Ong et al., 2006).

Las dos últimas tendencias con respecto al diseño de algoritmos meméticos han sido propuestas en (Gandibleux et al., 2001) y en (Deb and Goel, 2001) donde hay dos etapas claramente diferenciadas: la búsqueda global mediante el algoritmo genético y la búsqueda local. En (Gandibleux et al., 2001) se aplica en primer lugar una búsqueda local para inicializar los individuos y pasar a la ejecución del algoritmo de búsqueda global. Por el contrario, en (Deb and Goel, 2001), se ejecuta un algoritmo multiobjetivo primero y, tras su ejecución, se aplica una búsqueda local sobre los individuos.

El algoritmo desarrollado integra ambas técnicas, realizando una inicialización de los individuos mediante técnicas de búsqueda locales para continuar con una evolución multiobjetivo incluyendo un operador de búsqueda local y, por último, optimizando cada uno de los individuos en el Pareto frontera mediante otra búsqueda local.

4.2.2. Representación de los individuos

El algoritmo considera una población de individuos donde cada uno representa una RBFNN. Para que un individuo codifique completamente una RBFNN se deben almacenar los siguientes elementos:

1. El conjunto de variables que alimentan las unidades de procesamiento

2. Las posiciones de los centros de las RBF.

3. Los radios de cada RBF.

Para poder codificar el conjunto de variables que sirven de entrada a la red se ha optado por la codificación binaria, donde se almacena un vector con tantos elementos como variables y se asigna un 1 o un 0 en función de si la variable se considera o no (Siedlecki and Sklansky, 1989). Existen otras codificaciones que han sido estudiadas para resolver este problema (Punch et al., 1993); (Kelly and Davis, 1991); (Raymer et al., 1997) con el objetivo de ponderar la influencia de ciertas variables, sin embargo, debido a la simpleza, eficiencia y espacio de soluciones finito, se ha optado por la codificación binaria.

El resto del cromosoma se encarga de albergar tanto los valores de los radios como las coordenadas de los centros de las RBF. Para ello se utiliza la codificación real donde cada valor se almacena mediante un número representado en coma flotante.

Los pesos de la red no son almacenados dentro del cromosoma por las siguientes razones:

- Los pesos no necesitan ser evolucionados dado que pueden ser calculados de forma óptima dado un conjunto de centros y radios.

- Cada modificación en un centro o en un radio conlleva el recálculo de los pesos de modo que el valor de los pesos almacenados no sería usado.

- Como el algoritmo, según se mostrará en las subsecciones siguientes, se paraleliza, necesita el intercambio periódico de individuos. Como la

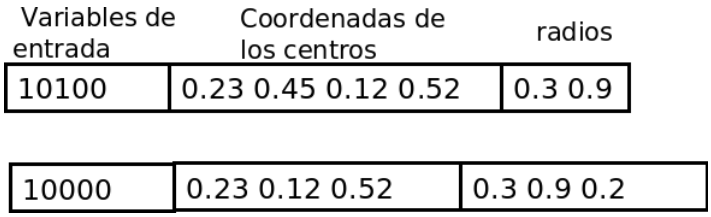


Fig. 4.2: Codificación de una RBFNN en un individuo

eficiencia en el envío depende de la cantidad de datos para ser enviados, el enviar los pesos causaría una perdida innecesaria de tiempo.

4.2.3. Inicialización de la primera población

Aunque teóricamente los algoritmos evolutivos son capaces de encontrar una solución para el problema desde una población inicial generada aleatoriamente, es conveniente partir de una población cuyos individuos estén bien situados en el espacio de búsqueda. Como se muestra en (Reeves, 1993), si la población está inicializada adecuadamente, se convergerá a una buena solución en menos tiempo.

Como tanto los parámetros que definen una RBFNN como las variables de entrada de ésta son optimizados por el algoritmo, es necesaria una correcta inicialización de ambos elementos. Para el problema de la identificación de las variables que determinan la entrada de la red se ha empleado el concepto de información mutua para determinar las variables que proporcionan más información sobre la salida. Para dar un primer valor a los centros y a los radios se han empleado las técnicas descritas en el capítulo anterior.

4.2.3.1. Teoría de la Información Mutua

Dado que los individuos codifican el conjunto de variables de entrada usando un vector de ceros y unos, el número de posibles combinaciones es

de 2^d. Este número puede no parecer muy grande a priori pero se ha de considerar que para cada combinación existen multiples posibilidades de diseño de RBFNN. Por tanto, es lógico partir de soluciones que sean buenas pero sin descartar todas las posibilidades. Para poder definir un conjunto de variables adecuado se utilizará el concepto de Información Mutua (*Mutual Information*, MI), también conocido como entropía cruzada. Sea $X = \{\vec{x}_k\}$ e $Y = \{y_k\}$ para $k = 1...n$ entonces la MI entre X e Y puede ser definida como la cantidad de información que el grupo de variables X proporcionan sobre la variable de salida Y y se expresa como:

$$I(X, Y) = H(Y) - H(Y|X). \tag{4.5}$$

donde $H(Y)$ es la entropía de la variable Y, la cual mide la incertidumbre sobre Y. En el caso de tener variables continuas y, siguiendo la formulación de Shannon, puede definirse como:

$$H(Y) = -\int \mu_Y(y) \log \mu_Y(y) dy, \tag{4.6}$$

donde $\mu_Y(y)$ es la función de densidad marginal. Esta función puede definirse a su vez como la unión de la función de densidad de probabilidad $\mu_{X,Y}$ de X con Y, es decir:

$$\mu_Y(y) = \int \mu_{X,Y}(x, y) dx. \tag{4.7}$$

La entropía condicional $H(Y|X)$, que mide la incertidumbre de Y dado un X conocido, se define en el caso continuo como:

$$-\int \mu_X(x) \int \mu_Y(y|X = x) \log \mu_Y(y|X = x) dy dx. \tag{4.8}$$

La conclusión que se obtiene es que la información mutua $I(X, Y)$ es el grado de disminución de la incertidumbre sobre Y una vez que X es conocido.

Gracias a las propiedades de la entropía y la información mutua, ésta última puede definirse también como:

$$I(X,Y) = H(X) + H(Y) - H(X|Y), \tag{4.9}$$

lo cual nos lleva a:

$$I(X,Y) = \int \mu_{X,Y}(x,y) \log \frac{\mu_{X,Y}(x,y)}{\mu_X(x)\mu_Y(y)} dxdy. \tag{4.10}$$

Esto nos indica que para calcular la información mutua entre dos grupos de variables pertenecientes a X, solo es necesario estimar la función de densidad de probabilidad conjunta entre X y Y. Aunque para realizar esta estimación se pueden emplear diversas técnicas como se describe en (B.V. Bonnlander and A.S. Weigend, 2004), la que se utiliza para la inicialización de los individuos se basa en el algoritmo de los K vecinos más cercanos (*K-Nearest Neighbors*, KNN) debido a que posee ciertas ventajas sobre los otros métodos basados en histogramas o en *kernels* (Kraskov et al., 2004).

Se ha utilizado la implementación disponible en (Milca) para calcular la información mutua. Esta implementación requiere la inicialización de un parámetro que define el número de vecinos más cercanos que se ha de considerar en la estimación. Tal y como se recomienda en (Harald et al., 2004) se ha fijado este valor a 6.

Una vez que la información mutua $MI_i, i = 1...d$ para cada variable $\vec{x} = \{x_1, x_2, ..., x_d\}$ se ha calculado, se normalizan sus valores entre 0 y 1 y el valor que toma el vector que codifica las variables de cada red se calcula mediante:

$$round(\vec{MI}_{norm} \cdot \vec{rand}) \tag{4.11}$$

donde *round* es una función que redondea un valor real al entero más cercano y $\vec{rand} \in \mathbb{R}^d$ es un vector con valores aleatorios entre 0 y 1 obtenidos

de una distribución uniforme. Este método de inicialización permite que las variables con un valor alto de información mutua tengan más probabilidades de ser seleccionadas que las otras aunque no se cierra la posibilidad de que variables con valores menores de información mutua sean seleccionadas.

Para satisfacer la posibilidad de que todas las variables sean seleccionadas, dado que es una posibilidad válida, el primer individuo de la población tiene todas las entradas del vector igual a 1.

4.2.3.2. Inicialización de Centros y Radios

Cada RBFNN debe tener un valor inicial asignado a sus radios y centros para poder empezar a evolucionarlos. Como punto de partida se van a generar un conjunto de individuos usando las técnicas de inicialización presentadas en el capítulo anterior (ICFA, FPCFA, PFCFA y OVI) usando valores aleatorios para cada uno de los parámetros que utilizan estos algoritmos. Gracias a esta selección aleatoria de parámetros para los algoritmos obtenemos una mayor diversidad en la población. En cuanto a la inicialización de los radios, se emplea la heurística de los K vecinos salvo cuando el algoritmo usado sea el OVI, que utilizará su propio método. El valor de K para cada individuo también se generará aleatoriamente. Dado que los algoritmos de inicialización de los centros son robustos, también se incluyen en la población individuos generados de forma totalmente aleatoria para mantener diversidad en la población.

Tras generar la mitad de la población procediendo como se ha indicado en el párrafo superior, el resto de la población se obtiene como resultado de aplicar varias iteraciones del algoritmo $Levenberg - Marquardt$. Gracias a este paso, los individuos ajustan adecuadamente los valores de sus correspondientes parámetros y se generan nuevos individuos que aumentan la diversidad de la población así como la calidad de los resultados finales.

En lo referente al número de RBF que cada red posee, este valor también se genera aleatoriamente aunque procurando que no sea grande ya que de lo contrario:

- el algoritmo tendría que trabajar con redes de gran tamaño desde las primeras iteraciones haciendo un esfuerzo quizá innecesario puesto que a priori se desconoce cómo de compleja deberá ser la red

- como se mostrará a continuación, un operador de cruce y varios de mutación permiten aumentar el número de RBF de la red siendo posible hacer una evolución incremental de las redes.

4.2.4. Operadores de cruce

Como se mostró en el Capítulo 2 (sección 2.3.2.1), el algoritmo genético propuesto con anterioridad posee un único operador de cruce que permite transmitir de generación en generación el material genético de los progenitores. Sin embargo, el diseño de una RBFNN y la selección de sus variables de entrada es una tarea ardua y compleja que requiere la evolución de muchos parámetros, haciendo que un único operador de cruce no se ajuste a la evolución de esos parámetros independientemente. En esta sección se presentan 3 operadores de cruce mediante los cuales es posible explotar el material genético de los antecesores en cada uno de los aspectos del problema, obteniendo descendientes que combinan la información de los padres pero que no sufren transformaciones drásticas como podría ocurrir si se aplicara un único operador de cruce que considerase todos los elementos a evolucionar.

4.2.4.1. Operador de cruce 1: Adición de la neurona con el menor error local

El operador selecciona una neurona de cada individuo y la añade al otro individuo de modo que cada descendiente tiene una neurona más que su progenitor. Este operador de cruce realiza un intercambio de información entre los individuos aprovechando el conocimiento que se dispone del problema a resolver. Concretamente, la elección de la neurona que se transmitirá al otro individuo se hace en base al error de aproximación que la red obtiene. Otro aspecto importante de este operador es que permite a las redes crecer en tamaño, pudiendo explorar nuevas topologías.

Para realizar la selección de la neurona que será añadida a la otra red, se ha de calcular el error local de cada neurona. El error local de una neurona consiste en la suma de los errores cometidos en los puntos de entrada que activan esa neurona (por encima de un umbral) cuando se calcula la salida de la RBFNN. Por ejemplo, en la Figura 4.3 se muestra una función objetivo para ser aproximada mediante una RBFNN. Se ha generado una RBFNN inicializando los centros y los radios de manera aleatoria puesto que en este ejemplo, la calidad de la aproximación no es interesante, proporcionando la aproximación que se muestra en la Figura 4.4. Esta RBFNN esta compuesta por dos neuronas cuyos valores de la matriz de activación están representados en la Figura 4.5, las cuales, superpuestas con las dos figuras anteriores, se muestran en la Figura 4.6. Para decidir cuál de las dos neuronas sería la elegida para ser añadida a la otra red con la que se desea cruzar, se ha de calcular el error entre la salida real y la salida generada por la red, tras esto, para cada punto que active a la neurona de una forma considerable (por encima de un valor de 0.5) se acumulará el error que hay en la aproximación. En la Figura 4.7 están representadas la matriz de activación de la red y el

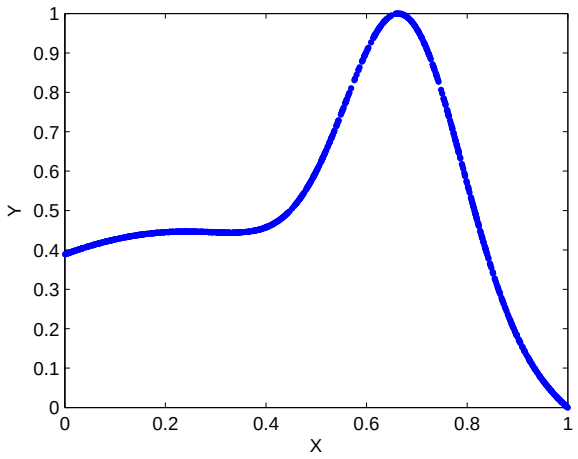


Fig. 4.3: Función objetivo para ser aproximada

error (en valor absoluto) de la aproximación con respecto a la salida real. Como se puede apreciar en la función, la neurona de la derecha acumula más error en la zona donde se activa, por tanto, se seleccionaría la otra neurona.

Al añadir una neurona perteneciente a otra red puede ocurrir el caso de que la red de destino tenga distintas variables de entrada. Para que la nueva neurona pueda formar parte de la red, debe poseer las mismas variables de entrada así que estas situaciones se resuelven siguiendo el criterio que se muestra a continuación para cada variable:

- si la neurona posee una coordenada correspondiente a esta variable, su valor se conserva
- si la neurona no la posee, se le asigna un valor aleatorio dentro del espacio de los vectores de entrada.

El hecho de asignar un valor aleatorio permite obtener eficiencia, diversidad y exploración en la coordenada correspondiente. Podrían utilizarse heurísticas basadas en clustering de modo que la nueva coordenada se situara

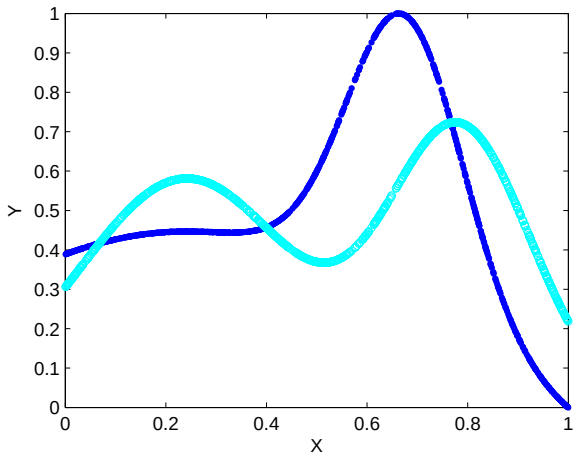


Fig. 4.4: Función objetivo para ser aproximada y aproximación obtenida mediante una RBFNN.

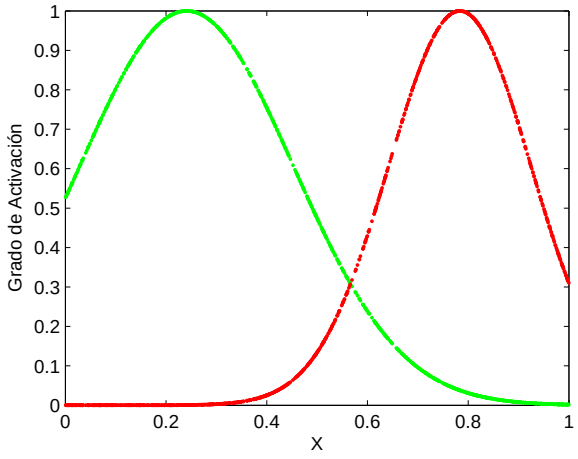


Fig. 4.5: Valores de la función de activación en cada punto de entrada para las dos neuronas que componen la RBFNN

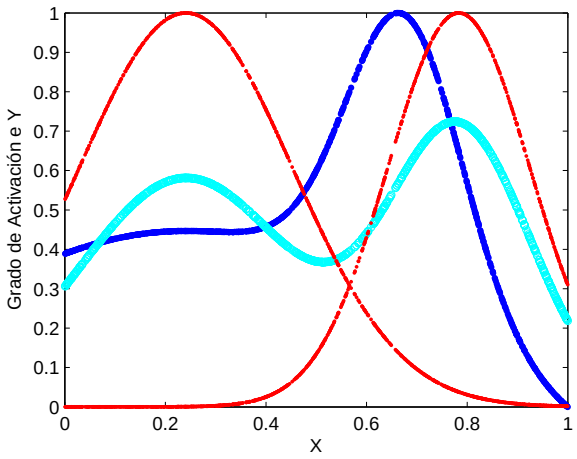


Fig. 4.6: Función objetivo para ser aproximada, aproximación mediante una RBFNN y valores de activación para las dos neuronas

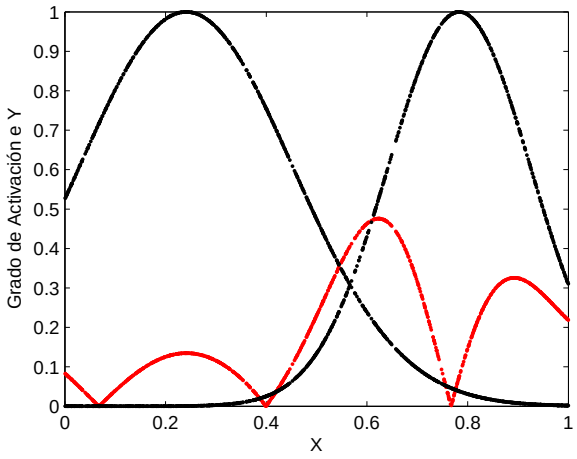


Fig. 4.7: Diferencia en valor absoluto entre la función objetivo y la aproximación obtenida por la RBFNN y los valores de activación para cada punto de entrada

en una zona no cubierta en esa dimensión por las otras neuronas, pero:

1. la eficiencia se vería disminuida considerablemente y ésta es un factor importante puesto que el operador de cruce requiere un coste computacional elevado al tener que recalcular la matriz de activación.

2. el razonamiento en el que se basa esa heurística es fácilmente rebatible puesto que si una zona del espacio solución no está cubierta, ha podido ser la evolución la que lo determine y, el situar ahí una neurona puede entorpecer el proceso evolutivo.

3. muy relacionado con el punto anterior, la aplicación de la heurística del *clustering* o la asignación de un área no cubierta por la red puede llevar a la perdida de diversidad y exploración en comparación con la asignación de un valor aleatorio.

Se han propuesto otros métodos para determinar qué neurona es la más importante dentro de la red, siendo el método de los mínimos cuadrados ortogonales (*Orthogonal Least Squares*, OLS) el que mejores resultados proporciona (González et al., 2003); (Barreto et al., 2006). Para poder decidir qué criterio utilizar para determinar qué neurona se selecciona, se ha hecho un estudio del comportamiento del operador de cruce utilizando ambos criterios: la neurona con menor error local y la que tenga un coeficiente de disminución del error más alto según el método OLS.

El estudio se ha hecho basándose en el Principio de Optimalidad de Bellman (Bellman, 1966), el cual establece que, dada una secuencia de pasos para resolver un problema de forma óptima, si se parte de una situación cualquiera y se toman decisiones óptimas, se alcanzará un valor óptimo para la solución de problema. Dado que, aproximadamente, se puede definir un

algoritmo genético como una secuencia recursiva de pasos, donde si en cada uno se toma una decisión óptima, se llega a una solución óptima, para evaluar los operadores de cruce se ha partido de un poblaciones generadas aleatoriamente y que se pueden suponer óptimas tras la ejecución de pasos con decisiones óptimas. Estas poblaciones se han generado teniendo como objetivo a aproximar 10 funciones de base radial de los tipos:

$$\sin(\log(\frac{||\vec{x}_k - \vec{c}_j||^2}{r_j^2})) \tag{4.12}$$

$$e^{-\frac{||\vec{x}_k - \vec{c}_j||^2}{r_j^2}} \tag{4.13}$$

y para cada función se han creado 40 individuos aleatorios agrupado en 20 pares. Tras cruzar los 20 pares de individuos utilizando ambos criterios (OLS y menor error local), se obtuvieron 40 descendientes para cada criterio. Al realizar el test de Kruskal-Wallis para el análisis de la varianza (ANOVA) del error de aproximación proporcionado por los descendientes, no se pudo apreciar una diferencia entre ambos criterios, proporcionando resultados muy similares. Partiendo de la suposición de que los individuos generados pertenecen a la última generación, se les aplicó una búsqueda local para obtener las soluciones finales y se realizo el ANOVA del error de aproximación tras la búsqueda local, el cual indicó que no había diferencias significativas entre uno y otro. Sin embargo, al medir el tiempo de ejecución entre ambos y realizar el ANOVA sí se encontraron diferencias indicando que el OLS es más lento que el criterio del error local. Este dato es el que sustenta la elección de este criterio puesto que la eficiencia en este operador de cruce puede ser crítica a la hora de evitar retenciones en la comunicación entre procesos, como se

mostrará adelante, ya que los otros operadores son muy veloces.

4.2.4.2. Operador de cruce 2: Intercambio de neuronas hibridado con el cruce BLX-α

Este operador, originalmente propuesto en (González et al., 2003), realiza un intercambio de neuronas entre dos individuos tal y como se muestra en la Figura 4.10. El operador original, selecciona dos neuronas aleatoriamente de cada uno de los individuos y las intercambia, con la única restricción de que los vectores de entrada que las activan no se solapen demasiado. El operador propuesto en esta memoria ha modificado esta restricción y considera únicamente que la neurona elegida para ser intercambiada no esté ya incluida en la red de destino. La razón de no considerar las zonas de activación está motivada por la posibilidad de que, aún activándose por vectores de entrada parecidos o cercanos, si el peso de una de las neuronas es positivo y el otro negativo, es posible que la función se modele correctamente tal y como se muestra en la Figura 4.11. Si se quitara alguna de las dos neuronas que se solapan, se obtendrían los resultados mostrados en las Figuras 4.12 y 4.13 donde se puede apreciar cómo la aproximación es más errónea que en el caso en el que se mantengan las dos neuronas solapadas. En el peor de los casos, cuando dos neuronas se solapen demasiado y tengan sus parámetros muy similares, dado que se realiza una evolución multiobjetivo, esta red tenderá a desaparecer, ya que poseerá un número de neuronas mayor que otras redes que proporcionarán un error similar con menor número de neuronas.

Como consecuencia de la representación del problema, dentro de la misma población coexistirán redes con distintos conjuntos de variables de entrada.

	SS	df	MS	Chi-sq	Prob>Chi-sq
f1					
Columnas	1773.8	1	1773.76	2.98	0.0842
Error	47611.2	82	580.62		
Total	49385	83			
f2					
Columnas	555.4	1	555.429	0.93	0.3341
Error	48829.6	82	595.483		
Total	49385	83			
f3					
Columnas	1612.2	1	1612.19	2.71	0.0997
Error	47772.8	82	582.6		
Total	49385	83			
f4					
Columnas	618.9	1	618.857	1.04	0.3078
Error	48766.1	82	594.709		
Total	49385	83			
f5					
Columnas	1525.8	1	1525.76	2.56	0.1093
Error	47859.2	82	583.65		
Total	49385	83			
f6					
Columnas	1320.1	1	1320.11	2.22	0.1363
Error	48064.4	82	586.15		
Total	49384.5	83			
f7					
Columnas	1475.05	1	1475.05	2.48	0.1154
Error	47909.95	82	584.27		
Total	49385	83			
f8					
Columnas	744.05	1	744.048	1.25	0.2635
Error	48640.95	82	593.182		
Total	49385	83			
f9					
Columnas	1057.2	1	1057.19	1.78	0.1825
Error	48327.8	82	589.36		
Total	49385	83			
f10					
Columnas	2100	1	2100.01	3.53	0.0603
Error	47285	82	576.646		
Total	49385	83			

Tab. 4.1: Tablas resultado del test de Kruskal-Wallis sobre el error de aproximación tras la aplicación de la búsqueda local usando los operadores OLS y Error local.

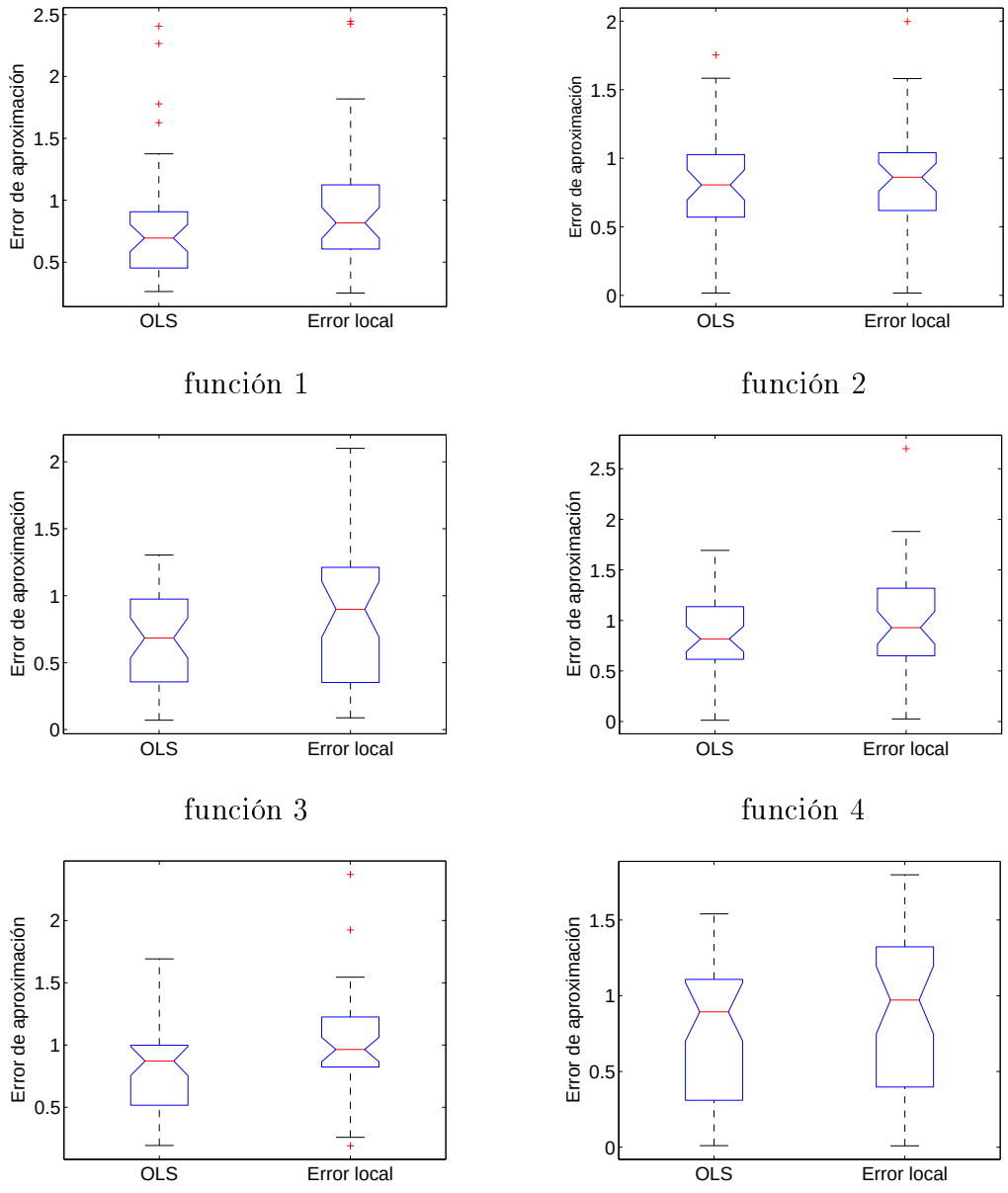


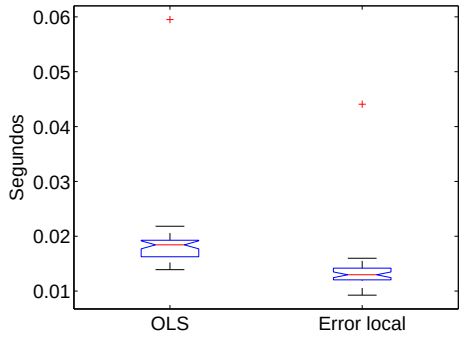
Fig. 4.8: Resultados del test de Kruskal-Wallis sobre el error de aproximación para comparar los dos operadores métodos de identificación de la neurona más relevante de las 6 primeras funciones.

	SS	df	MS	Chi-sq	Prob>Chi-sq
f1					
Columnas	220.2	1	220.19	0.37	0.543
Error	49164.8	82	599.571		
Total	49385	83			
f2					
Columnas	138.9	1	138.857	0.23	0.629
Error	49246.1	82	600.563		
Total	49385	83			
f3					
Columnas	45.8	1	45.762	0.08	0.7815
Error	49339.2	82	601.698		
Total	49385	83			
f4					
Columnas	149.3	1	149.333	0.25	0.6164
Error	49235.7	82	600.435		
Total	49385	83			
f5					
Columnas	8.05	1	8.048	0.01	0.9074
Error	49376.95	82	602.158		
Total	49385	83			
f6					
Columnas	3.4	1	3.44	0.01	0.9394
Error	49381.1	82	602.208		
Total	49384.5	83			
f7					
Columnas	476.2	1	476.19	0.8	0.371
Error	48908.8	82	596.449		
Total	49385	83			
f8					
Columnas	0.05	1	0.048	8e-5	0.9929
Error	49384.95	82	602.256		
Total	49385	83			
f9					
Columnas	466.7	1	466.714	0.78	0.3758
Error	48918.3	82	596.564		
Total	49385	83			
f10					
Columnas	105.2	1	105.19	0.18	0.6741
Error	49279.8	82	600.973		
Total	49385	83			

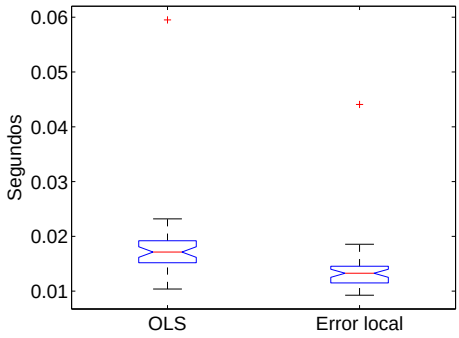
Tab. 4.2: Tablas resultado del test de Kruskal-Wallis sobre el error de aproximación tras la aplicación de la búsqueda local usando los operadores OLS y Error local.

Source	SS	df	MS	Chi-sq	Prob>Chi-sq
f1					
Columnas	25935.4	1	25935.4	43.61	4.01e-011
Error	23428.6	82	285.7		
Total	49364	83			
f2					
Columnas	15201.2	1	15201.2	25.55	4.31e-007
Error	34173.8	82	416.8		
Total	49375	83			
f3					
Columnas	15309	1	15309	25.74	3.93e-007
Error	34056	82	415.32		
Total	49365	83			
f4					
Columnas	8201.2	1	8201.19	13.79	0.0002
Error	41164.8	82	502.01		
Total	49366	83			
f5					
Columnas	9988.8	1	9988.76	16.8	4.14e-005
Error	39352.2	82	479.91		
Total	49341	83			
f6					
Columnas	8845.8	1	8845.76	14.87	0.0001
Error	40512.2	82	494.05		
Total	49358	83			
f7					
Columnas	7619.05	1	7619.05	12.82	0.0003
Error	41721.95	82	508.8		
Total	49341	83			
f8					
Columnas	11620.8	1	11620.8	19.54	9.83e-006
Error	37732.2	82	460.1		
Total	49353	83			
f9					
Columnas	6985.2	1	6985.19	11.75	0.0006
Error	42352.8	82	516.5		
Total	49338	83			
f10					
Columnas	13783.05	1	13783.05	23.19	1.46e-006
Error	35545.95	82	433.49		
Total	49329	83			

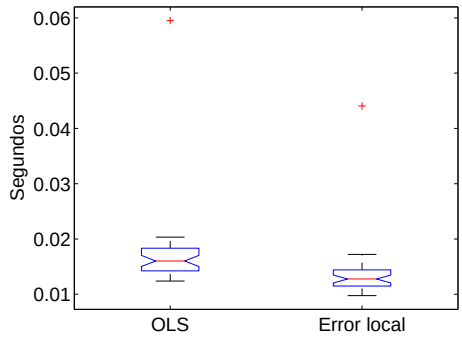
Tab. 4.3: Tablas resultado del test de Kruskal-Wallis para los operadores OLS y Error local cuando se comparan los tiempos de ejecución.



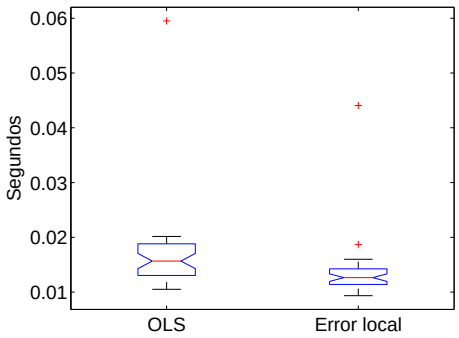
función 1



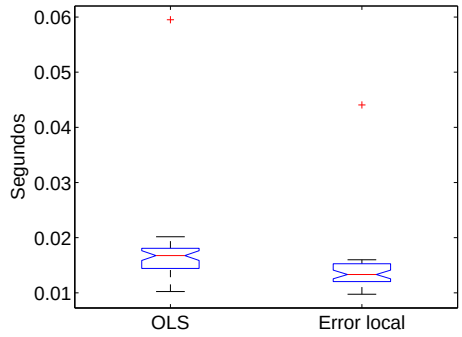
función 2



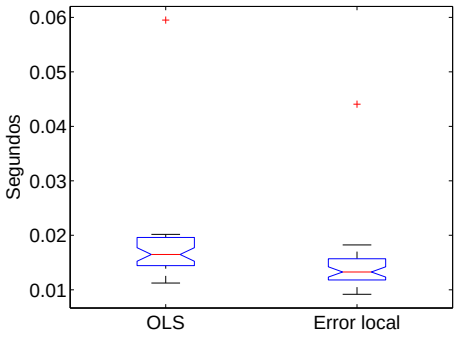
función 3



función 4



función 5



función 6

Fig. 4.9: Resultados del test de Kruskal-Wallis sobre los tiempos de ejecución de los dos métodos de identificación de la neurona más relevante en las 6 primeras funciones.

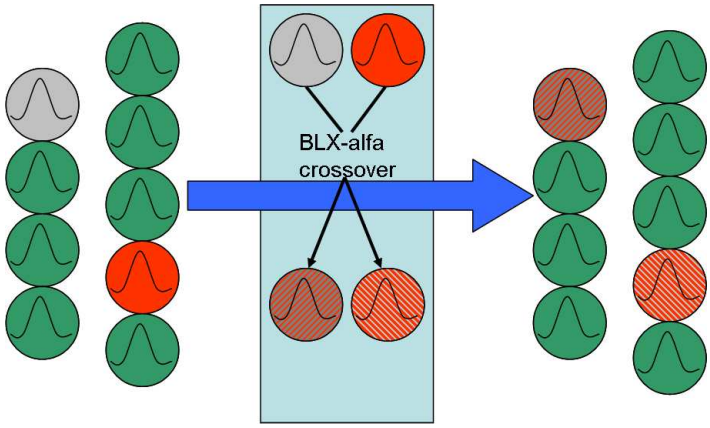


Fig. 4.10: Operador de cruce BLX-α

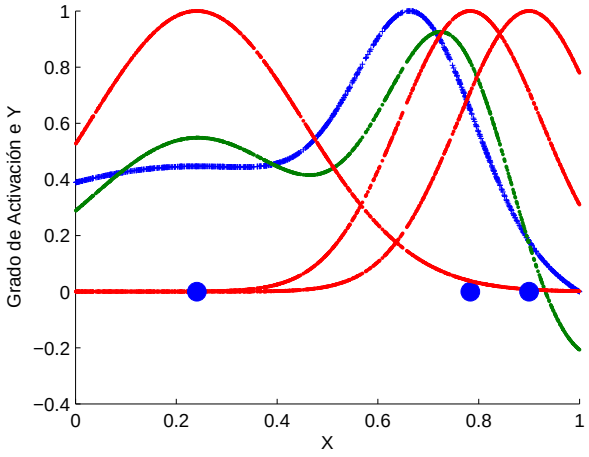


Fig. 4.11: Función objetivo a aproximar (azul), aproximación mediante una RBFNN con 3 neuronas (verde), valores de activación para cada neurona (rojo) y posición de los centros de las RBF (circulos). El error de aproximación (NRMSE) = 0.5238 .

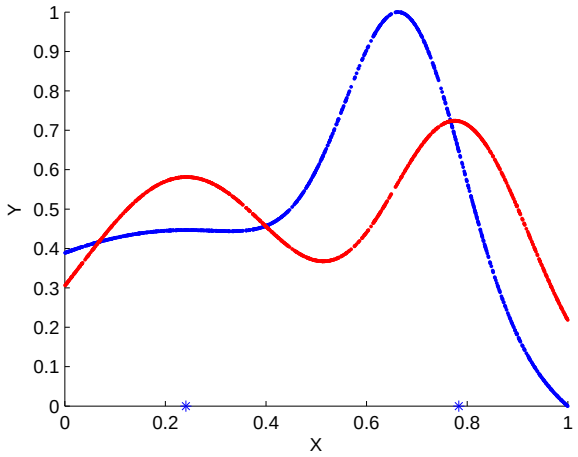


Fig. 4.12: Aproximación (rojo) con un error (NRMSE) = 0.9089 tras haber eliminado la primera de las neuronas que se solapan.

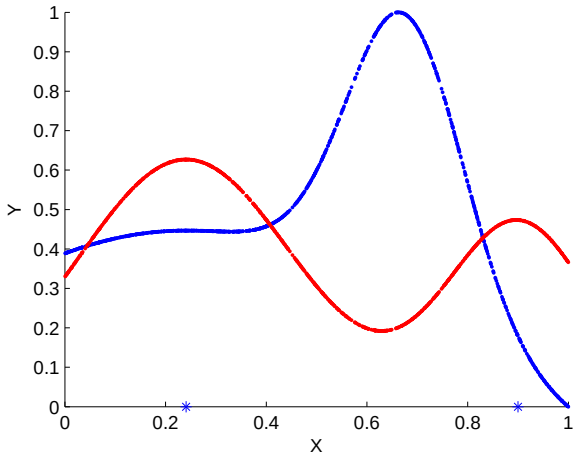


Fig. 4.13: Aproximación (rojo) con un error (NRMSE) = 1.4301 tras haber eliminado la segunda de las neuronas que se solapan

Esto conlleva a que las dimensiones de las matrices que almacenan los valores de activación de cada RBF puedan ser distintas en cada red. Cuando se realiza el intercambio de neuronas, la neurona seleccionada para ser intercambiada tomará los valores de las posiciones de los centros de la neurona de la otra red pero considerando únicamente las variables en común. Por ejemplo, dadas las siguientes redes:

Red_1:

variables = [1 0 1]

centros =

$\vec{c}_1 = (0.2 \ ; 0.3)$

$\vec{c}_2 = (0.4 \ ; 0.2)$

radios =

$r_1 = 0.6$

$r_2 = 0.1$

Red_2:

variables = [1 1 0]

centros =

$\vec{c}_1 = (0.15 \ ; 0.65)$

$\vec{c}_2 = (0.75 \ ; 0.95)$

radios =

$r_1 = 0.03$

$r_2 = 0.9$

si se eligen el centro $\vec{c}_2$ de la primera red y el centro $\vec{c}_1$ de la segunda, se obtienen los descendientes mostrados a continuación:

$Hijo_1$:

variables = [1 0 1]

centros =

$\vec{c}_1 = (0.2\ ;\ 0.3)$

$\vec{c}_2 = (\mathbf{0.75}\ ;\ 0.2)$

radios =

$r_1 = 0.6$

$r_2 = \mathbf{0.9}$

$Hijo_2$:

variables = [1 1 0]

centros =

$\vec{c}_1 = (\mathbf{0.4}\ ;\ 0.65)$

$\vec{c}_2 = (0.75\ ;\ 0.95)$

radios =

$r_1 = \mathbf{0.1}$

$r_2 = 0.9$

donde se muestran en negrita los valores obtenidos tras ser intercambiados.

Esta operación de intercambio de neuronas permite explorar todas las combinaciones de neuronas en distintas topologías pero no evoluciona los parámetros que definen una red. Este hecho impide explorar el espacio de solución de los centros y radios, no garantizando la obtención de una red que proporcione una buena aproximación. Para solventar este problema, se ha hibridado este cruce con uno específico para algoritmos con codificación real: el operador BLX-α (Eshelman et al., 1993) que ha sido seleccionado debido a su simplicidad y efectividad. Teniendo en cuenta que tanto los centros de las RBF como sus radios están codificados con números reales, gracias a este

operador, se podrán evolucionar esos valores en concreto.

El operador original consiste en dados dos individuos $I_1 = (i_1^1, i_2^1, ...i_h^1)$ e $I_2 = (i_1^2, i_2^2, ...i_h^2)$ con $(i \in \mathbb{R})$ se genera un descendiente $S = (s_1, ..., s_j, ..., s_h)$, donde s_j es un valor elegido aleatoriamente de una distribución uniforme dentro del intervalo $[i_{min} - \alpha \cdot B, i_{max} + \alpha \cdot B]$ donde $i_{min} = \text{mín}(i_j^1, i_j^2)$, $i_{max} = \text{máx}(i_j^1, i_j^2)$, $B = i_{max} - i_{min}$ y $\alpha \in \mathbb{R}$.

El valor del parámetro α afecta a la capacidad de exploración y de explotación del espacio de soluciones. En situaciones en las que no exista presión selectiva, un valor inferior a 0.5 causa que la población converja a los valores que hay en el centro de los intervalos, aumentado la posibilidad de originar una convergencia prematura a valores no óptimos. Un valor recomendado para α es 0.5 ya que representa un compromiso entre la explotación y la exploración del espacio de soluciones de los número reales (Eshelman et al., 1993).

Debido a que los individuos del algoritmo propuesto son más complejos que los descritos en el párrafo anterior, se ha de aplicar este operador de forma distinta. Cada RBFNN codificada posee un matriz una para almacenar las coordenadas de los centros y un vector para almacenar los radios así que, en primer lugar, se elige una RBF de cada individuo a ser cruzado obteniendo dos vectores de reales, uno que codifica el centro de la RBF y otro (de dimensión 1×1) que codifica el valor de su radio. Puesto que cada RBF tiene un conjunto de entradas distinto, las dimensiones de los vectores que codifican los centros de las RBF seleccionadas pueden ser distintas, en tal caso, sólo se opera con los valores asignados a las variables de entrada coincidentes. Para los radios se aplica el operador de forma independiente a los centros.

4.2.4.3. Operador de cruce 3: Cruce multipunto sobre las variables de entrada

Este operador de cruce es una extensión del original propuesto por (Holland, 1975) en el que, en vez de tener un único punto de cruce a partir del cual se intercambian los cromosomas, se establecen dos puntos de cruce definiendo un intervalo del cromosoma que será intercambiado con el otro progenitor. En la Figura 4.14 puede apreciarse cómo ocurriría un intercambio de genes entre dos cromosomas.

Aparentemente esta operación es simple, aunque si se analiza con detenimiento, el incorporar o eliminar variables de entrada a la red requiere una transformación considerable de ésta. Cada RBF posee una coordenada para cada variable de entrada de modo que, si se reduce el número de variables de entrada, habrá que eliminar las coordenadas correspondientes a cada RBF. Por el contrario, si se añaden variables, será necesario asignar valores a esas coordenadas. El primer caso en el que se reduzcan variables tiene fácil solución, se descartan los valores existentes, sin embargo, para el segundo caso, la solución no es tan sencilla. Esta segunda situación también se presentó cuando se describió el operador 1 ya que al añadir una neurona a otra red, también es posible que la red de destino tenga variables nuevas. Al igual que en el caso del operador 1, la asignación de un valor aleatorio parece la opción más adecuada ya que es eficiente, rápida y asegura diversidad y exploración del espacio solución.

4.2.5. Operadores de mutación

La gran cantidad de elementos a evolucionar han originado el diseño de varios operadores de cruce como se ha mostrado en la subsección anterior. Los operadores de mutación no son una excepción y se ha diseñado una gran

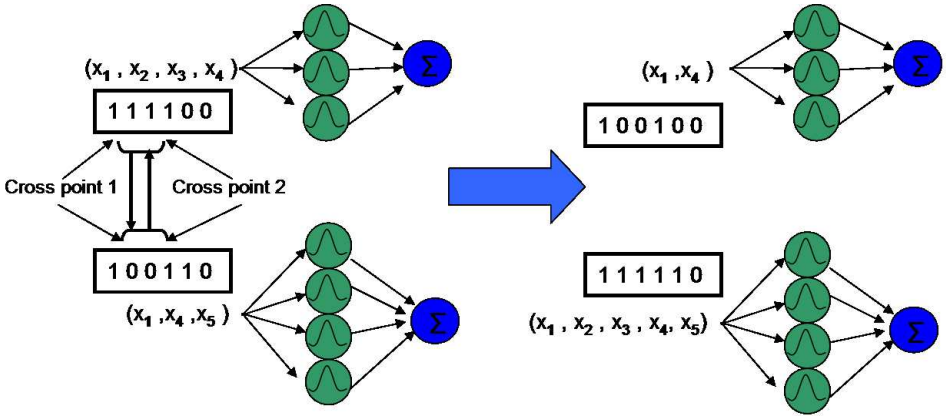


Fig. 4.14: Operador de cruce multipunto

cantidad de ellos para influir sobre todos los aspectos del diseño de RBFNN. Al igual que con los operadores de cruce, unos operadores de mutación influirán de forma totalmente aleatoria sobre los individuos mientras que otros utilizarán la salida de la función a aproximar para intentar guiar la búsqueda y explotar más ciertas áreas concretas del espacio solución. El tratamiento que se ha hecho en los casos en los que hay que añadir o quitar coordenadas a los centros es igual que en los operadores de cruce.

4.2.5.1. Mutaciones ciegas

Dentro de las mutaciones que actúan de forma completamente aleatoria se proponen:

- Adición de una RBF: se añade una RBF con la posición de su centro y el valor de su radio aleatorios.
- Eliminación de una RBF: es el opuesto al anterior, se selecciona una RBF aleatoriamente y se elimina de la red. Se ha establecido una restricción mediante la cual únicamente se puede aplicar este operador si la red posee más de 2 RBF.

- Mutación de codificación real sobre los centros: todas las coordenadas de los centros de una RBF se ven desplazadas una distancia aleatoria comprendida entre [-0.5,0.5] dado que el espacio de entrada está normalizado a [0,1].

- Mutación de codificación real sobre los radios: al igual que los centros, el valor del radio de una RBF se incrementa o decrementa con un valor aleatorio no superior a 0.5.

- Mutación de bit: este operador selecciona/deselecciona una variable de entrada a la red asignando el valor opuesto a uno de los bits del vector que codifica las variables de entrada de la red.

Los dos primeros operadores modifican la estructura de la red, mientras que el tercero y cuarto modifican los parámetros de las RBF, y el quinto afecta a las variables de entrada de la red, por tanto es posible saltar a cualquier posición del espacio solución mediante estos operadores.

4.2.5.2. Mutaciones con conocimiento experto

Los operadores de mutación que se presentan a continuación utilizan la información que se dispone sobre la salida de la función a aproximar, permitiendo guiar la búsqueda a zonas donde se encuentren valores para los parámetros que permitan disminuir el error de la red o evitar situaciones de sobreentrenamiento. Los operadores son:

- Inserción de una RBF: se inserta una RBF con la posición del vector de entrada en el cual hay un mayor error de aproximación.

- Eliminación de una RBF: este operador es el opuesto al anterior y elimina la neurona más cercana al vector de entrada en el cual el error de

aproximación es el más pequeño. Aunque pueda parecer ilógico, esta operación puede evitar situaciones de sobreentrenamiento así como de posibles redundancias entre las redes. Se ha de considerar que el algoritmo mantendrá el Pareto considerando los ancestros, por tanto, aunque la neurona eliminada fuese una correcta, el antecesor del individuo permanecerá en la población conservandola.

- El tercer operador introduce una RBF cerca de la RBF que posee un mayor error local.

- El cuarto operador elimina de la red la neurona que posee el menor error local. El razonamiento para aplicar este operador es el mismo que el empleado en el segundo. En este caso, todavía tiene más sentido dado que uno de los operadores de cruce añade la neurona con menor error local así que, si se elimina ésta, el material genético que recombinará será más diverso.

- El quinto operador aplica una búsqueda local (Levenberg-Mardquardt) para hacer un ajuste fino de los valores de los radios y de los centros aunque teniendo en cuenta que sólo unas pocas iteraciones deben realizarse, de lo contrario, se corre el riesgo de caer en mínimos locales (González et al., 2003).

4.2.6. Función de evaluación

La función de evaluación es la que indica la calidad de un individuo y la que establece los valores de las soluciones para el problema de optimización multiobjetivo. En el algoritmo propuesto, se ha establecido que la función de evaluación debe proporcionar dos valores: el número de RBF (f_1) que posee la red y el error de aproximación f_2, definiendo la función de evaluación como:

$$f(\iota_j) = (f_1(\iota_j), f_2(\iota_j)) \tag{4.14}$$

donde ι_j es una solución del espacio solución, en el caso que se trata en esta memoria, una RBFNN con su conjunto de variables de entrada.

El error de aproximación se puede relativizar ya que se aproxima un conjunto de datos que, en la aplicación real a un problema, puede cambiar constantemente con la inclusión de nuevos vectores de entrada. Gracias a este hecho, se puede manipular el conjunto de datos entrenando a la red con distintos subconjuntos para ver si los parámetros que definen a la red son adecuados para que ésta sea capaz generalizar o no. El objetivo es intentar minimizar el hecho de que el conjunto de entrenamiento se aprenda de forma muy precisa de modo que cuando la red tenga que procesar datos nuevos no sea capaz de generalizar. Para evitar estas situaciones se ha incorporado la técnica de validación cruzada creando 3 subconjuntos a partir del conjunto de entrenamiento inicial. Se ha estimado que 3 subconjuntos es el valor mínimo para que haya diversidad en el conjunto de entrenamiento sin sacrificar prestaciones. La perdida de prestaciones es causada por la necesidad de recalcular los pesos de la red para cada subconjunto.

Se ha de tener en cuenta que, aunque se evalúe la calidad de un individuo en función de la validación cruzada, todos los operadores que utilizan conocimiento experto consideran todos los datos del conjunto de entrenamiento original. Por otra parte, la aplicación de la búsqueda local una vez finalizada la parte evolutiva del algoritmo memético también utilizará todo el conjunto de entrenamiento.

4.2.7. Criterio de dominancia difuso

Como se mostró previamente en la introducción del algortimo NSGA-II, para definir un Pareto se establece un criterio denominado dominancia, mediante el cual se define si una solución domina a otra. En la definición de este criterio se utilizan comparaciones estrictas para cada objetivo, siguiendo el orden normal establecido para los número enteros en el caso del tamaño de la RBF, y de los números reales para el caso del error de aproximación.

Esta conlleva a que a la hora de comparar dos soluciones ι_1 y ι_2 puede darse el caso de $f(\iota_1) = (f_1(\iota_1), f_2(\iota_1))$ =(3 , 0.5) y $f(\iota_2) = (f_1(\iota_2), f_2(\iota_2))$ =(9 , 0.50005). Según el criterio establecido, las dos soluciones no se dominan entre ellas pudiendo estar dentro del mismo Pareto. Evidentemente, si tenemos una red que proporciona un error de 0.5 con 3 RBF y otra con 9 cuyo error es de 0.50005, es preferible la que tiene un tamaño menor debido a que es más probable que sea capaz de generalizar más. El ejemplo anterior puede parecer un caso extremo pero en la práctica es bastante probable ya que el diseño de las RBFNN se realiza de una forma incremental, usando uno de los operadores de cruce y varios de mutación, así que es posible que una misma red vaya añadiendo neuronas paulatinamente decrementando muy levemente el error de aproximación, entrando a formar parte del Pareto sin que sus neuronas hayan evolucionado. Es más, el Pareto puede verse repleto de redes de gran tamaño entre las cuales no hay una gran mejora del error de aproximación (Figura 4.15). Las consecuencias de tales situaciones son la dificultad para explotar el espacio solución, ya que se mantienen individuos no adecuados en el Pareto, y la ineficiencia del algoritmo al tener que operar con redes de gran tamaño.

Para solventar este inconveniente se propone redefinir el criterio de dominancia tal que, en vez de realizar comparaciones estrictas para los valores del

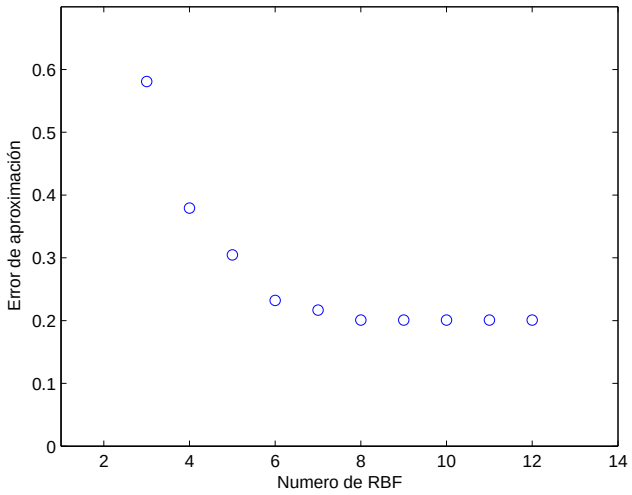


Fig. 4.15: Pareto donde no se mejoran significativamente las soluciones con respecto al error de aproximación (eje Y)

error de aproximación, se realice una comparación difusa donde se establezca que un número es menor que otro si existe una diferencia apreciable entre ellos. Para implementar esta idea, una solución ι_j define el conjunto difuso μ_{ι_j} como:

$$\mu_{\iota_j} = \begin{array}{ll} \dfrac{1}{1+|f_2(\iota_i)-f_2(\iota_j)|} & si \quad f_1(\iota_i) > f_1(\iota_j) \\ 1 & si \quad f_1(\iota_i) <= f_1(\iota_j) \end{array} \tag{4.15}$$

Utilizando este conjunto difuso se puede reescribir el criterio de dominancia como:

$$\begin{array}{rl} \iota_1 \prec \iota_2 \Longrightarrow & (f_1(\iota_1) < f_1(\iota_2) \quad \wedge \quad \mu_{\iota_1}(f_2(\iota_2)) > \alpha_{cut}) \\ & \vee \\ & (f_1(\iota_1) = f_1(\iota_2) \quad \wedge \quad f_2(\iota_1) < f_2(\iota_2)) \end{array} \tag{4.16}$$

El inconveniente de este método es el tener que definir un valor para α_{cut}, aunque gracias a que se emplea el NRMSE para medir la calidad de las aproximaciones, se puede elegir un valor que sea adecuado para cualquier problema ya que el intervalo en el que se mueve el NRMSE es igual para todos.

4.3. Experimentos

Todos los elementos descritos en la sección anterior fueron integrados en un único algoritmo cuya estructura general será descrita en el próximo capítulo (Capítulo 5, sección 5.1) donde se analizarán distintos aspectos del paradigma de computación paralela.

Esta sección muestra cómo, efectivamente, el problema del diseño de RBFNN y de la selección de variables se resuelve satisfactoriamente mediante el algoritmo diseñado en esta memoria.

Para todos los ejemplos, el algoritmo ha sido ejecutado 20 veces durante 250 generaciones utilizando los siguientes parámetros:

- tamaño de la población = 40. El valor de este parámetro ha sido establecido tras hacer varias ejecuciones variando su valor entre 30 y 70. La razón de establecer estos límites es debido a qué las islas deben tener diversidad para no converger prematuramente, pero al mismo tiempo deben explotar sus soluciones gracias a su especialización, por tanto es necesario un valor de la población ni muy grande ni muy pequeño.

- probabilidad de cruce = 0.8 mutación = 0.1. Estos valores son acordes a los valores más comunes empleados en la bibliografía (Herrera et al., 1998).

centros		radios	pesos
0.2502	0.5961	0.0503	1.3835
0.8562	0.4738	0.1598	-0.8553
0.0487	0.8987	0.0459	0.2882
0.1973	0.0992	0.1848	1.5188
0.2228	0.9978	0.2257	-0.4512
0.1906	0.9913	0.1509	2.1927
0.0037	0.5445	0.0745	0.9518
0.5573	0.1110	0.0364	1.9960

Tab. 4.4: Parámetros de la función G_3

Aunque existen propuestas para la autoregulación dinámica de ciertos parámetros de los algoritmos (Davis, 1989); (Fogarty, 1989); (Schaffer and Morishma, 1987); (Srinivas and Patnaik, 1994a), en esta memoria se han considerado los valores fijados anteriormente porque sus valores están argumentados y el hacer un estudio de como adaptar los distintos parámetros dinámicamente sería muy complejo dadas las carácterísticas del algoritmo. En concreto, como se mostrará en el siguiente capítulo, el hecho de ser un algoritmo paralelo que especializa sus operadores y que resuelve el problema de la selección de variables junto con el diseño de la red en paralelo hace que el estudio de la influencia de todos los parámetros sea tan complejo que queda fuera de esta memoria aunque sí se considera la posibilidad de incorporar mecanismos autoreguladores en un futuro.

4.3.1. Función G_3

En este caso, el objetivo es aproximar una función sintética (G_3) de 2 dimensiones diseñada calculando la salida de una RBFNN gaussiana sobre una malla de 20×20 puntos. Los parámetros de la red han sido obtenidos mediante la generación de valores aleatorios y se muestran en la Tabla 4.4.

Los datos de esta función han sido modificados antes de usarlos como

entrada para el algoritmo con el objetivo de ver si éste es capaz de identificar las variables originales que definen la función además de obtener una buena aproximación de la función. Para poder estudiar el comportamiento del algoritmo se han añadido 8 variables de entrada más a las 2 dimensiones originales de la función. Las 10 variables de entrada de la función que se le han proporcionado al algoritmo como entrada se han calculado del siguiente modo:

$$
\begin{aligned}
X_1^s &= rand \\
X_2^s &= X_1 + (rand \cdot 0{,}5) \\
X_3^s &= rand \\
X_4^s &= rand \cdot 0{,}5 \\
X_5^s &= X_1 \\
X_6^s &= rand \\
X_7^s &= X_2 \\
X_8^s &= X_2 + (rand \cdot 0{,}5) \\
X_9^s &= X_1 + (rand \cdot 0{,}15) \\
X_1^s 0 &= X_2 + (rand \cdot 0{,}15)
\end{aligned}
$$

donde *rand* representa un vector columna con números aleatorios de una distribución uniforme con valores en $[0{,}1]$, X_1 representa todos los valores de la primera variable de entrada de la función original y X_2 los valores para la segunda. Por tanto, el algoritmo recibirá una serie de valores de ciertas entradas totalmente aleatorios y sin ninguna relación con la salida, otra serie de variables con los datos originales distorsionados por ruido, y los valores originales de la función (las variables X_6^s y X_7^s). El resultado de la ejecución será satisfactorio si el algoritmo es capaz de identificar las variables originales como entrada de las redes que evolucione.

Los resultados obtenidos han sido satisfactorios dado que el algoritmo ha sido capaz de identificar las variables originales que definen la salida y, al mismo tiempo, ha proporcionado modelos muy precisos que aproximan la

función. En concreto el error de entrenamiento ha sido de 0.0684 (0.0921) y el de test 0.1100 (0.0703) donde el primer valor es la medida del error es el NRMSE y el valor entre paréntesis es la desviación estándar. Si se observa este último valor podemos ver cómo el algoritmo proporciona resultados robustos.

Para mostrarla efectividad del enfoque memético frente al clásico evolutivo, se han ejecutado independientemente los algoritmos de *clustering* seguidos de la búsqueda local realizándose 500 ejecuciones, seleccionándose el mejor resultado obtenido. Por otra parte, también se ha ejecutado el algoritmo genético multiobjetivo sin realizar la inicialización de la población y sin aplicar la búsqueda local tras realizar todas las generaciones. Los resultados obtenidos en comparación con el algoritmo memético se muestran en la Tabla 4.5. Como reflejan estos resultados, el algoritmo memético es capaz de obtener modelos más precisos de manera más robusta.

	Training	**Test**
Algoritmo propuesto	0.0684 (0.0921)	0.1100 (0.0703)
Genético	0.1184 (0.1021)	0.1579 (0.0917)
Clustering	0.1216	0.1799

Tab. 4.5: Errores de aproximación para la función G_3

4.3.2. Función G_4

La función G_4 está diseñada del mismo modo que la función anterior, aunque se ha añadido una variable de entrada más, siendo sus parámetros los incluidos en la Tabla 4.6.

Al igual que se ha procedido con la función G_3, se han añadido unas coordenadas a las originales de la función mediante:

centros			**radios**	**pesos**
0.9501	0.4565	0.9218	0.4103	0.5556
0.2311	0.0185	0.7382	0.8936	0.8111
0.6068	0.8214	0.1763	0.0579	0.7949
0.4860	0.4447	0.4057	0.3529	2.4152
0.8913	0.6154	0.9355	0.8132	1.0888
0.7621	0.7919	0.9169	0.0099	0.7953

Tab. 4.6: Parámetros de la función G_4

$$
\begin{aligned}
X_1^{2s} &= rand \\
X_2^{2s} &= X_3 + (rand \cdot 0{,}5) \\
X_3^{2s} &= rand \\
X_4^{2s} &= X_3 \\
X_5^{2s} &= (rand \cdot 0{,}5) \\
X_6^{2s} &= X_1 \\
X_7^{2s} &= rand \\
X_8^{2s} &= X_2 \\
X_9^{2s} &= X_2 + (rand \cdot 0{,}5) \\
X_1^{2s}0 &= X_1 + (rand \cdot 0{,}15) \\
X_1^{2s}1 &= X_3 + (rand \cdot 0{,}35) \\
X_1^{2s}2 &= X_2 + (rand \cdot 0{,}75) \\
X_1^{2s}3 &= X_1 + (rand \cdot 0{,}5)
\end{aligned}
$$

En esta ocasión se obtienen 13 dimensiones de las cuales 3 son las originales que definen la función (las variables X_4^{2s}, X_6^{2s} y X_8^{2s}). Tras ejecutar el algoritmo propuesto se obtienen los siguientes resultados:

Error de aproximación de test = 0.1105 (0.0973)

Error de aproximación de entrenamiento = 0.1025 (0.0880)

Número de RBF = 9.2 (2.8206)

Variables seleccionadas = 0 % 0 % 0 % 100 % 0 % 80 % 0 % 100 % 0 % 20 %
0 % 0 % 0 %

Los resultados muestran cómo el algoritmo obtiene unas buenas aproximaciones de la función tanto para el conjunto de entrenamiento como para el de test. Los errores de aproximación no tienen una gran desviación estándar lo que muestra la robustez del algoritmo. Al igual que con los errores de test, las RBFNN poseen una complejidad parecida con 9.2 RBF de media lo cual no difiere mucho de las 6 RBF que definen la función original, por tanto, el algoritmo mantiene un compromiso aceptable entre complejidad y calidad de aproximación. Por último, se puede apreciar cómo el algoritmo es capaz de identificar las variables originales con una gran precisión, concretamente, las variables X_1 y X_2 se seleccionan en el 100 % de las ejecuciones. X_3 es seleccionada en el 80 % de los casos aunque cuando no es seleccionada, se selecciona X_9^{2s} la cual se corresponde con X_3 con ruido añadido.

4.3.3. Serie temporal de Mackey-glass

Las series temporales basadas en la ecuación diferencial de Mackey-Glass (Mackey and Glass, 1977) se suelen utilizar como patrón de test en la mayoría de algoritmos de identificación y optimización de modelos. Estas series se generan a partir de la siguiente ecuación en diferencias:

$$\frac{ds(t)}{dt} = \alpha \cdot \frac{s(t-\tau)}{1+s^{10}(t-\tau)} - \beta s(t) \tag{4.17}$$

Siguiendo las indicaciones propuestas en estudios previos (Whitehead and Choate, 1996), los parámetros libres han sido fijados a $\alpha = 0{,}2$ y $\beta = 0{,}1$, lo que convierte a la ecuación diferencial anterior en una serie temporal caótica sin un periodo claramente definido; no converge ni diverge y es muy sensible a las condiciones iniciales.

De los 1000 datos que se proporcionan para realizar pruebas con esta serie, 500 han sido utilizados para entrenamiento y el resto para test. Cada

vector de entrada se define como una componente de 50 variables de entrada correspondientes a los valores en los 50 instantes de tiempo previos al valor de salida actual. Los resultados obtenidos tras la ejecución del algoritmo son:

Error de aproximación de test = 0.0191 (0.0113)
Error de aproximación de entrenamiento = 0.0193 (0.0113)
Número de RBF = 6.6 (3.5653)
Variables seleccionadas = 70 % 50 % 30 % 20 % 0 % 10 % 10 % 10 % 10 %
10 % 20 % 20 % 10 % 20 % 20 % 20 % 30 % 30 %
20 % 20 % 60 % 40 % 40 % 10 % 40 % 10 % 10 %
10 % 0 % 10 % 10 % 0 % 0 % 10 % 10 % 20 %
0 % 10 % 0 % 0 % 10 % 0 % 10 % 20 % 10 %
0 % 10 % 0 % 10 % 0 %

Al igual que ocurrió con las anteriores funciones, las RBFNN entrenadas utilizando el algoritmo obtienen resultados bastante precisos, con poca complejidad y siendo capaces de determinar que variables dentro de los vectores de entrada son más relevantes.

4.4. Conclusiones

Este capítulo ha presentado los componentes de un nuevo algoritmo que recibe como entrada una serie de muestras con su valor de salida asociado y es capaz de diseñar una RFBNN que, no solo infiere la relación entre las entradas y la salida, sino que es capaz de determinar qué elementos (variables) del vector que identifica cada muestra son necesarios para poder calcular una nueva salida a partir de una entrada nueva.

Para poder realizar este diseño, el algoritmo hibrida dos técnicas de diseño de RBFNN, la búsqueda local y la evolutiva, integrándolas de modo que suplan las carencias de una y de otra entre ambas. Dentro de la optimización mediante búsqueda local, se ha hecho uso de los nuevos algoritmos presentados en el capítulo anterior. En el campo de los algoritmos evolutivos, el nuevo algoritmo ha presentado varios elementos nuevos:

1. la propuesta de nuevos operadores de cruce y de mutación para evolucionar una RBFNN

2. un nuevo método para controlar la complejidad de la red basado en el criterio para determinar la dominancia entre soluciones.

5. PARALELISMO

Tras haber diseñado los distintos componentes para construir un algoritmo evolutivo multiobjetivo que diseñe RBFNN en el Capítulo 4, este capítulo tiene la tarea de ensamblarlos de modo que el funcionamiento del conjunto sea adecuado.

Para poder aprovechar las posibilidades en cuanto a paralelismo de datos y paralelismo funcional que el algoritmo evolutivo, tal y como se ha presentado, posee, se ha propuesto un algoritmo paralelo basado en el modelo de islas. Esto permitirá al algoritmo ser más veloz y encontrar mejores soluciones, como se mostrará a lo largo del capítulo.

A la hora de implementar el algoritmo para que se ejecute en una plataforma de computo paralela concreta, es necesario disponer de las herramientas software para hacerlo. Dado que no existen herramientas que sean lo suficientemente portables, se ha diseñado una interfaz que permite ejecutar el algoritmo en cualquier plataforma que proporcione una biblioteca de funciones de paso de mensajes.

5.1. Paradigma de paralelización

El algoritmo desarrollado en esta memoria (**Guillén** et al., 2007e) se ha implementado en una plataforma paralela dividiendo la población en subpoblaciones que evolucionan independientemente y se comunican cada cierto número de generaciones. Este tipo de implementación se corresponde con los paradigmas denominados algoritmos genéticos (*Genetic Algorithms*, GA) de grano grueso (Spears et al., 1993), GA distribuidos (Yao, 1999) o modelo de islas (Falkenauer, 1998).

Existen varias razones que motivan el diseño de GA paralelos (van Veldhuizen et al., 2003), entre ellas, la necesidad de reducir el tiempo de ejecución y aumentar la capacidad de explorar el espacio de búsqueda han sido las que han influido en el diseño del algoritmo propuesto: pEFSFA (*parallel Evolutionary Feature Selection and Function Approximator*). Estos dos objetivos se obtienen directamente tras hacer una implementación básica del algoritmo en la que cada procesador ejecute el mismo algoritmo pero, en vez de con toda la población, con un subconjunto de ésta. Sin embargo, el diseño de un algoritmo paralelo que sea capaz de aprovechar profundamente todas las ventajas que ofrece el paralelismo es una tarea compleja. Tal y como se propuso en (Lin et al., 1994), se pueden distinguir dos tipos de algoritmos genéticos distribuidos basados en islas:

- Homogéneos: Cada subpoblación de individuos generados aleatoriamente realiza el mismo tipo de búsqueda utilizando los mismos operadores.

- Heterogéneos: Las subpoblaciones son evolucionadas de distinto modo utilizando distintas configuraciones de los algoritmos, operadores, etc. Dentro de esta categoría se pueden distinguir varios niveles de hetero-

geneidad:

- A nivel de parámetro: se ejecuta el mismo código en cada isla aunque se utilizan distintos parámetros para el cruce, la mutación y la migración de individuos entre islas. Estos parámetros pueden ser prefijados (Adamidis and Petridis, 1996); (Adamidis and Petridis, 2002), asignados aleatoriamente durante la ejecución del algoritmo (Miki and Negami, 1999); (Miki et al., 1999) o pueden seguir una estrategia autoadaptativa (Hinterding et al., 1996); (Schnecke and Vornberger, 1996).
- A nivel de operador: cada isla ejecuta distintos operadores de cruce y mutación (Herrera and Lozano, 2000).
- A nivel de representación: cada isla posee individuos que codifican de distinto modo las soluciones, implicando por tanto el uso de distintos operadores de cruce y mútación (Aickelin and Bull, 2002); (Lin et al., 1994).

En (Herrera and Lozano, 2000); (Alba et al., 2004); (Alba et al., 2003); (**Guillén** et al., 2006a); (**Guillén** et al., 2007f) se muestra cómo, efectivamente, una implementación distribuida heterogénea de un algoritmo evolutivo no sólo permite una ganancia en velocidad sino también una mejora en los resultados.

En el caso particular del diseño de RBFNN, como ya se ha comentado, existen varios aspectos que pueden ser tratados por separado dando lugar a operadores de cruce y mutación específicos. Esto ha sido determinante para diseñar el algoritmo pEFSFA de modo que cada isla evoluciona las redes en sus distintos aspectos y, periódicamente, intercambia redes con las otras islas. El algoritmo pEFSFA presenta 3 tipos de islas:

- Isla 1: En esta isla se lleva a cabo la evolución de la estructura de las redes. Se aplica el operador de cruce 1, especifico para la evolución de la estructura de la red y las mutaciones que se utilizan modifican la estructura de la red introduciendo o eliminando neuronas, tanto con conocimiento experto como sin él.

- Isla 2: La segunda isla evoluciona los parámetros de cada RBF, para ello se vale del operador de cruce 2 que permite explorar los posibles valores de los centros y los radios gracias a la inclusión del operador BLX-α. Para especializar la isla se han incorporado las mutaciones sobre codificaciones reales donde se modifican aleatoriamente los valores para los centros y los radios así como la búsqueda local, puesto que también modifica el valor de estos parámetros.

- Isla 3: La tarea de descubrir qué conjunto de variables es el más apropiado para aproximar la función objetivo es llevada a cabo por la tercera isla. Aquí se utiliza el cruce binario multipunto sobre la parte del individuo que codifica las variables de entrada así como la mutación que selecciona/deselecciona una variable.

Cada isla se comunica con las otras dos mediante un mecanismo de migración por el cual se intercambian individuos cada cierto número de generaciones. El esquema de comunicación que sigue el algoritmo se muestra en la Figura 5.1. El tipo de comunicaciones corresponde a una topología completamente conectada donde cada isla (proceso) se comunica con el resto. Aunque este tipo de topologías incrementan el tráfico de mensajes, pueden proporcionar mejores resultados (Cantú-Paz, 2000). En el caso concreto del algoritmo de esta memoria, es más que recomendable puesto que hay una especialización en la cual un mismo individuo puede verse evolucionado en

ciertos aspectos en cada isla. Como consecuencia del planteamiento del algoritmo, a la hora de establecer qué individuos se envían a las otras islas, se ha decidido que se intercambien los individuos de la Pareto frontera. Gracias a esto, es posible mantener una Pareto frontera global que sea actualizada cada cierto número de generaciones de modo que, en ese número de generaciones, el Pareto común sea evolucionado en los diferentes aspectos del diseño de las RBFNN. Además, cada isla mantendrá su propia población especializada en cierto aspecto, manteniendo la diversidad y la exploración de soluciones. En (Cantú-Paz, 2000) se realiza un estudio intensivo sobre los algoritmos genéticos distribuidos, sin embargo, se hace a nivel de un algoritmo genético básico con codificación binaria y con todas las islas ejecutando el mismo código. Dada la complejidad del problema y el distinto mecanismo de representación de soluciones, desafortunadamente, ninguna de las conclusiones pueden aplicarse al algoritmo propuesto. Por ejemplo, se recomienda que se realice una migración una vez que la población converja, sin embargo, el algoritmo propuesto debe intercambiar la información constantemente para poder evaluar la calidad de los individuos en los distintos aspectos, es decir, para poder determinar qué un conjunto de variables de entrada es adecuado, es necesario diseñar redes que lo utilicen, y estas redes deben ser diseñadas teniendo en cuenta la topología el valor de sus parámetros. Concluyendo, dada la peculiaridad del algoritmo, no se pueden seguir la orientaciones propuestas en (Cantú-Paz, 2000).

Como se puede observar tras el párrafo anterior, son obvios los beneficios de haber seleccionado el algoritmo *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) como esqueleto para construir el algoritmo tal y como se describió en la sección 4.1. Su estrategia de elitismo, en la cual se van seleccionando sucesivamente Paretos no dominados, sugiere que el conjunto

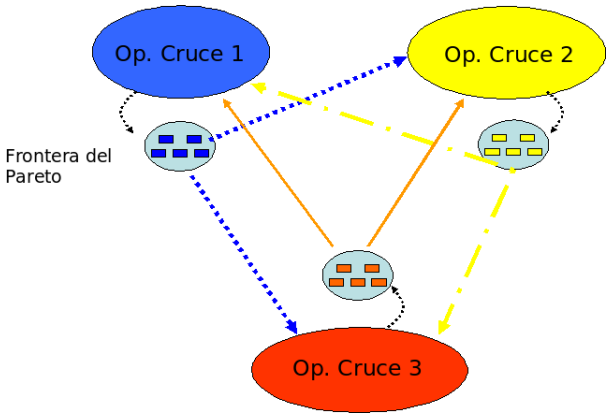


Fig. 5.1: Topología completamente conectada

de individuos a migrar sea el Pareto frontera de cada población. Además, una vez que las poblaciones converjan al Pareto, este podrá estirarse de modo que se exploten las soluciones contenidas en él. Otra ventaja del modo en el que se obtiene la población de la siguiente generación es que resulta más sencillo integrar a los individuos de otras islas ya que tan solo han de unirse a la población constituida por padres e hijos y someterla al proceso de ordenación no dominada.

Por último, un parámetro nuevo debe ser introducido en el diseño del algoritmo: el que determina cuándo debe realizarse una migración. En primer lugar se considera el valor más extremo, una migración por iteración. Si este caso ocurre, el coste de las comunicaciones entre los procesos podría afectar al comportamiento del algoritmo, ralentizándolo demasiado. Además, las islas no tendrían oportunidad de explotar unos individuos en concreto puesto que la Pareto frontera podría sufrir demasiados cambios. Obviamente, el caso opuesto en el que apenas se realicen migraciones implica la convergencia de las poblaciones y no es acorde con la filosofía del algoritmo. Un estudio simplificado de este hecho se muestra en la sección 5.2.

5.1.1. Esquema Final

Una vez que se han descrito los componentes del algoritmo y cómo se relacionan entre ellos, es posible representar en la Figura 5.2 el esquema que sigue el algoritmo completo. En el esquema se ve reflejada la aplicación de búsquedas locales al principio y al final del algoritmo, lo que lo confina dentro de la categoría de algoritmo memético como ya se comentó en la sección 4.1. Otro elemento fácilmente visible es el paralelismo, el cual, rompe el flujo de ejecución de una generación normal y permite el intercambio de individuos entre las distintas islas. Cuando una isla recibe todos los Paretos de las otras islas, elimina los repetidos y los incluye en su población para poder continuar con el mecanismo de la ordenación utilizado en el algoritmo NSGA-II.

5.2. Beneficios del Paralelismo Funcional

Para poder ilustrar los beneficios del paralelismo a nivel de obtención de mejores resultados, se han realizado diversas pruebas utilizando una versión simplificada del mismo (**Guillén** et al., 2006a); (**Guillén** et al., 2007f). Esta versión simplificada no considera el problema de seleccionar las variables de entrada de las redes y se centra más en el problema de evolucionar la estructura de la red, confiando el ajuste de los parámetros de las neuronas a los operadores de mutación y a la búsqueda local. Esto simplifica en gran medida el problema a resolver pero es lo suficientemente ilustrativo como para justificar el paralelismo dado que si gracias a éste es posible mejorar los resultados para el problema simple, cuanto más complejo sea el problema, mayores beneficios se obtendrán.

Dado que el problema ha sido reducido en envergadura, los operadores que actúan sobre las RBFNN también se han disminuido así que en esta

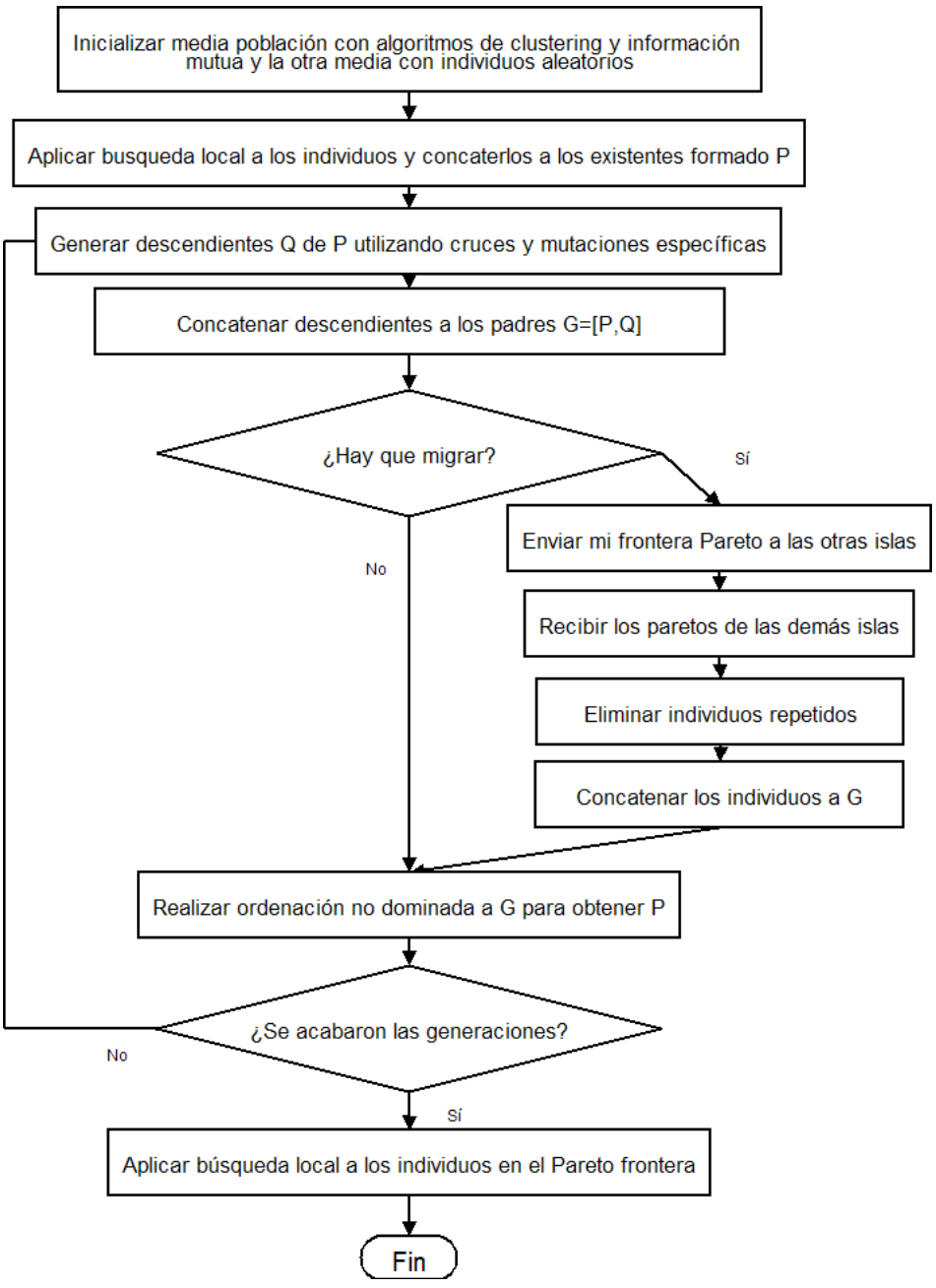


Fig. 5.2: Esquema general del algoritmo pEFSFA

subsección solo se consideran: operador de cruce de intercambio de neuronas tal como fue propuesto en (González et al., 2003) (Cruce 1), operador de cruce de la adición de la neurona con menor error local (Cruce 2), operadores de mutación real sobre centros y radios (Movimiento) y operadores de mutación de adición de una neurona aleatoria (Incremento) y de eliminación de una neurona aleatoria (Decremento). Las combinaciones de estos operadores han dado fruto a los siguientes algoritmos distribuidos y heterogéneos:

1. *División de los operadores de cruce (P1).* Esta implementación consiste en hacer una especialización a nivel de los operadores de cruce definiendo 2 islas: una en la que se ejecutará el operador de cruce 1 y otra que ejecutará el cruce 2. Esta primera implementación paralela será confrontada con las implementaciones no distribuidas para ver cómo la especialización de los operadores de cruce puede proporcionar mejores resultados.

2. *Cruce 1 + Movimiento + Decremento y Cruce 2 + Movimiento + Incremento (P2).* Esta combinación de operadores tiene como objetivo aumentar la capacidad de exploración de topologías del Cruce 2 y la de explotación de soluciones del Cruce1. El aumento de la capacidad de exploración del Cruce 2 viene dado por el operador de mutación que añade una neurona de tal modo que sólo produzca redes de mayor tamaño. Para compensar este efecto, la otra isla incorpora el operador Decremento pudiendo reducir así el tamaño de las redes que se intercambien.

3. *Cruce 1 + Movimiento + Incremento y Cruce 2 + Movimiento + Decremento (P3).* Esta implementación hace lo opuesto a la anterior, restringiendo el crecimiento de la redes en la isla que utiliza el Cruce 2

mediante el empleo únicamente del operador Decremento. Además, la isla con el Cruce 1, que tiene la tarea de explotar las soluciones, utiliza el operador Incremento teniendo la posibilidad de explorar otras topologías.

4. *Cruce 1 + Movimiento y Cruce 2 + Incremento + Decremento (P4)* Este enfoque es el que más especializa a nivel conceptual puesto que las tareas de explotación y exploración del espacio solución están totalmente divididas. Por una parte, la isla que usa el Cruce 1 explota las soluciones buscando las diferentes combinaciones de neuronas, teniendo la posibilidad de optimizar los parámetros de cada una mediante los operadores de Movimiento. En la otra isla se tiene al Cruce 2 y al operador Incremento buscando de modo incremental la mejor topología, con la posibilidad de seguir explorando otras alternativas gracias al operador Decremento.

Para comparar los algoritmos se ha diseñado una función bidimensional generada mediante una RBFNN cuyos parámetros fueron obtenidos aleatoriamente (Tabla 5.1).

Cada uno de estos algoritmos proporcionará un conjunto no dominado compuesto por RBFNN. Aunque se está tratando de resolver el problema del diseño de una RBFNN, en esta subsección se está analizando el efecto del paralelismo a nivel del algoritmo multiobjetivo, por tanto, todas las RBFNN proporcionadas por cada algoritmo serán comparadas. Aunque existen diversas métricas para comparar conjuntos no dominados (Knowles and Corne, 2002), en (van Veldhuizen et al., 2003) se sugiere:

"... Siempre y cuando la Pareto frontera pueda ser visualizada,

centros		radios	pesos
0.7181	0.8162	0.2237	0.4996
0.5692	0.9771	0.0715	2.4712
0.4608	0.2219	0.0628	0.4887
0.4453	0.7037	0.2332	-0.7052
0.0877	0.5221	0.0327	-0.2006
0.4435	0.9329	0.2352	-0.8020
0.3663	0.7134	0.1755	1.2668
0.3025	0.2280	0.2119	-0.5123
0.8518	0.4496	0.0523	1.0884
0.7595	0.1722	0.1138	-0.5318
0.9498	0.9688	0.0203	2.0797
0.5579	0.3557	0.2237	0.5002
0.0142	0.0490	0.0715	2.2935
0.5962	0.7553	0.0628	-0.8135

Tab. 5.1: Parámetros de la función G_2

el mejor método para evaluar la efectividad de un pMOEA es el ojo humano..."

Dado que las soluciones que proporcionan los algoritmos son pares de números que se pueden visualizar gráficamente, se ha realizado un análisis en base a las gráficas, evaluando el conjunto de soluciones completo.

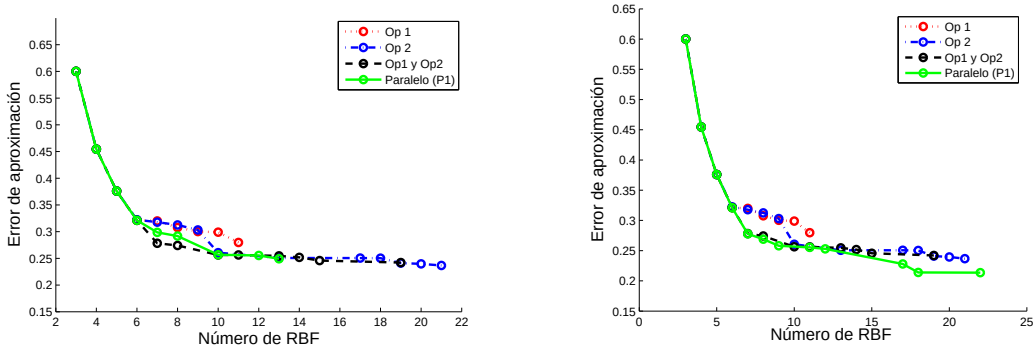
La población inicial fue exactamente la misma para todos los algoritmos, utilizando los mismos valores para todos los parámetros del algoritmo genético. La probabilidad de cruce fue fijada en 0.5, la de mutación 0.25, el número de individuos era de 100 y el número de generaciones fue fijado a 100. El valor de los parámetros fue escogido siguiendo el criterio de no perder diversidad debido a un tamaño de la población reducido así como iterar sobre la población un número adecuado de veces para poder observar la evolución de los individuos en base a las migraciones. La probabilidad de mutación es alta

[0] *pMOEA* significa algoritmos evolutivos multiobjetivo paralelos (*parallel MultiObjective Evolutionary Algorithms*)

pero, dadas las características del experimento, es necesaria para poder explorar/explotar las soluciones en base a la especialización de los operadores de mutación. En cuanto a las migraciones de los individuos en las implementaciones distribuidas, la política de reemplazo se adaptó al proceder del algoritmo NSGA-II de modo que los individuos recibidos se introducen en la unión de los padres con los descendientes a partir de la cual se van obteniendo los Paretos no dominados y se va construyendo la nueva población. Para intercambiar individuos entre las islas se realiza una migración cada cierto número de generaciones. En concreto, se han establecido los siguientes números de generaciones que han de transcurrir entre migración y migración: 40,20,10 y 5. Esto, para el número de generaciones total fijado, proporciona 2,4,9 y 19 migraciones por ejecución respectivamente.

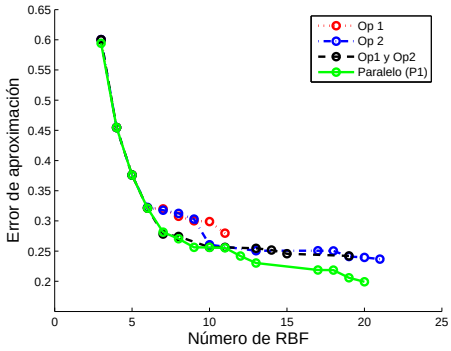
En primer lugar se compara la implementación P1 con respecto a los algoritmos no distribuidos. Los algoritmos no distribuidos son aquellos que únicamente mantienen una población sobre la cual aplican los mismos operadores. Se han definido 3 algoritmos no distribuidos diferenciados por el operador de cruce: uno utiliza el Cruce 1, otro el Cruce 2 y el último aplica el Cruce 1 o el Cruce 2 con igual probabilidad. En cuanto a los operadores de mutación, todos los enfoques no distribuidos utilizan los 4 descritos anteriormente. En la Figura 5.3 se muestran los Paretos obtenidos tras las ejecuciones de los algoritmos no distribuidos y de P1. Como se puede apreciar, el enfoque paralelo es capaz de diseñar RBFNN más precisas, explorando mayor número de topologías que los otros algoritmos no distribuidos.

Tras haber observado cómo una especialización a nivel de operador de cruce puede mejorar la calidad de los resultados de un algoritmo multiobjetivo básico para el diseño de RBFNN, ahora se comparan los distintos enfoques distribuidos descritos anteriormente para ver si la especialización a

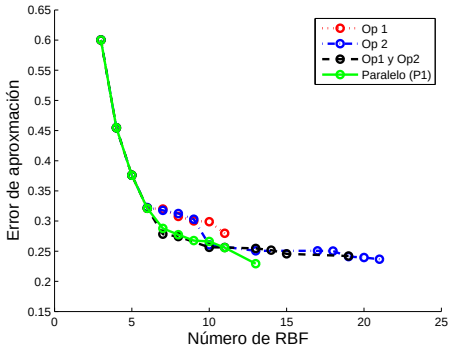


2 migraciones

4 migraciones

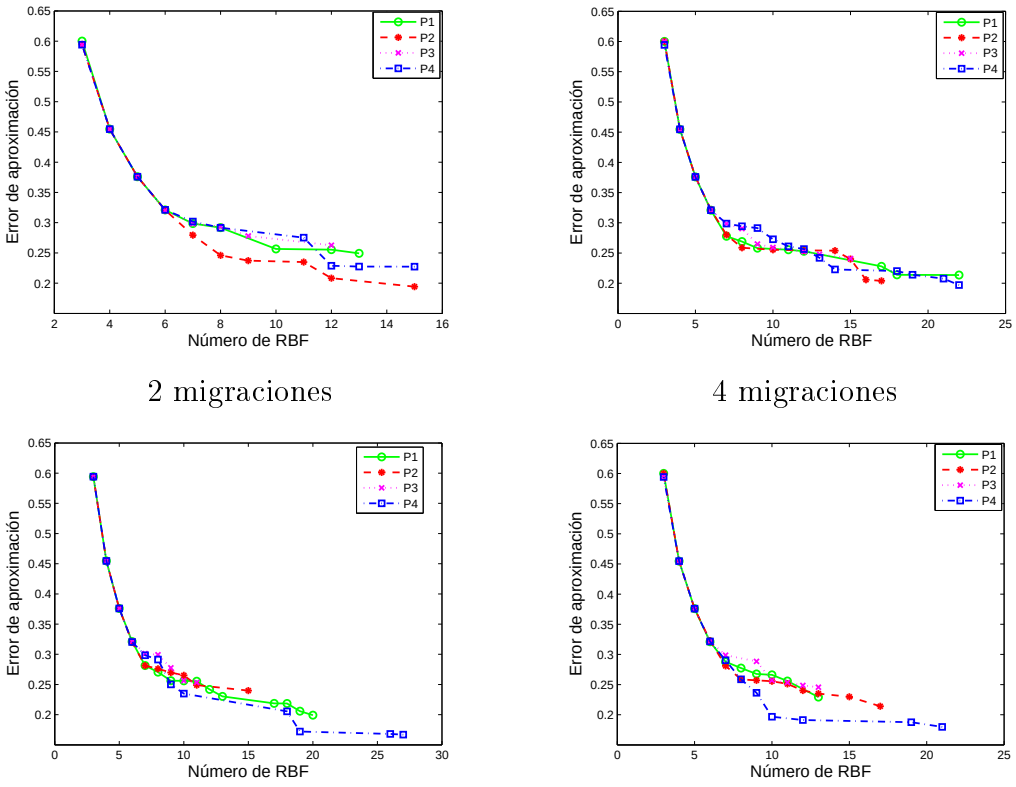


9 migraciones



19 migraciones

Fig. 5.3: Paretos frontera obtenidos tras ejecutar P1 con distintos pasos de migración y los enfoques no distribuidos



2 migraciones 4 migraciones

9 migraciones 19 migraciones

Fig. 5.4: Paretos frontera obtenidos tras ejecutar P1, P2, P3 y P4 con distintos pasos de migración

nivel de mutación puede aportar beneficios. En la Figura 5.4 se muestran los resultados obtenidos tras la ejecuciones de los algoritmos P1,P2,P3 y P4.

La implementación P3 se muestra como la peor alternativa en comparación con el resto de implementaciones, lo cual no es de extrañar puesto que en este caso, la especialización de los operadores de mutación contrarresta los objetivos de los operadores de cruce. En concreto, el Cruce 2 es contrarrestado por el operador de Decremento o por Movimiento, más adecuado para la tarea de explotación. Al mismo tiempo, el Cruce 1, encargado de explotar las distintas topologías es contrarrestado por el operador Incremento.

Las mejores combinaciones de operadores son las proporcionadas por P2 y P4. En P4, el grado de especialización es el máximo posible dividiendo la explotación de la exploración tanto a nivel de operadores de cruce como de mutación. En P2 la especialización también es muy alta dado que la primera isla se encarga de optimizar los parámetros de las RBF y la segunda de explorar nuevas topologías aunque en este caso, en la primera isla es posible disminuir el número de neuronas permitiendo explotar más aún las RBF restantes. Como en P2 se realiza una explotación de las RBF de una manera exhaustiva los resultados mejoran conforme las migraciones no se realizan con excesiva frecuencia dado que primero se explotan las redes existentes obteniendo redes pequeñas pero con neuronas bien evolucionadas y al mismo tiempo en la otra isla solo se explora en la dirección de obtener redes más grandes así que, pasado cierto tiempo, estos individuos se intercambian y una isla puede sacar provecho de lo realizado por la otra.

En el caso de P4, se obtienen mejores resultados cuando las migraciones son más frecuentes. La razón de este hecho es que la isla que evoluciona la estructura de la red, puede incrementar y disminuir el tamaño de ésta con el inconveniente de no haber explotado una solución lo suficiente como para

poder determinar si los parámetros de la red son prometedores. Por tanto, necesita una comunicación más frecuente con la isla que sí determina los parámetros de las RBF para que pueda indicarle si las redes que disminuyen de tamaño son prometedoras.

En general podemos obtener la siguiente conclusión: puesto que el problema del diseño de una RBFNN es una tarea de gran complejidad, relegar la optimización de todos los parámetros a un sólo operador de cruce y de mutación requeriría un gran número de evaluaciones para poder obtener algún resultado de calidad. Al dividir, aunque de una manera muy primitiva, las subtareas del diseño de las redes se han obtenido mejores resultados realizando el mismo número generaciones. Es lógico pensar que, conforme aumente la complejidad del problema, si se pueden especializar distintas islas en evolucionar los distintos aspectos del problema, será posible obtener mejores soluciones. En la sección 5.3 se muestra la ejecución del algoritmo paralelo propuesto en la sección anterior y que ha sido diseñado en base a esta filosofía.

5.3. Beneficios del paralelismo de datos

Como ha mostrado la sección 5.2, el hecho de dividir la población en subpoblaciones y evolucionarlas de un modo especializado, puede proporcionar unos resultados mejores que al tratar a toda la población de forma equitativa y conjunta. Esta sección analizará cómo la división en subpoblaciones implica una reducción en el tiempo de computación sin tener que sacrificar la calidad de los resultados.

El paralelismo funcional se corresponde con el que realiza distintas funciones en paralelo, de modo que la evolución de distintos aspectos de una RBFNN en distintas islas puede considerarse paralelismo funcional. El paralelis-

mo de datos implica que se realice la misma operación sobre distintos datos en paralelo. Si se considera una única población como un conjunto de datos, se puede dividir en subconjuntos y realizar las mismas operaciones sobre ellos, obteniendo así paralelismo a nivel de datos. Por tanto, se puede considerar el algoritmo completo, con sus tres islas, como una operación de procesamiento sobre una población, la cual puede ser dividida en subpoblaciones que sean a su vez procesadas por otros grupos de 3 islas.

Partiendo de esta situación, se ha definido un tamaño de población igual a 300 que se ha ido descomponiendo en subpoblaciones, las cuales, han sido procesadas del mismo modo, es decir, en subgrupos de 3 islas tal y como se muestra en la Figura 5.5. Esta descomposición de la población original en subpoblaciones cada vez más pequeñas hacen que el algoritmo pueda realizar un paralelismo de grano fino en lo que al procesamiento de los datos se refiere.

La función objetivo, mostrada en la Figura 5.6, se ha generado del mismo modo que se hizo con las funciones sintéticas del capítulo anterior. Del mismo modo, también se han añadido unas coordenadas a los vectores de entrada mediante:

$$
\begin{aligned}
X_1^{3s} &= rand \\
X_2^{3s} &= rand \cdot 0{,}5. * X_1 \\
X_3^{3s} &= X_2 \\
X_4^{3s} &= rand \cdot 0{,}25 \cdot X_2 \\
X_5^{3s} &= X_1
\end{aligned}
$$

Los resultados obtenidos tras las ejecuciones se muestran en las Tablas 5.2 y 5.3. Se puede apreciar cómo en todas las ejecuciones el algoritmo fue capaz de identificar correctamente las variables originales que definen la función objetivo. Al observar los errores de aproximación, el algoritmo muestra cómo es capaz de diseñar redes que aproximan con gran precisión la función objetivo. Esto muestra que la capacidad de explotación y exploración no se

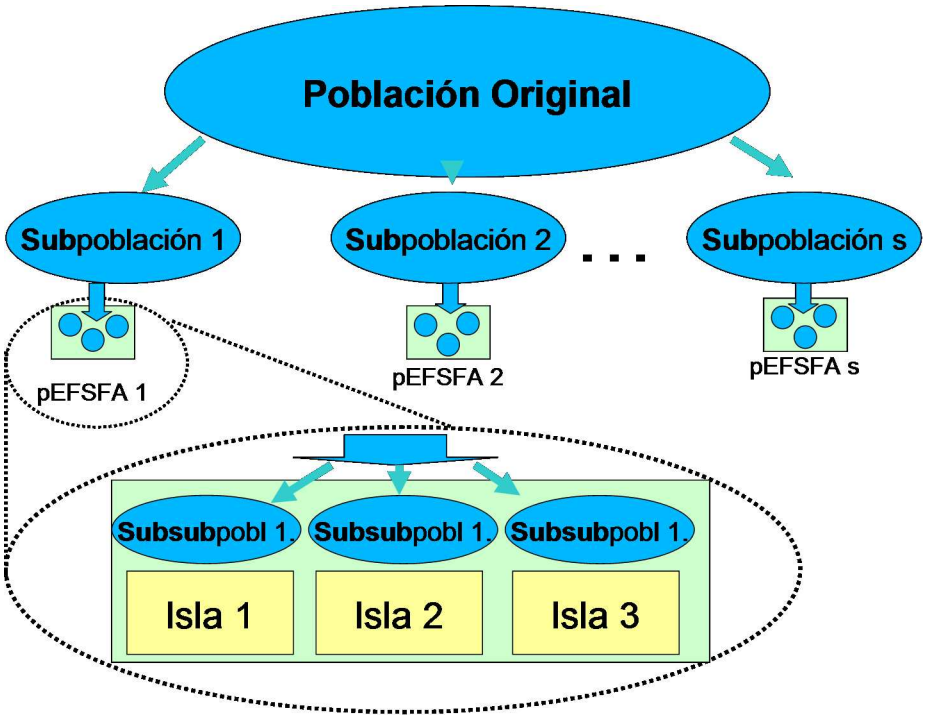


Fig. 5.5: Proceso de obtención del paralelismo a nivel de datos

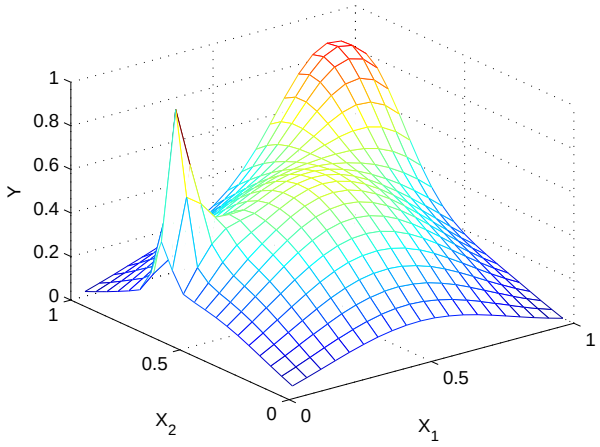


Fig. 5.6: Función objetivo G_5.

Procesadores	Entrenamiento	Test	$\overline{RBF}$
3	0.0255(0.0220)	0.0267(0.0230)	4.6 (1.15)
6	5.18e-4(2.54e-4)	6.0534e-4(2.75e-4)	5.6 (0.57)
9	0.0553(0.0148)	0.0792(0.0340)	3.3 (0.57)
12	0.0154(0.0260)	0.0276(0.0468)	5.3 (1.15)
15	4.87e-4(7.95e-5)	6.09e-4(6.8201e-5)	6 (0)
30	0.0258(0.0215)	0.0270(0.0224)	4.3 (0.57)

Tab. 5.2: Número medio de RBF, media de los errores de aproximación y desviación estándar (en paréntesis) para los conjuntos de test y training de la función G_5 tras 10 ejecuciones usando distinto número de procesadores.

Procesadores	X_1^{3s}	X_2^{3s}	X_3^{3s}	X_4^{3s}	X_5^{3s}
3	0 %	0 %	100 %	0 %	100 %
6	0 %	0 %	100 %	0 %	100 %
9	0 %	0 %	100 %	0 %	100 %
12	0 %	0 %	100 %	0 %	100 %
15	0 %	0 %	100 %	0 %	100 %
30	0 %	0 %	100 %	0 %	100 %

Tab. 5.3: Variables seleccionadas para la función G_5.

ve reducida significativamente al hacer una descomposición de grano fino de la población siempre y cuando se mantenga una topología donde todas las islas se comunican entre sí.

Como cabe esperar, al tener que procesar menos individuos en cada isla, el tiempo de computación total se ve reducido conforme aumenta el número de procesadores. En la Tabla 5.4 y Figura 5.7 se muestran los tiempos de ejecución medidos en las distintas ejecuciones y en la Figura 5.8 la ganancia en velocidad con respecto a la primera ejecución con 3 procesadores.

	Tiempo de ejecución
3 Proc.	2620(117.21)
6 Proc.	809.5(5.29)
9 Proc.	504.3(21.59)
12 Proc.	346.3(20.55)
15 Proc.	281.6(25.73)
30 Proc.	165.2(21.07)

Tab. 5.4: Medias, en segundos, de los tiempos de ejecución y desviación estándar (en paréntesis) medidos para la función G_5.

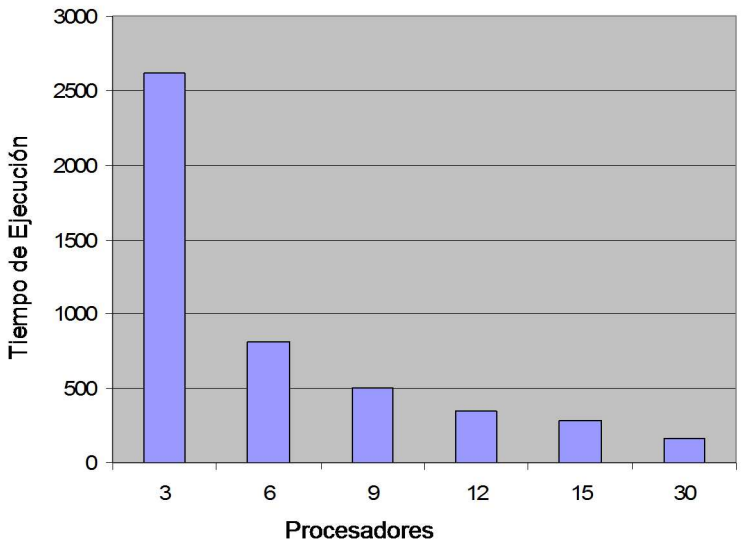


Fig. 5.7: Tiempos de ejecución medidos en segundos para la función G_5.

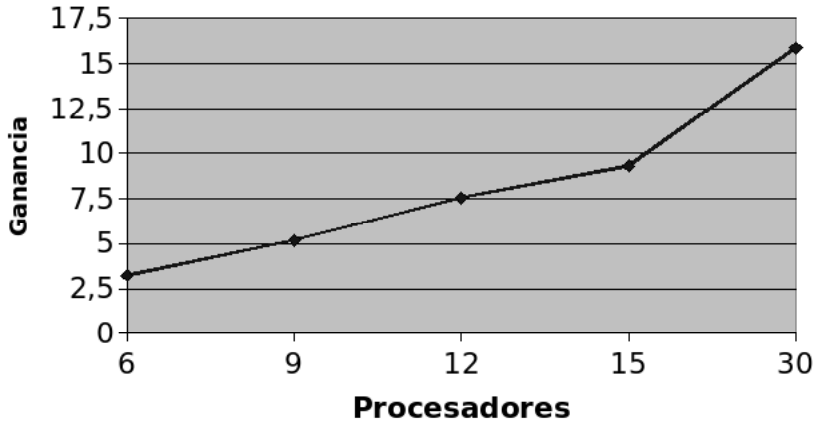


Fig. 5.8: Ganancia en velocidad obtenida al incrementar el número de procesadores para G_5.

5.4. Implementación del paralelismo

Además de todo el trabajo realizado a nivel de diseño del algoritmo, también se ha desarrollado una nueva interfaz para poder ejecutar la aplicación en diversas plataformas de computo sin problemas de portabilidad.

Todos los programas han sido escritos utilizando MATLAB. Este programa posee archivos binarios para poder ser ejecutado en todas las plataformas más comunes: Windows, UNIX y MacOS. La cuestión es que MATLAB no proporciona un soporte para que las aplicaciones desarrolladas sean capaces de invocar funciones de bibliotecas de paso de mensajes. Aunque se desarrolló una *Toolbox* en (Fernández et al., 2006) que permite a MATLAB llamar a funciones MPI, está restringida por la arquitectura sobre la que se ejecuta. Por tanto, para poder ejecutar los programas desarrollados en distintas plataformas se ha implementado una interfaz mediante la cual los scripts de MATLAB son capaces de invocar a las rutinas de cualquier librería que siga el estándar MPI.

El uso de esta interfaz junto con la portabilidad proporcionada por MAT-

LAB hacen que el algoritmo pueda ser ejecutado en cualquier plataforma para la que MATLAB disponga de una implementación y para cualquier biblioteca de paso de mensajes que siga el estándar MPI.

5.4.1. MATLAB Compiler

El programa MATLAB dispone de una herramienta denominada *Compiler* que permite crear aplicaciones ejecutables independientes de MATLAB. Una vez que la aplicación ha sido empaquetada y se han creado sus archivos de interfaz podrá ser ejecutada utilizando la MATLAB *Component Runtime* (MCR). MCR es una componente formada por un conjunto de bibliotecas que permiten la ejecución de los archivos .m obteniendo la misma funcionalidad que MATLAB cuando está procesando un archivo.

El MATLAB *Compiler* utiliza un archivo de componente tecnológica (*Component Technology File* (CTF)) en el que se almacenan y comprimen todos los archivos que forman parte de la aplicación. Toda la información incluida en el archivo CTF se encripta de modo que no es posible conocer el contenido de los archivos .m de la aplicación original. Una vez descomprimida la aplicación en su primera ejecución, los archivos .m serán visibles aunque su contenido seguirá estando encriptado.

El proceso para generar una aplicación ejecutable independiente de una instancia de MATLAB se hace de manera automática y transparente al usuario de modo que éste solo ha especificar qué archivos .m desea incluir en la aplicación. Tras esto, el *Compiler* se encarga de realizar las siguientes operaciones:

- Análisis de dependencias entre los archivos .m y mex de la aplicación.
- Generación de código: se genera el código interfaz en C o C++ que se

utilizará a la hora de lanzar el MCR.

- Creación de archivos: Una vez resueltas las dependencias, el *Compiler* crea el archivo CTF incluyendo todos los archivos .m y mex que sean referenciados para poder invocarlos en tiempo de ejecución. Todos los archivos incluidos son encriptados y se almacenan siguiendo una estructura determinada de directorios.

- Compilación: se genera el código objetivo de los archivos de interfaz.

- Enlazado: se enlaza el código objeto generado con las bibliotecas de MATLAB necesarias.

Todo este proceso se encuentra ilustrado en la Figura 5.9.

Los ficheros de interfaz, también conocidos como envoltorios se encargan de crear los archivos binarios específicos de la máquina donde se ejecutará la aplicación. Estos archivos permiten que se interpreten los archivos .m de la aplicación MATLAB desarrollada.

5.4.2. MPIMEX: Una nueva interfaz MPI

En (Fernández et al., 2006) se presentó la *Message Passing Interface ToolBox* (MPITB) que permite a MATLAB hacer uso de las funciones de paso de mensajes. Sin embargo, MPITB fue compilada únicamente para una de las implementaciones del estándar MPI: LAM/MPI y para arquitecturas x86 ejecutando como sistema operativo Linux.

Ante la oportunidad de trabajar con máquinas del Centro de Computación Paralela de la Universidad de Edimburgo (EPCC), se planteó el problema de poder generar una aplicación independiente mediante el *Compiler* que pudiera utilizar la implementación del estándar de MPI de Sun Microsystem

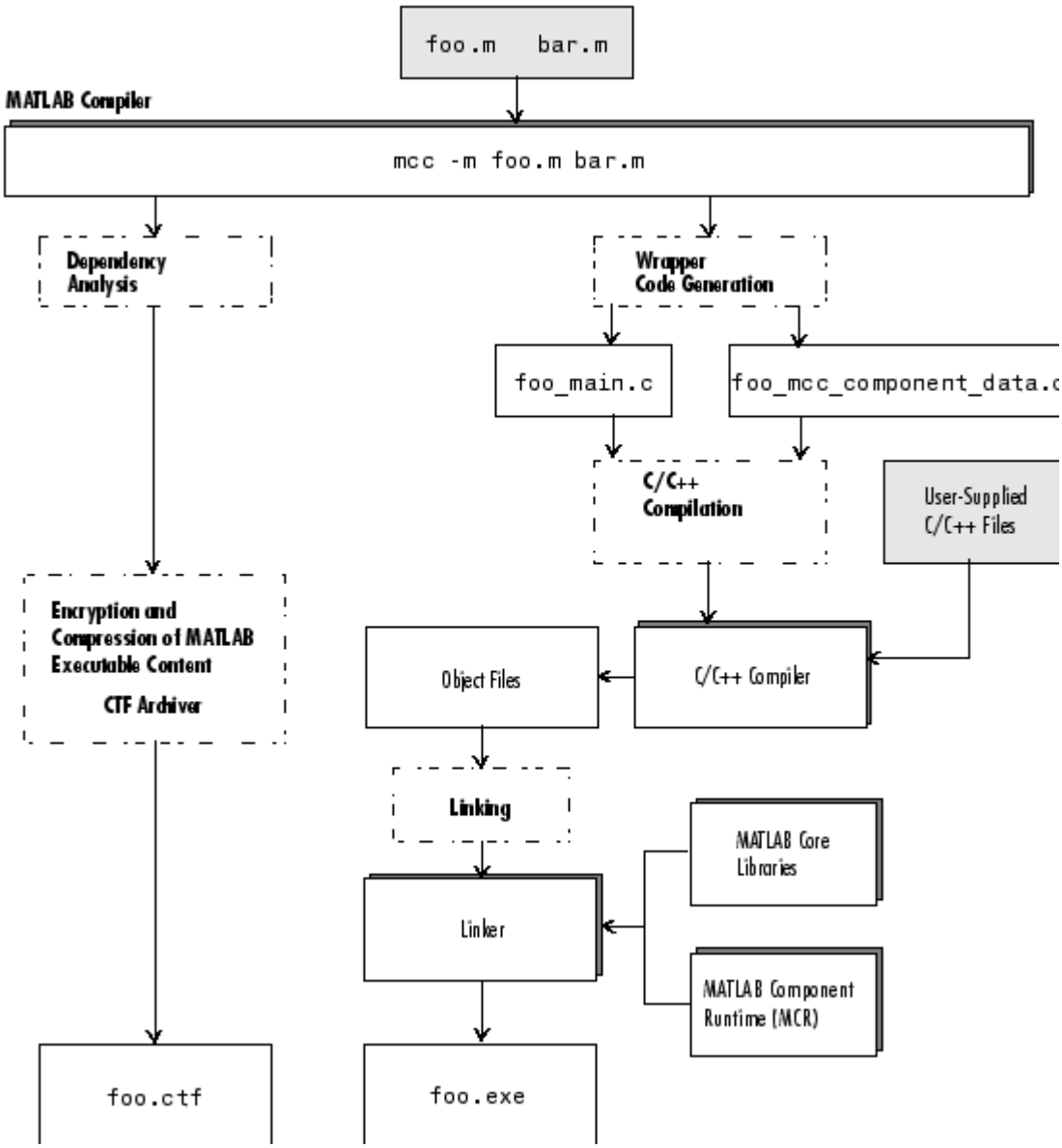


Fig. 5.9: Proceso de generación de una aplicación independiente de MATLAB mediante el MATLAB *Compiler*.

y ejecutarse correctamente. Para ello era necesario implementar una interfaz que permitiera invocar funciones del estándar MPI desde MATLAB independientemente de la plataforma en la que se trabajara.

MATLAB proporciona un método para incorporar código escrito en C/C++ o FORTRAN de modo que se pueda ejecutar en el interprete de comandos de MATLAB y, por tanto, en una aplicación generada por el *Compiler*. Se denominan archivos mex a aquellos que contienen subrutinas que pueden ser enlazadas y ejecutadas dinamicamente por MATLAB. El motivo para incorporar estos archivos es poder reutilizar código ya implementado y ejecutar código de bajo nivel.

Un archivo mex se comporta exactamente igual que las funciones incorporadas que posee MATLAB. A diferencia de los archivos .m que interpreta MATLAB y que son independientes de la plataforma, los archivos mex, al ser compilados a partir de otro lenguaje, son dependientes de la plataforma y poseen las siguientes extensiones:

Plataforma	**Extension MEX-File**
Linux (32-bit)	mexglx
Linux x86-64	mexa64
Macintosh (PPC)	mexmac
Macintosh (Intel)	mexmaci
32-bit Solaris SPARC	mexsol
64-bit Solaris SPARC	mexs64
Windows (32-bit)	mexw32
Windows x64	mexw64

Una vez escrita y compilada un función en su correspondiente archivo .mex, puede invocarse desde la línea de comandos igual que se llama una función contenida en un archivo .m.

La interfaz se implementa en un fichero denominado MPI.c que posteriormente es compilado por MATLAB para ser añadido al paquete en el cual

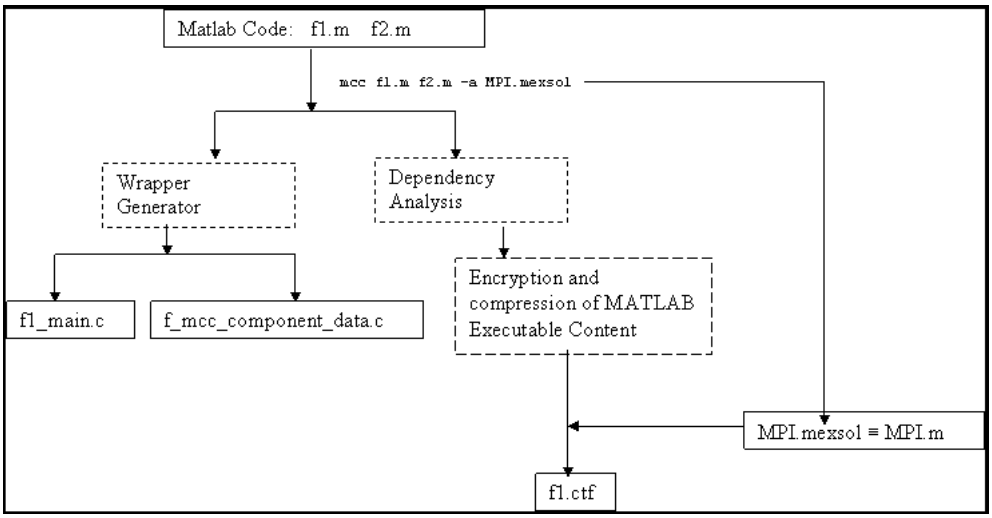


Fig. 5.10: Obtención de aplicaciones ejecutables desarrolladas con MATLAB y MPI utilizando la interfaz desarrollada

se incluyen todas las funciones de las *Toolboxes* que proporciona MATLAB. El procedimiento para generar la aplicación está representado en la Figura 5.10.

5.4.2.1. Ganancia en prestaciones

La portabilidad e independencia de la plataforma no es la única ventaja que proporciona la nueva interfaz desarrollada en comparación con otras *Toolboxes* para paso de mensajes en MATLAB. Al comparar entre MPITB y la interfaz desarrollada en esta memoria, se puede observar cómo el tiempo de latencia entre la llamada a una función de la biblioteca utilizando la interfaz desarrollada en esta memoria es inferior que la desarrollada en (Fernández et al., 2006). Para mostrar este hecho, se analizará el comportamiento de las funciones básicas para realizar la comunicación entre dos procesos: MPI_Send y MPI_Recv cuyas cabeceras según el estándar son:

MPI_Send Realiza un envío básico

int MPI_Send(void*buf, intcount, MPI_Datatypedtype, int dest,
int tag, MPI_Commcomm)

Ej: MPI_Send(message, strlen(message)+1, MPI_CHAR, dest,
tag, MPI_COMM_WORLD);

Entrada:

buf - direcciónde iniciodel buffer a enviar

count - númerode elementosa enviar

dtyp - tipode datode MPI quese vaa enviar

dest - rank del procesode destino

tag - etiqueta

comm - comunicador

Observaciones:

- Esta función realiza un envió bloqueante hasta que el mensaje sea totalmente recibido por el destinatario.

MPI_Recv-Recepción básica

intMPI_Recv(void*buf, intcount, MPI_Datatypedtype, int src, int tag,
MPI_Commcomm, MPI_Status*stat)

Ej: MPI_Recv(message,100,MPI_CHAR, source, tag,
MPI_COMM_WORLD, &status);

Entrada:

count - máximo número de elementos a recibir

dtype - tipo de dato a recibir

src - rankdel origen del mensaje

tag - etiqueta

comm - comunicador

Salida:

buf - dirección de memoria de comienzo del buffer

stat - objeto Status, que puede ser sustituido por la constante de MPI: MPI_STATUS_IGNORE si los datos de estado no interesan.

Observaciones:

- Dentro de las constantes definidas en MPI, existen:

- MPI_ANY_SOURCE Indica que el mensaje a recibir puede venir de cualquier destinatario

- MPI_ANY_TAG Indica que la etiqueta no tiene que ser igual a la del envío.

Se ha implementado una función sencilla en la cual se realiza el envío de un mensaje de un procesador a otro mediante las funciones citadas. Se ha repetido la ejecución de la función 10000 veces midiendo el tiempo transcurrido justo antes de invocar a la función MPI_Send hasta que ésta terminase.

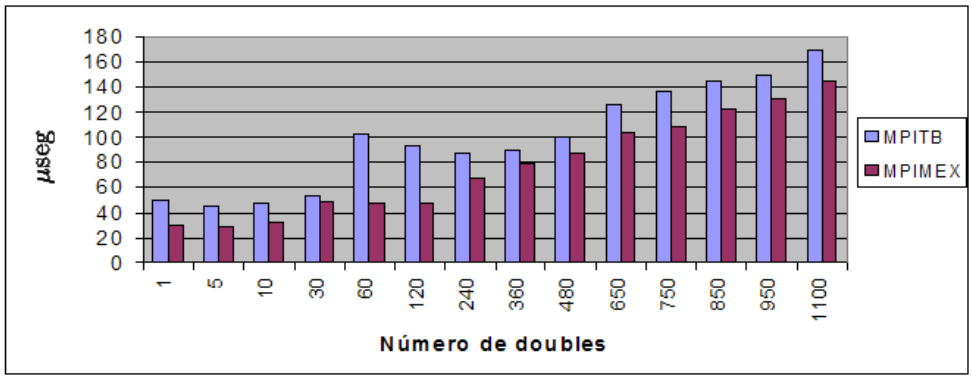


Fig. 5.11: Comparación entre interfaces MPI/MATLAB para la función MPI_Send

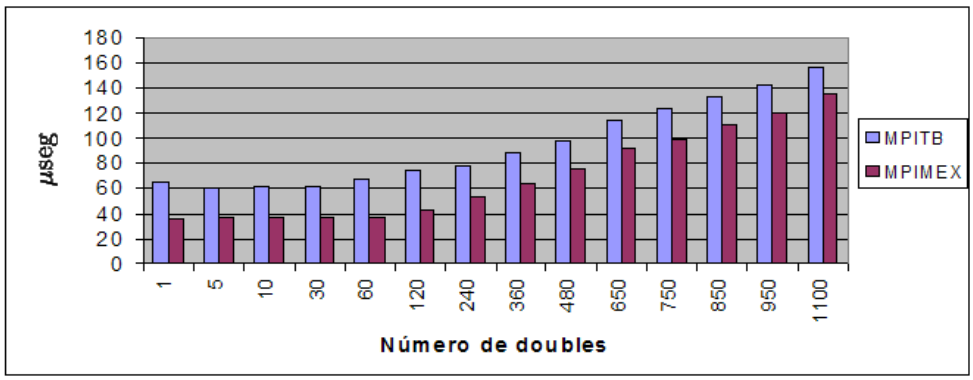


Fig. 5.12: Comparación entre interfaces MPI/MATLAB para la función MPI_Recv

Se ha procedido del mismo modo con la función MPI_Recv que se invocaba desde el otro procesador. Los resultados se muestran en la Tabla 5.5 para MPI_Send y en la Tabla 5.6 para MPI_Recv, los cuales están representados gráficamente en las Figuras 5.11 y 5.12 respectivamente.

Como se puede apreciar, existe un mayor retardo al realizar las comunicaciones cuando se emplea la MPITB que cuando se utiliza la interfaz desarrollada en esta memoria. Esta diferencia viene originada por la simpleza de la nueva interfaz, la cual, solo hace uso de un único fichero mex sin perder por esto la funcionalidad requerida. En los resultados puede apreciarse cómo,

Función MPI_Send		
# paquetes	MPITB	MPIMEX
1	49.85 (168.7)	30.01 (8.2)
5	44.41 (9.8)	28.81 (8.1)
10	47.66 (9.5)	32.33 (10.76)
30	52.69 (7.8)	49.09 (552.1)
60	102.45 (859.7)	46.69 (402.6)
120	93.11 (577.5)	47.52 (281.1)
240	87.92 (409.6)	66.96 (563.1)
360	89.32 (210.3)	79.58 (575.3)
480	100.91 (255.7)	87.32 (417.1)
650	125.09 (461.4)	103.25 (434.2)
750	136.41 (480.1)	109.30 (431.4)
850	145.37 (482.0)	122.20 (444.6)
950	149.08 (326.3)	130.22 (447.8)
1100	169.11 (495.4)	145.61 (449.5)

Tab. 5.5: Medias de los tiempos de ejecución en μseg. y desviación estándar (en paréntesis) para realizar una llamada a MPI_Send variando el número de elementos a enviar.

a medida que el tamaño del mensaje crece, la diferencia se ve reducida. Esto es lógico dado que el tiempo que tarda en transmitirse y recibir el mensaje empieza a ser considerablemente más grande que el tiempo ahorrado en las llamadas a las funciones.

En el ejemplo anterior se ha mostrado el comportamiento de la interfaz en una comunicación síncrona entre un par de procesadores pero la mejora en prestaciones también puede apreciarse en comunicaciones colectivas como, por ejemplo, la función MPI_Bcast, cuya página man la describe como:

Función MPI_Recv		
# paquetes	MPITB	MPIMEX
1	65.63 (178.1)	35.91 (17.4)
5	59.92 (20.2)	36.63 (33.8)
10	62 (154.4)	36.28 (33.4)
30	60.74 (9.1)	37.31 (29.5)
60	66.72 (6.6)	36.64 (7.0)
120	73.93 (57.8)	42.56 (10.2)
240	77.61 (11.4)	52.75 (15.1)
360	88.01 (15.7)	63.98 (20.9)
480	97.71 (19.3)	76.12 (24.4)
650	113.97 (24.3)	92.52 (38.2)
750	124.24 (27.9)	99.35 (38.4)
850	133.23 (35.8)	110.89 (45.9)
950	142.53 (44.3)	120.05 (51.3)
1100	156.00 (50.6)	135.43 (65)

Tab. 5.6: Medias de los tiempos de ejecución en μseg. y desviación estándar (en paréntesis) para realizar una llamada a MPI_Recv variando el número de elementos a enviar.

MPI_Bcast - Envía un mensaje desde el proceso con rank *root* a todos los demás procesos en el comunicador

int MPI_Bcast(void *buff, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Entradas/salida:

buff - dirección de inicio del buffer

count - número de elementos en el buffer

datatype - tipo de dato de los elementos del buffer

root - rank del proceso que envía

comm - comunicador donde se envía el mensaje

Observaciones:

- Si hay menos de 4 procesos se itera sobre un bucle, en caso contrario, se utiliza un algoritmo basado en una estructura de árbol.

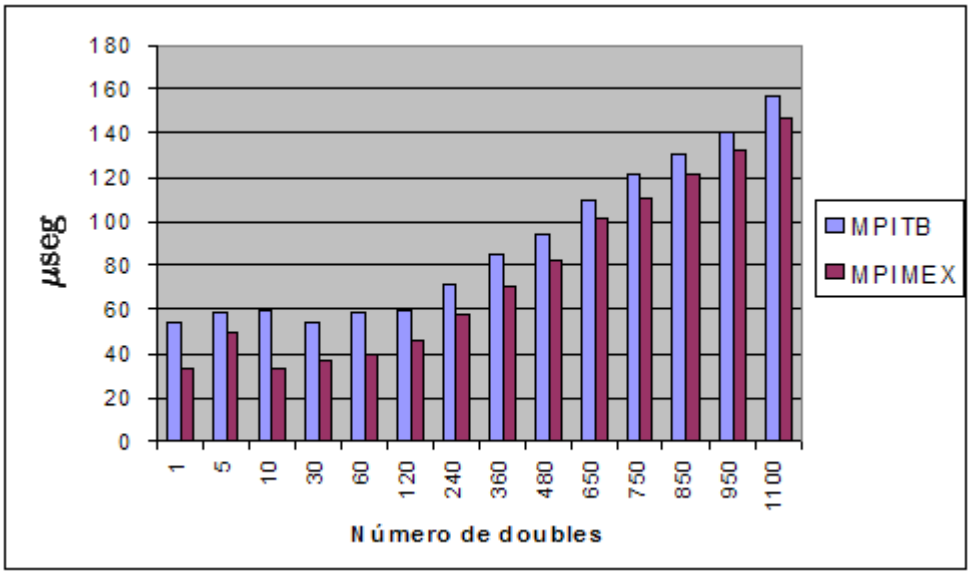


Fig. 5.13: Comparación entre interfaces MPI/MATLAB para la función MPI_Bcast con 2 procesadores (receptor)

Dado que MPI_Bcast es una operación colectiva en la que se ven implicados un conjunto de procesos, además de variar el tamaño del mensaje, también se ha modificado el número de procesadores ejecutando la función. Los resultados para las ejecuciones utilizando 2,3,4 y 5 procesadores se muestran respectivamente en las Tablas 5.7, 5.8 y 5.9 y en la Figuras 5.13,5.14 y 5.15. En la operación colectiva MPI_Bcast, hay un proceso raíz que envía mientras que todos los demás reciben, para cada ejecución se muestran los resultados del proceso raíz y de uno de los destinos, siendo el tiempo entre éstos prácticamente idéntico. Al igual que en la ocasión anterior, siempre se obtiene un tiempo menor al realizar la comunicación cuando se utiliza la nueva interfaz tanto para el proceso que envía el mensaje como para los que lo reciben, independientemente del tamaño del mensaje o del número de procesadores que intervengan en la operación.

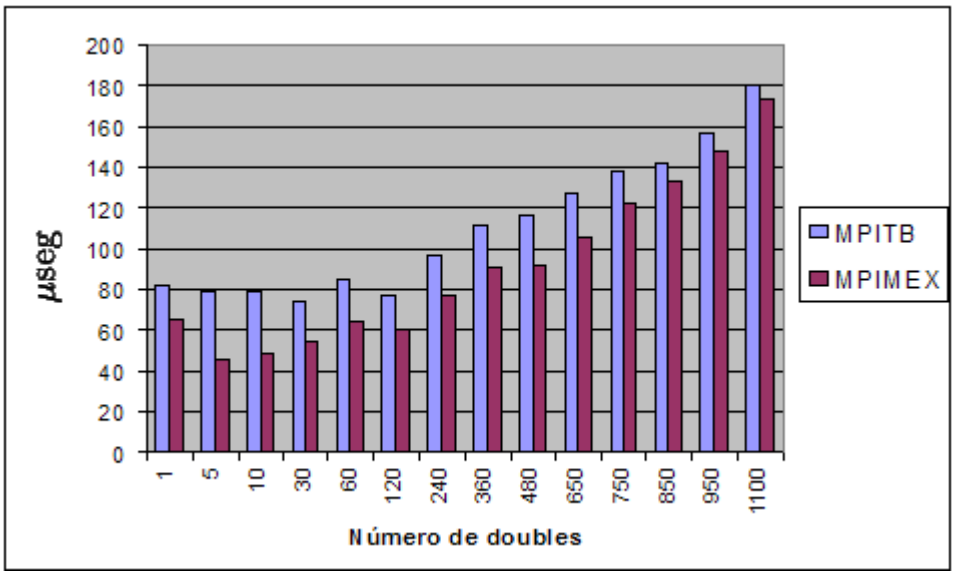


Fig. 5.14: Comparación entre interfaces MPI/MATLAB para la función MPI__Bcast con 3 procesadores (root)

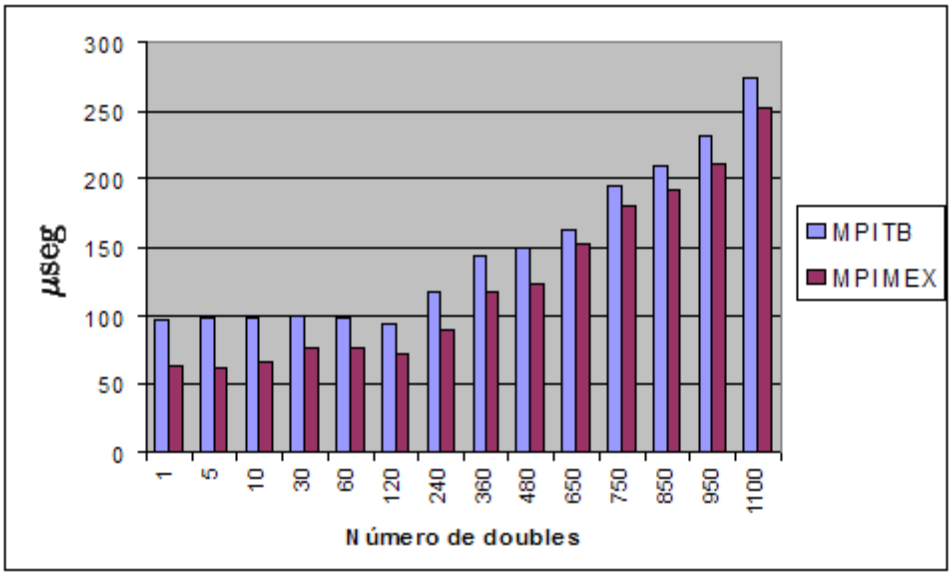


Fig. 5.15: Comparación entre interfaces MPI/MATLAB para la función MPI_Bcast con 4 procesadores (root)

	Función MPI_Bcast			
# paquetes	MPITB		MPIMEX	
1	52.26 (57.2)	53.68 (57.1)	30.74 (4.7)	33.52 (15.1)
5	55.02 (6.4)	59.01 (34.3)	36.58 (40.1)	49.61 (98.2)
10	55.36 (8.6)	59.84 (45.9)	43.93 (602.6)	33.53 (4.4)
30	53.23 (10.8)	54.01 (59.4)	45.80 (400.6)	37.35 (27.8)
60	57.26 (8.3)	59.01 (56.1)	42.89 (240.3)	39.29 (8.1)
120	62.37 (136.5)	59.32 (9.3)	46.71 (189.1)	46.33 (11.5)
240	72.83 (77.8)	71.19 (13.9)	57.48 (149.1)	57.45 (30.1)
360	84.25 (119.1)	84.26 (19.8)	70.50 (136.6)	70.43 (22.5)
480	94.63 (151.6)	94.31 (26.7)	84.81 (190.5)	82.09 (27.1)
650	109.53(183.7)	109.26 (33.4)	101.12 (215.4)	100.89 (39.6)
750	121.65(171.6)	120.77 (37.9)	110.94 (215.1)	110.52 (45.5)
850	131.47(198.8)	130.46 (44.9)	121.39 (230.5)	121.08 (51.3)
950	140.06(170.1)	140.28 (51.1)	131.31 (209.8)	131.83 (59.3)
1100	157.59(186.8)	156.68 (64.6)	146.91 (239.5)	146.52 (66.4)

Tab. 5.7: Medias de los tiempos de ejecución en μseg. y desviación estándar (en paréntesis) para realizar una llamada a MPI_Bcast variando el número de elementos a enviar y 2 procesadores.

5.4.3. Plataforma paralela utilizada

A continuación se describe la máquina sobre la cual se han realizado todos los experimentos. Esta máquina está emplazada en el EPCC de la Universidad de Edimburgo y está disponible para todos los investigadores europeos que participen en el programa HPC-Europa, que está financiado gracias a la Unión Euoropea y que permite la movilidad de investigadores europeos a través de las más distinguidas instituciones en el campo de la computación paralela.

5.4.3.1. SunFire E15K

El servidor Sun Fire E15K posee elementos hardware redundantes para obtener una máxima disponibilidad y robustez frente a fallos. Puede llegar

	Función MPI_Bcast			
# paquetes	MPITB		MPIMEX	
1	82.29 (58.7)	82.10 (57.3)	64.73 (3.4)	66.53 (11.3)
5	78.70 (10.8)	80.26 (17.7)	45.56 (10.3)	54.90 (17.1)
10	79.09 (10.3)	85.26 (13.3)	48.17 (7.3)	59.79 (15.9)
30	74.71 (9.8)	85.50 (11.9)	54.62 (33.9)	66.36 (10.5)
60	85.04 (3.4)	85.29 (13.3)	64.45 (3.6)	68.02 (11.8)
120	77.03 (8.8)	77.28 (15.2)	59.81 (3.4)	60.41 (13.4)
240	96.39 (129.4)	96.30 (17.2)	77.21 (3.4)	78.43 (17.9)
360	111.22 (12.7)	114.34 (23.9)	91.04 (12.7)	96.02 (26.3)
480	116.21 (7.1)	117.29 (21.2)	92.02 (109.8)	99.84 (25.8)
650	126.56 (15.7)	129.70 (40.1)	105.65 (95.8)	106.97 (29.4)
750	138.02 (16.9)	142.60 (29.7)	123.02 (101.1)	128.74 (26.1)
850	141.78 (14.4)	147.02 (29.6)	133.31 (16.1)	139.57 (28.7)
950	156.24 (16.4)	162.64 (33.4)	147.40 (14.9)	154.40 (32.9)
1100	180.69 (18.1)	186.26 (33.3)	172.94 (16.6)	181.73 (34.8)

Tab. 5.8: Medias de los tiempos de ejecución en μseg. y desviación estándar (en paréntesis) para realizar una llamada a MPI_Bcast variando el número de elementos a enviar y 3 procesadores.

poseer 106 procesadores UltraSPARC® III Cu 1.2 GHz y es capaz de tener una memoria de 1/2 TeraByte por dominio. El ancho de banda del Sun Fire puede llegar a los 172.7 GigaBytes por seggundo a través de la interconexión redundante Sun Fireplane y la funcionalidad *Memory Placement Optimization* (MPO). Es una plataforma muy flexible gracias a la posibilidad de reconfigurarse dinamicamente según demanda.

Especificaciones:

- Procesadores: 52 UltraSPARC® III Cu Superscalar SPARC® V9, ECC; Caché protegida por procesador; Caché de nivel 1: protegida por paridad con una capacidad de 32 KB para instrucciones y 64 KB para datos; Cáché de nivel 2 externa: 8 MB protegida por ECC; Sistema de interconexión: barras cruzadas de 18×18 150 MHz Sun Fireplane.

	Función MPI_Bcast			
# paquetes	MPITB		MPIMEX	
1	97.94 (57.5)	98.27 (54.8)	63.23 (12.6)	78.63 (6.3)
5	98.53 (6.5)	98.96 (13.1)	62.96 (11.8)	69.58 (13)
10	98.56 (7.4)	99.52 (14.1)	65.71 (13.1)	76.52 (10.6)
30	100.44 (7.6)	100.51 (14.5)	76.19 (8.4)	75.58 (12.5)
60	98.82 (8.6)	101.51 (16.1)	76.74 (3.6)	77.38 (5.3)
120	94.15 (9.5)	94.49 (10.2)	72.21 (5.6)	73.11 (8.3)
240	117.61 (11.9)	121.43 (13.5)	90.53 (8.6)	96.62 (7.7)
360	144.13 (21.5)	149.32 (13.4)	117.48 (12.4)	121.75 (18.1)
480	149.85 (14.9)	153.97 (17.3)	122.79 (12.2)	127.58 (15.4)
650	163.48 (12.6)	171.02 (9.6)	151.87 (13.4)	158.08 (10.2)
750	195.67 (21.7)	203.57 (18.1)	180.73 (159.2)	187.05 (12.7)
850	209.51 (15.1)	214.84 (6.4)	192.28 (17.2)	199.67 (7.8)
950	230.84 (17.6)	237.49 (7.7)	211.81 (20.1)	221.63 (12)
1100	273.32 (22.9)	277.83 (10.9)	253.19 (21.5)	262.29 (12.2)

Tab. 5.9: Medias de los tiempos de ejecución en μseg. y desviación estándar (en paréntesis) para realizar una llamada a MPI_Bcast variando el número de elementos a enviar y 4 procesadores.

- El sistema CPU/Memoria llega hasta las 18 *Uniboard* tarjetas, con 4 procesadores cada una y un total de 32GB de memoria por tarjeta; Entrada/Salida (E/S): *hot-swappable* slots PCI, 36 slots tienen una velocidad de 66 MHz; 36 slots de 33 MHz; soporta los siguientes tipos de conexiones: 10/100BaseT ethernet, gigabit ethernet, UltraSCSI (LVD and HVD), ATM, FC-AL, and HiPPI.

- Almacenamiento: UltraSCSI Sun StorEdge 9900, 6900, 3900, T3, A5200, S1, D1000, y dispositivos de cinta.

- Rendimiento: Ancho de banda total del sistema llega a 172.8 GB/sec. Ancho de banda de E/S 21.6 GB/sec. sostenido.

- Dimensiones y peso: Altura 191 cm , Anchura 85 cm, Profundidad 166 cm , Peso totalmente configurado 1122 kg.

Fig. 5.16: SunFire E15K

5.5. Conclusiones

Este capítulo profundiza en el análisis e implementación del algoritmo evolutivo propuesto en el Capítulo 4, centrándose y estudiando con detalle su carácter paralelo. Se ha mostrado cómo el paralelismo funcional proporcionado por un algoritmo genético distribuido heterogéneo, donde cada isla se especializa en un aspecto del diseño, es capaz de mejorar los resultados de un algoritmo genético no distribuido. Además, el paralelismo no proporciona mejores resultados únicamente sino que, al obtenerse un paralelismo de datos, también se reduce considerablemente el tiempo de ejecución.

Una característica muy importante del algoritmo es su escalabilidad ya que, como se ha mostrado, conforme aumenta el número de recursos, se disminuye el tiempo de ejecución pero no la calidad de los resultados como podría ocurrir al disminuir el número de individuos por isla.

Por último, se ha descrito uno de los elementos que permiten la ejecución

de algoritmos paralelos desarrollados en MATLAB que fue desarrollado específicamente para poder implementar el algoritmo propuesto aunque puede ser utilizado por cualquier programa MATLAB que desee realizar llamadas a la biblioteca de funciones MPI. Se ha comparado la nueva interfaz desarrollada con otra interfaz existente y se ha mostrado cómo la nueva interfaz puede proporcionar menos latencia de comunicación.

6. APLICACIÓN DEL ALGORITMO DESARROLLADO A DIVERSOS PROBLEMAS DEL MUNDO REAL

Este capítulo muestra cómo el algoritmo propuesto a lo largo de la memoria puede ser de utilidad en problemas reales. El primer problema abordado consiste en el cálculo de la cantidad de combustible necesaria para llevar a cabo el proceso de reducción de mineral para obtener Níquel. El segundo problema consiste en la predicción del peso que tendrá un recién nacido, con el objetivo de prever ciertas maniobras y complicaciones que pueden producirse durante el parto.

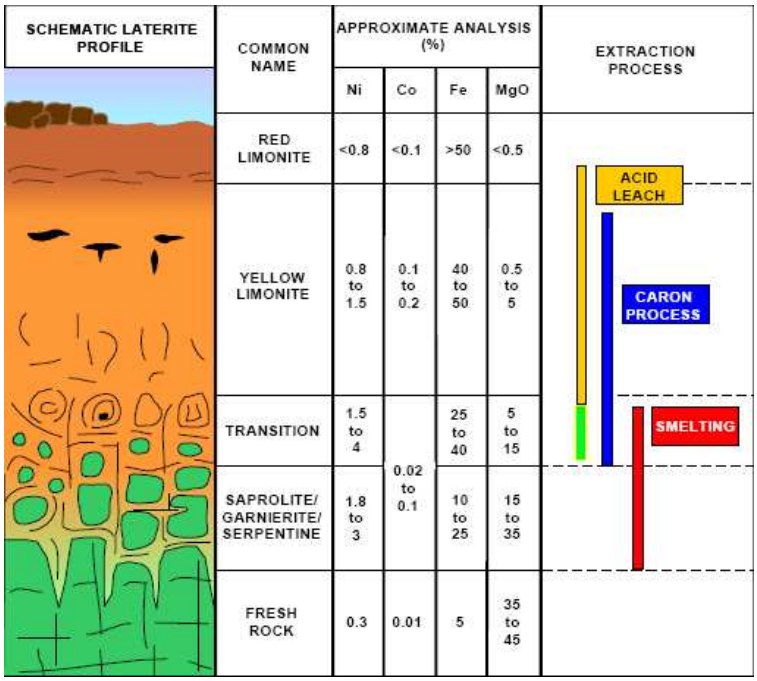


Fig. 6.1: Distribución del Niquel en la Naturaleza

6.1. Aplicación 1: Proceso de Reducción del Mineral

El níquel se encuentra en la naturaleza formando silicatos, óxidos, sulfuros, sulfatos, etc. Igual que muchos otros minerales; puede ser explotado a cielo abierto y subterráneo. Es un mineral de gran demanda en la industria, principalmente para la obtención de aceros de gran calidad y en muchísimas aleaciones con Cobre, Cromo, Aluminio, Plomo, Cobalto, Manganeso, Plata y Oro. Es, después del manganeso, el metal más usado en ferroaleaciones. Comúnmente se comercializa en forma de lingotes, municiones, pellets y polvo, así como en forma de óxido conteniendo de 75 a 90 % de níquel.

El níquel es un elemento bastante abundante que constituye cerca de 0.008 % de la corteza terrestre y 0.01 % de las rocas ígneas, se presenta además en pequeñas cantidades en plantas, animales, en el agua de mar, el petróleo y en la mayor parte del carbón 6.1.

Cuba posee parte de los yacimientos niquelíferos más importantes del mundo, los cuales representan aproximadamente el 37.3 % de las reservas mundiales. El grueso de la producción de níquel en Cuba pertenece al grupo de Clase II, éstos se caracterizan por poseer un contenido de níquel muy amplio de hasta 99.8 %, lo que resulta muy importante en la fabricación de aceros inoxidables, ello explica porqué este tipo de producto cubre el 60 % del mercado internacional de este mineral.

Desde el punto de vista tecnológico existen en Cuba dos procesos de obtención de Níquel: Lixiviación Carbonato Amoniacal o Proceso CARON y Lixiviación Ácida.

Estos procesos proporcionan las siguientes ventajas:

1. Es un proceso que se realiza en condiciones de presión atmosférica.
2. El equipamiento tecnológico del proceso se distingue por su sencillez y amplia utilización de los aparatos conocidos (hornos de soleras múltiples, espesadores, columnas de destilación, etc.).
3. El esquema amoniacal permite las mezclas de los minerales lateríticos y serpentínicos , mientras que el esquema de lixiviación con ácido sulfúrico permite solamente la elaboración de la fracción laterítica.

En cuanto a las desventajas destacan:

1. El bajo porcentaje de extracción: 75-76 % de Ni y 25-30 % de Co.
2. El alto consumo de portadores energéticos: electricidad y fuel oil.

6.1.1. Proceso Industrial

El proceso industrial desarrollado en la Planta de Punta Gorda engloba las siguientes actividades:

6.1.1.1. Mina

Suministra el mineral a la fábrica. El mineral de la capa superior está compuesto de Limonita y el cuerpo de Laterita y Serpentina (blanda). Los componentes fundamentales del mineral son el Níquel, el Cobalto y como acompañante en cantidades considerables el Hierro.

6.1.1.2. Planta de Preparación del Mineral

En esta planta el mineral se somete a un proceso de secado y molienda y se suministra a los silos de almacenaje, de donde se bombea a los hornos de reducción.

6.1.1.3. Planta de Hornos de Reducción

En esta planta se realiza el proceso de reducción del Ni contenido en el mineral. Está constituida por 24 hornos de soleras múltiples que descargan el mineral a los enfriadores, de donde pasa a la planta de lixiviación y lavado.

6.1.1.4. Planta de Lixiviación y Lavado

La pulpa de mineral reducido pasa por 3 sistemas paralelos de tres etapas de Lixiviación a contracorriente con el licor carbonato - amoniacal. La Lixiviación se realiza con el licor carbonato-amoniacal en los espesadores por medio de la aereación de la pulpa con aire (en los turboareadores). Después de la lixiviación, la pulpa se envía al sistema de lavado (dos en paralelo). El licor enriquecido en Ni y Co es enviado a la planta de separación de Cobalto, la pulpa de desecho es enviada a la planta de recuperación de amoníaco.

6.1.1.5. Planta de Separación de Cobalto

El Licor enriquecido en Ni y Co se somete a una inyección de Hidrosulfuro de Amonio o Sulfhidrato de Sodio para precipitar el Co en forma de sulfuro. El licor descobaltizado se envía a la planta de recuperación de amoníaco.

6.1.1.6. Planta de Recuperación del Amoníaco

El licor carbonato amoniacal enriquecido en Ni recibe un tratamiento con vapor en las torres de destilación obteniéndose el Carbonato Básico de Níquel. La pulpa de desecho de la última etapa de lavado se envía a las torres de destilación de Colas donde recibe tratamiento con vapor para la recuperación del licor amoniacal contenido en esa pulpa. El producto de desecho(cola) es enviado a la presa de Cola. La pulpa de Carbonato de Niquel se envía a la planta de Calcinación y Sinter.

6.1.1.7. Planta de Calcinación y Sinter

Luego de filtrado el carbonato básico de Niquel es alimentado a los hornos de calcinación para la obtención del óxido de Ni, que es utilizado en el proceso de sinterización en las máquinas destinadas para este fin; obteniéndose el óxido Sinter que constituye el producto final de la planta y de la fábrica. Este producto es envasado y trasladado al puerto para su comercialización.

6.1.1.8. Plantas Auxiliares

Central Termoelectrica, Potabilizadora de Agua y Tratamiento Quimico de Agua.

6.1.2. Factores que Influyen en la Operación del Horno

Entre los distintos elementos que pueden modificar el comportamiento del horno mientras está funcionando se encuentran:

- Temperatura: Este es un parámetro fundamental en todo el proceso pirometalúrgico, ya que la temperatura facilita el cambio de estado o el debilitamiento de la estructura cristalina. El perfil de temperatura se mantiene mediante la utilización de quemadores de petróleo que se encuentran en las cámaras de combustión. En éstas se trata de mantener una relación aire-petróleo que garantice la combustión incompleta, a la vez ayuda a enriquecer la atmósfera reductora dentro del horno, además en los hogares 4 y 6 se introduce aire secundario proveniente de las camisas de enfriamiento en los hogares 4 y 6 para quemar el C0 residual. Debido a esta reacción exotérmica se produce una cantidad de calor adicional que contribuye al calentamiento del mineral y a mantener el perfil térmico del horno. Durante la operación se debe mantener un perfil de temperatura aumentando de arriba hacia abajo que garantice un calentamiento gradual para que las pérdidas de agua de los silicatos no sean bruscas y no se afecte la extracción del mismo.

- Granulometría: Por este ser un proceso heterogéneo la granulometría influye determinantemente en los buenos resultados de reducción. Las reacciones ocurren fundamentalmente en la línea divisoria de las fases y la velocidad de la reacción está determinada por la penetración del gas al interior de la partícula. Si estas son pequeñas, aumentara la superficie activa del mineral, será mayor el contacto entre las fases, lo que incide directamente en la conversión de la reacción. En la práctica se trabaja con un 85 % de fracción 0.074mm, con lo que se puede lograr

extracciones aceptables para este proceso sin que el arrastre de polvo sea incrementado sustancialmente.

- Composición de la Materia Prima: La composición química de la materia prima influye directamente en los resultados de la reducción. Atendiendo a esta se fijan las temperaturas en el horno y el régimen de calentamiento de mineral así por ejemplo, cuando el mineral es serpentinoso o sea que su contenido de hierro es bajo, el régimen de temperatura debe ser más alto, así como la concentración de reductores debe ser elevada para reducir el níquel en el momento del debilitamiento de los cristales por la expulsión de agua cristalina. Para el mineral laterítico. Las condiciones de reducción pueden ser menos severas ya que estos minerales se reducen a bajas temperaturas.

- Concentración de los Gases: La reducción se lleva a cabo mediante el contacto de los gases reductores con el mineral. Ambas fases se ponen en contacto en la cama de mineral y las caídas de un hogar a otro. Al encontrarse el petróleo aditivo en hogares con temperaturas superiores a 350°C, comienza la descomposición del mismo, formándose el CO y H2 una vez reaccionado el carbono activo en la primera etapa de la cadena de reacción química, además la atmósfera reductora es enriquecida por el gas producto de la combustión incompleta en los quemadores de petróleo. La concentración de los gases influyen directamente en la conversión del níquel.

- Número de Hogares: Este ha sido un factor de importancia en el desarrollo de los hornos modernos, ya que fue detectado que el número de caídas de un hogar a otro influía directamente en los resultados de la reducción debido a que en el momento de las caídas ocurre mayor

contacto entre las fases, y las partículas son bañadas completamente por el gas, calentándolas a las temperaturas indicadas y reduciendo el níquel. La reducción en la cama de mineral lleva el peso de la reducción alrededor de un 70 % de la misma, gracias al petróleo aditivo.

- Estabilidad en la Alimentación: La inestabilidad en la alimentación al horno afecta seriamente el perfil de temperaturas, y por ende la operación del mismo, ya que todas las condiciones son fijadas para el tonelaje a procesar, y si en este tiempo al horno se les suministra más mineral que el fijado, la temperatura comenzarán a bajar y si ocurre lo contrario la temperatura aumentan y ambas situaciones no son conveniente para el proceso ni para una buena estabilidad en la operación.

- Tiempo de Retención: En este equipo éste es un factor que incide directamente en los resultados de la reducción por el grado de terminación de la reacción.

6.1.3. Problema de Optimización Para Reducción del Mineral

En el proceso de producción del níquel de Punta Gorda, se pueden describir muchos problemas de optimización dinámica, en los que hay que alcanzar un equilibrio entre la ganancia inmediata y el comportamiento óptimo del sistema a lo largo del tiempo, y donde existen cambios en las entradas de la planta que aportan un cierto grado de incertidumbre al problema.

Como ejemplo se puede considerar el proceso de Reducción de Mineral. A lo largo de dicho proceso existe un consumo de petróleo tecnológico para poder establecer el perfil térmico de los hornos que determina las distintas reacciones químicas necesarias para el correcto desarrollo del proceso. Se

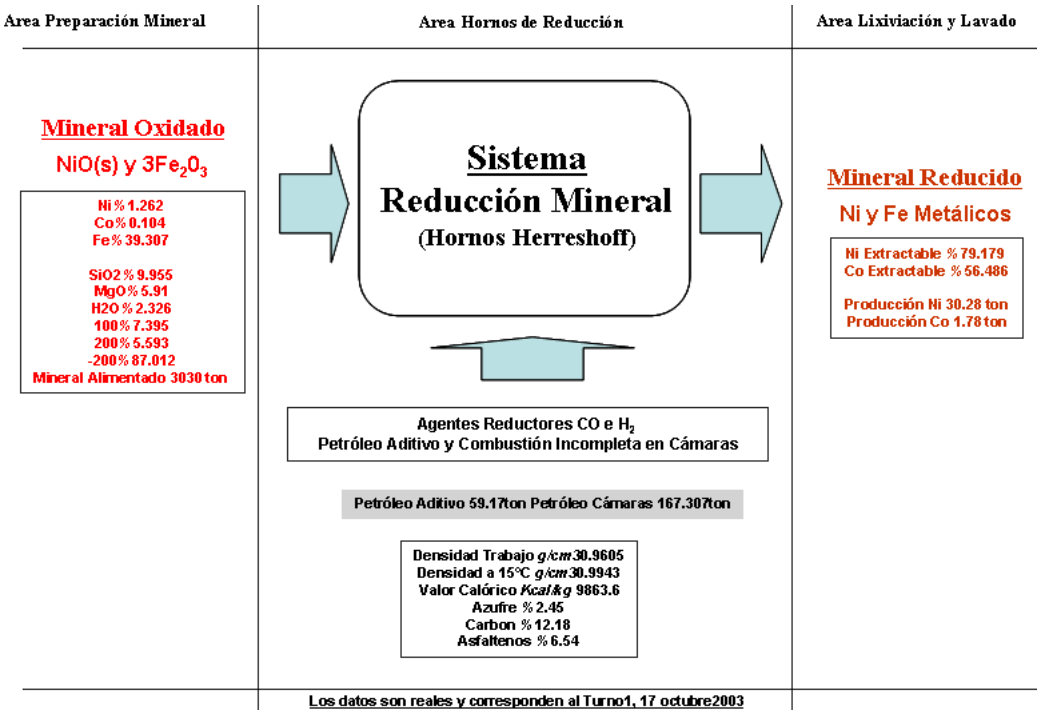


Fig. 6.2: Variables del Proceso Reducción Mineral

trata de un proceso complejo que requiere de decisiones que muchas veces debe tomar un operador en base a su experiencia e intuición. El objetivo de optimizar las necesidades de combustible a través de un estudio de los datos disponibles correspondientes a las entradas/salidas, puede ser considerado un trabajo de gran beneficio tecnológico.

En la figura se muestran las principales variables sobre las cuales trabajaremos para determinar las posibles relaciones entre ellas, mediante algoritmos de aproximación de funciones.

Con relación al proceso de Reducción de Mineral podemos plantear las siguientes funciones que lo caracterizan:

% Extracciones = f (Mineral Entrada, Perfil Temperatura Hogares, Agentes Reductores)

Perfil Temperatura Hogares = f (Mineral Entrada, Temperatura Cámaras)

Agentes Reductores = f (Mineral Entrada, Petróleo Aditivo, Petróleo en Cámaras)

Con la estimación de una función f partir de un conjunto finito de vectores o datos de entrada, el objetivo principal es aprender una relación funcional desconocida entre los vectores de entrada y sus salidas asociadas, utilizando un conjunto de ejemplos de entrenamiento. Una vez que dicha relación ha sido identificada, ésta puede ser utilizada para la obtención de nuevas salidas a partir de entradas que no se encuentren en el conjunto de entrenamiento.

Dado que las RBFNN tienen la capacidad de predecir, el módulo de aproximación funcional estará compuesto por una RBFNN diseñada con el algoritmo evolutivo presentado en los capítulos 4 y 5. En este caso concreto, cada 8 horas se dispondrá de una nueva muestra que podrá ser utilizada en el entrenamiento de la red y, gracias al paralelismo, se pueden diseñar redes muy precisas en ese intervalo de tiempo.

6.1.4. Resultados experimentales

Los datos empleados en los experimentos fueron obtenidos mediante la medición de los siguientes parámetros en el sistema real:

- *Entradas*: Ni, Fe, Co, Si, Mg, Ton, Temp. Hornos (9), *Salida*: Indice de Petroleo Aditivo

Los vectores de entrada se obtuvieron midiendo cada 8 horas los parámetros especificados arriba. Las muestras utilizadas fueron recogidas en 288 días del año 2006 de modo que el conjunto de datos está formado por 864 vectores de entrada con 27 variables cada uno. Aproximadamente, el 70 % de

	$\overline{RBF}$	# **Variables**	**Error de entrenamiento**	**Error de test**
MOFA	15 (3.1)	27	0.798 (0.036)	0.659 (0.074)
pEFSFA	7 (2.6)	4 (2.2)	0.642 (0.025)	0.515 (0.045)

Tab. 6.1: Medias y desviaciones estándar de los resultados obtenidos para la predicción del consumo de petróleo en el proceso de reducción del mineral.

las muestras disponibles han sido utilizadas para el entrenamiento dejando el resto para test. De este modo se escogieron de forma aleatoria 617 de las 864 muestras para entrenar las redes y las 247 restantes se utilizaron para test.

El algoritmo pEFSFA propuesto a lo largo de la memoria ha sido comparado con otro algoritmo *MOFA* propuesto en (**Guillén** et al., 2006c) para diseñar RBFNN el cual no considera la selección de variables de entrada ni mecanismos específicos que permitan mantener un equilibrio entre complejidad de las redes y la calidad de la aproximación. Los resultados se muestran en la Tabla 6.1. Como se puede observar, las mejoras introducidas gracias a la especialización paralela y la selección de variables permiten al algoritmo propuesto en esta memoria obtener unos resultados aceptables, mejorando significativamente los proporcionados por el algoritmo evolutivo propuesto inicialmente para resolver este problema (**Guillén** et al., 2007g).

6.2. Aplicación 2: Predicción de la Macrosomía

La macrosomía fetal, definida arbitrariamente como un peso del recién nacido superior a 4000 gramos complica más del 10 % de todos los embarazos en los Estados Unidos. Está asociada con un incremento en el riesgo de sección de cesárea y trauma del canal de nacimiento y del feto. La macrosomía fetal es difícil de predecir y las estimaciones ultrasonográficas suelen ser erróneas. La sección selectiva de cesárea para una posible macrosomía puede reducir

un gran número de procedimientos innecesarios.

Los profesionales del cuidado maternal se encuentran frecuentemente mujeres embarazadas donde se sospecha que pueda existir macrosomía fetal. Debido a los riesgos reconocidos en los embarazos con macrosomía, los médicos han intentado encontrar modos precisos para predecir el peso del recién nacido de forma precisa para optimizar el proceso del nacimiento (Langer et al., 1991); (Parks and Ziel, 1978); (Irion and Boulvain, 2000).

A continuación se hace una revisión de los distintos métodos utilizados para su predicción y su eficacia.

6.2.1. Predicción de la Macrosomía Fetal

El término "feto macrosómico" es erróneo dado que el peso al nacer no es conocido hasta que se da a luz. El criterio más común propuesto para macrosomía es un peso mayor que 4000 (Gregory et al., 1998) gramos o 4500 gramos (Berard et al., 1998). En 1990, siguiendo el primer criterio, supuso un 10.9 % de los embarazos y, siguiendo el segundo, un 1.8 % de los bebés nacidos en los Estados Unidos (of Health and Human Services, 1994). La definición más útil desde el punto de vista clínico es un peso por debajo del cual no aparecen complicaciones macrosómicas, tal como distocia de hombro. Desafortunadamente, las series de casos indican que la mitad de todos los casos de dystocia de hombro ocurren con un peso menor que el umbral establecido para la macrosomía (4000 g.) (Acker et al., 1986). Es más, casi la mitad de todos los casos de lesiones permanentes en el plexo branquial ocurren en recién nacidos con un peso inferior a 4500 g. (Ouzounian et al., 1998).

6.2.2. Estrategias para Predecir la Macrosomía

Las tres estrategias más importantes a la hora de predecir la macrosomía son:

- análisis de factores de riesgo
- estimación clínica mediante maniobras de Leopold
- ultrasonografía.

Cada método tiene limitaciones substanciales.

6.2.2.1. Factores de Riesgo

Los factores de riesgo identificados para la macrosomía fetal son (Boyd et al., 1983); (Sermer M et al., 1995); (Golditch and Kirkman, 1978); (Lazer et al., 1986); (Chauhan et al., 1992):

- Diabetes maternal
- Alteración de la intolerancia materna a la glucosa.
- Multiparidad
- Fetos macrosómicos previos
- Gestación prolongada
- Obesidad materna
- Excesiva ganancia de peso
- Feto varón
- Estatura de los padres

El factor de riesgo más importante es la diabetes maternal (Boyd et al., 1983) aunque estos factores no son todo lo significativos que se desearía puesto que incluso cuando dos o más de estos factores de riesgo están presentes, el riesgo de que el recién nacido sea macrosómico es del 32 % (Zamorski and Biggs, 2001). Es más, el 34 % de los recién nacidos macrosómicos no presentaban ninguno de los factores de riesgo mencionados anteriormente.

6.2.2.2. Estimación Clínica del Peso Fetal

Dentro de los factores que complican la estimación del tamaño del feto mediante la palpación del muro abdominal están: el volumen de flujo amniotico, el tamaño y la configuración del útero y los hábitos corporales maternos (Chauhan et al., 1992). Estas estimaciones suelen proporcionar un error medio de unos 300 gramos. (Chauhan et al., 1992); (Watson et al., 1988)

6.2.2.3. Ultrasonografía

En (Miller et al., 1988), la ultrasonografía se ha propuesto como una de las técnicas más precisas para calcular el peso del feto aunque los errores cometidos por esta técnica oscilan entre los 300 y los 550 gramos. De hecho, un estudio comparando diversas técnicas de estimación del peso determina que ésta es la menos precisa (McLaren et al., 1995).

6.2.3. Consecuencias de la Macrosomía Fetal

6.2.3.1. Consecuencias en el Feto

Potencialmente, el parto de un feto macrosómico puede tener graves consecuencias tanto para la madre como para el feto. Una de las consecuencias más temidas es la distocia de hombro dado que un cuarto de los fetos con distocia de hombro experimentan lesiones en plexo braquial (p.ej. parálisis

de Erb-Duchenne), lesiones en los nervios faciales o fracturas en el húmero o la clavícula (Gherman et al., 1998). Otra complicación temida en el parto de un feto macrosómico es la asfixia, aunque no es tan común (Baskett and Allen, 1995); (Lipscomb et al., 1995).

6.2.3.2. Consecuencias en la Madre

Para la madre, la macrosomía implica un incremento en el riesgo de cesárea, además de incrementar el riesgo de laceraciones de la vagina (Baskett and Allen, 1995).

6.2.3.3. Inducción Temprana del Parto

Dado que el feto continua incrementando su peso en 230 gramos cada semana después de la semana 37 (Ott, 1998), se ha sugerido la inducción temprana del parto para prevenir la macrosomía y sus lesiones asociadas (Boyd et al., 1983). Sin embargo, esta práctica, tal y como se ha concluido tras unos estudios observacionales incrementa el riesgo de cesárea (Friesen et al., 1995); (Delpapa and Mueller-Heubach, 1991); (Levine et al., 1992); (Weeks et al., 1995), incrementando las probabilidades de tener que realizarla con respecto a las cesáreas debido a fetos macrosómicos.

6.2.4. Predicción del Peso del Feto Mediante RBFNN

Para poder entrenar la red se dispuso de una cohorte proporcionada por el Departamento de Medicina Preventiva del Hospital Clínico Universitario San Cecílio (HCUSC) donde se registraron los datos proporcionados por mujeres que parieron entre el 1 de Enero y el 31 de Diciembre de 1995. El HCUSC es un hospital de educación superior que funciona para el Servicio Andaluz de Salud. Los criterios para la inclusión en el estudio fueron:

1. residencia habitual y atención médica en el area correspondiente al HCUSC
2. embarazo simple
3. primera consulta al médico antes de la semana 28 de embarazo
4. edad gestacional en el momento del parto de al menos 28 semanas.

Los criterios para la exclusión en el estudio fueron:

1. diabetes de tipo 1
2. diabetes de tipo 2
3. intolerancia a los carbohidratos antes del embarazo
4. embarazo sin control médico
5. partos con riesgo que obligaran a ser atendidos fuera del area de acción del HCUSC

El Servicio Nacional de Salud de España tiene un programa gratuito de cuidado del embarazo para todas las mujeres embarazadas. En 1995, se propone un plan urgente para la monitorización de la Diabetes Gestacional Mellitus (GDM), el cual fue realizado mediante 50-g test oral de glucosa durante una hora a las 24-28 semanas de embarazo. En el caso de obtener un valor de 7.8 mmol/dl o más, fue considerado como positivo y estos casos fueron re-evaluados con un test oral 100-g de glucosa durante 3 horas.

De un total de 2780 partos registrados, 2574 fueron válidos para ser incluidos en el estudio. El estudio tenía el objetivo de obtener información para el diagnostico y la determinación de los factores de riesgo de la GDM aunque

los datos recogidos son válidos para la predicción de la macrosomía puesto que ésta última es un factor de riesgo para la GDM.

Las fuentes de información principales fueron la historia médica de la madre y la libreta sanitaria cuando fue posible. La libreta contiene los datos almacenados mensualmente en cada visita al médico durante el embarazo: número de visita, semana de gestación, peso materno, presión sanguínea y observaciones generales, antecedentes, resultados de analíticas o ultrasonidos y especificación de cualquier tratamiento farmacológico.

De los 1225 casos con 117 variables registradas completamente, se ordenaron en base al peso del recién nacido, partiendo de esa ordenación, se iteró sobre todos los casos, seleccionándolo para test o entrenamiento con una probabilidad del 50 %. Esto proporcionó un conjunto de test de 610 casos y uno de entrenamiento de 615. Al ordenar el conjunto original de datos se está asegurando el que haya pesos similares en ambos subconjuntos. Se han realizado distintos experimentos considerando diferentes grupos de variables.

6.2.4.1. Considerando 117 variables

Se le ha proporcionado al algoritmo las 117 variables registradas para cada mujer, sin hacer ningún tipo de preselección, obteniendo los resultados mostrados a continuación:

Error de aproximación de test = 0.5046 (0.0444)
Error de aproximación de entrenamiento = 0.5019 (0.0583)
Número de RBF = 2.6 (1.07)
Variables seleccionadas =

45.10 %	Semanas de gestación
46.10 %	Dias de gestación
51.70 %	Longitud femoral
52.60 %	Perímetro craneal
53.100 %	Perímetro torácico
59.10 %	Estrés fetal
73.60 %	Sospecha de feto grande
74.10 %	Sufrimiento fetal
93.20 %	Diabetes gestacional

En estos resultados podemos apreciar cómo el algoritmo es capaz de generar redes que aproximan razonablemente bien el peso del recién nacido tanto para el conjunto de entrenamiento como para el de test. De las 117 variables se seleccionan muy pocas, coincidiendo éstas con los factores de riesgo citados en la bibliografía así como en las dimensiones del bebé. La longitud, el perímetro craneal y el torácico son medidas hechas una vez el bebé ha nacido por tanto invalidan el experimento en lo que se refiere a la predicción estricta del peso dado que a la hora de predecirlo, esos valores son desconocidos. Sin embargo este experimento sirve para validar el algoritmo en lo que se refiere a capacidad de identificar variables significativas para la red y capacidad de aproximación de los datos proporcionados. Además, el resultado más importante es la consideración de estos parámetros para determinar el peso del bebé ya que sus valores pueden ser obtenidos mediante ecografía. Por tanto, una nueva línea de investigación se abre en la cual se pasen a registrar estos datos y se experimente con ellos.

6.2.4.2. Considerando un conjunto reducido de variables

Con la colaboración del profesor José Juan Jiménez Moleón del Departamento de Medicina Preventiva de la Universidad de Granada, se ha realizado una selección de las posibles 50 variables más influyentes en el peso del neona-

to. Tras ejecutar el algoritmo con estos datos se han obtenido los siguientes resultados:

Error de aproximación de test = 0.8608 (0.0424)
Error de aproximación de entrenamiento = 0.7855 (0.0304)
Número de RBF = 2.27 (0.7)
Variables seleccionadas =

11.1 %	Madre realiza algún trabajo
5.5 %	Antecedentes de diabetes en la familia
38.8 %	Fumadora
38.8 %	Número de embarazos previos
5.5 %	Número de abortos previos
16.6 %	Número de partos previos
16.6 %	Patologías embrionarias
38.8 %	Tipo de patología embrionaria
16.6 %	Diabetes gestacional previa
66.6 %	Antecedentes de macrosomia
22.2 %	Talla
22.2 %	Peso al inicio del embarazo
44.4 %	Sospecha de feto grande
94.4 %	Peso al final del embarazo
22.2 %	Infecciones durante el embarazo
38.8 %	Tipo de infecciones durante el embarazo
100 %	Número de semanas de gestación
77.7 %	Número de días de gestación
44.4 %	Sexo del feto
5.5 %	Edad materna
27.7 %	Intolerancia a la glucosa
11.1 %	Hipertensión arterial
22.2 %	Antecedentes de muerte fetal
27.7 %	Número de hijos

16.6 % Nivel de estudios de la madre
22.2 % Antecedentes de diabetes gestacional en la familia
33.3 % Diabetes gestacional
22.2 % Número de cigarros al día durante el embarazo

En esta ocasión la calidad de la aproximación se ve disminuida ya que los datos no son tan determinantes como pueden serlo las medidas de feto. Sin embargo es destacable de nuevo la capacidad de generalización que mantienen las redes obteniendo un error de test similar al de entrenamiento, mejorando estudios preliminares (**Guillén** et al., 2007d). El algoritmo coincide con la bibliografía considerando los factores más relevantes en el peso del bebé los antecedentes de macrosomía, semanas y días de gestación, peso materno final, etc.

6.3. Conclusiones

Este capítulo ha mostrado dos ejemplos reales sobre los que se puede aplicar el algoritmo de forma provechosa.

En la primera aplicación descrita, se pretende predecir el consumo de petróleo durante el proceso de extracción del mineral para poder optimizar los recursos y ahorrar pérdidas innecesarias. El algoritmo se ha mostrado efectivo aún cuando solo se poseen un número de muestras reducido, conforme aumente el número de muestras la predicción será más precisa y, gracias al carácter paralelo del algoritmo, este incremento en el número de muestras no supondrá un inconveniente para cumplir la restricción temporal sobre la que se debe realizar la predicción.

La segunda aplicación tiene una doble tarea de identificar los factores de riesgo u otras variables que afecten al peso del neonato así como de predecir este valor. El objetivo es prever si el bebé presentará un peso por encima de lo

normal, pudiendo causar este hecho complicaciones en el parto. Los resultados a nivel de identificación de factores de riesgo han sido muy satisfactorios, abriendo la puerta a una nueva línea de investigación. En lo referente a la predicción del peso, los resultados son aceptables, obteniendo unos errores de aproximación no excesivamente elevados, si bien es necesario obtener más muestras para poder realizar predicciones más precisas.

7. CONCLUSIONES Y PRINCIPALES APORTACIONES

El trabajo de tesis doctoral presentado en esta memoria es una contribución al diseño automático de redes de funciones base radiales a partir de un conjunto de ejemplos de entrenamiento en problemas de aproximación de funciones. Las principales aportaciones y conclusiones obtenidas se resumen a continuación.

1. El Capítulo 1 ha descrito de forma coloquial y formal el problema con el que esta memoria se afronta: la aproximación funcional. Para resolver este problema se debe diseñar un modelo matemático que, a partir de un conjunto de entradas con su salida, sea capaz de aprender la relación entre ambos de modo que cuando reciba entradas nuevas, el modelo sea capaz de generar salidas aproximadas. La memoria del trabajo desarrollado comprende la descripción formal del modelo matemático que aproxima funciones así como de diversas metodologías para su diseño existentes. Tras esto, se proponen nuevos algoritmos que permiten crear modelos capaces de aproximar con gran precisión y son aplicados a problemas del mundo real, mostrando la utilidad del trabajo realizado.

2. En el Capítulo 2 se ha presentado el modelo que permite resolver el problema de aproximación funcional así como dos metodologías de di-

seño. En cuanto a la primera metodología clásica de diseño de RBFNN, se ha mostrado cómo un algoritmo diseñado específicamente para inicializar los centros de las RBF es capaz de mejorar considerablemente al resto de los algoritmos de *clustering* tradicionales. Continuando con la metodología clásica, para fijar el radio de las funciones base, se han presentado las heurísticas más usadas en la literatura (KNN y CIV), y se ha mostrado cómo la primera de ellas produce demasiado solapamiento entre las funciones base conforme K aumenta, llegándose a perder por completo la localidad de las funciones base. Respecto a la segunda metodología de diseño, se ha descrito un algoritmo evolutivo existente en la bibliografía que establece todos los parámetros que definen una RBFNN salvo el número de neuronas que debe poseer la red. Para ello, se adaptaron la representación de los individuos y los operadores de cruce y mutación, añadiendo conocimiento experto para guiar las modificaciones realizadas en la red hacia mejores zonas del espacio solución.

3. En el Capítulo 3 se han aportado soluciones a los problemas presentados por el algoritmo CFA, mostrando cómo la simplificación de ciertos aspectos y la inclusión de nuevas técnicas permiten la mejora de los resultados. A lo largo del capítulo también se ha demostrado lo importante de hacer una adecuada inicialización de los centros para poder obtener RBFNN que sean capaces de hacer buenas aproximaciones, justificando el trabajo presentado. Dado que uno de los elementos más importantes en un algoritmo de *clustering* es el tipo de partición de los datos que realiza, se ha hecho un estudio en profundidad de cómo las distintas posibilidades afectan al comportamiento del algoritmo. Las conclusiones han mostrado cómo el enfoque difuso tiene un compor-

tamiento excepcional si bien, en ciertas circunstancias, la hibridación con el enfoque posibilístico, puede mejorar los resultados a costa de un mayor tiempo de ejecución y la necesidad de inicializar más parámetros. Tras haber estudiado la inicialización de los centros, se ha observado como también es necesario asignar un valor adecuado a los radios de cada RBF. Con el objetivo de mantener la interpretabilidad durante la inicialización de los centros, se ha propuesto un nuevo algoritmo que es capaz de situar los centros en zonas adecuadas y además permite dar un valor inicial adecuado también para los radios. Tal y como se ha mostrado en el capítulo, este nuevo algoritmo proporciona unos resultados bastante satisfactorios.

4. El cuarto capítulo ha presentado los componentes de un nuevo algoritmo que recibe como entrada una serie de muestras con su valor de salida asociado y es capaz de diseñar una RFBNN que, no solo infiere la relación entre las entradas y la salida, sino que es capaz de determinar qué elementos (variables) del vector que identifica cada muestra son necesarios para poder calcular una nueva salida a partir de una entrada nueva. Para poder generar las RBFNN, el algoritmo hibrida dos técnicas de diseño de RBFNN, la búsqueda local y la evolutiva, integrándolas de modo que suplan las carencias de una y de otra entre ambas. Dentro de la optimización mediante búsqueda local, se ha hecho uso de los nuevos algoritmos presentados en el capítulo anterior. En el campo de los algoritmos evolutivos, el nuevo algoritmo ha presentado varios elementos nuevos:

   a) la propuesta de nuevos operadores de cruce y de mutación para evolucionar una RBFNN

b) un nuevo esquema de hibridación entre búsqueda local y algoritmos evolutivos

c) un nuevo método para controlar la complejidad de la red basado en el criterio para determinar la dominancia entre soluciones.

5. Este capítulo profundiza en el análisis e implementación del algoritmo evolutivo propuesto en el Capítulo 4, centrándose y estudiando con detalle su carácter paralelo. Se ha mostrado cómo el paralelismo funcional proporcionado por un algoritmo genético distribuido heterogéneo, donde cada isla se especializa en un aspecto del diseño, es capaz de mejorar los resultados de un algoritmo genético no distribuido. Además, el paralelismo no proporciona mejores resultados únicamente sino que, al obtenerse un paralelismo de datos, también se reduce considerablemente el tiempo de ejecución. Una característica muy importante del algoritmo es su escalabilidad ya que, como se ha mostrado, conforme aumenta el número de recursos, se disminuye el tiempo de ejecución pero no la calidad de los resultados como podría ocurrir al disminuir el número de individuos por isla. Por último, se ha descrito uno de los elementos que permiten la ejecución de algoritmos paralelos desarrollados en MATLAB que fue desarrollado específicamente para poder implementar el algoritmo propuesto aunque puede ser utilizado por cualquier programa MATLAB que desee realizar llamadas a la biblioteca de funciones MPI. Se ha comparado la nueva interfaz desarrollada con otra interfaz existente y se ha mostrado cómo la nueva interfaz puede proporcionar menos latencia de comunicación.

6. El capítulo de aplicaciones ha mostrado dos ejemplos reales sobre los que se puede aplicar el algoritmo de forma provechosa. En la primera

aplicación descrita, se pretende predecir el consumo de petróleo durante el proceso de extracción del mineral para poder optimizar los recursos y ahorrar pérdidas innecesarias. El algoritmo se ha mostrado efectivo aún cuando solo se poseen un número de muestras reducido, conforme aumente el número de muestras la predicción será más precisa y, gracias al carácter paralelo del algoritmo, este incremento en el número de muestras no supondrá un inconveniente para cumplir la restricción temporal sobre la que se debe realizar la predicción. La segunda aplicación tiene una doble tarea de identificar los factores de riesgo u otras variables que afecten al peso del neonato así como de predecir este valor. El objetivo es prever si el bebé presentará un peso por encima de lo normal, pudiendo causar este hecho complicaciones en el parto. Los resultados a nivel de identificación de factores de riesgo han sido muy satisfactorios, abriendo la puerta a una nueva línea de investigación. En lo referente a la predicción del peso, los resultados son aceptables, obteniendo unos errores de aproximación no excesivamente elevados.

CONCLUSIONS AND MAIN INNOVATIONS

The contribution of the work described in this dissertation has concentrated on the automatic design of radial basis function neural networks. The conclusions reached and the main innovations made are summarized below.

1. Chapter 1 introduced the problem to be solved: function approximation. A mathematical model is used to solve this problem that, given a set of inputs with their corresponding outputs, one is able to learn the relationship between the inputs and the output and to generate new outputs from input vectors that were not in the original data set.

2. The mathematical model used throughout the dissertation is introduced in Chapter 2. The model uses a special type of artificial neural network which has as activation function a radial basis one. This chapter also introduces the two main methodologies for the design of such networks: local and global optimization. The local optimization methodology includes three main steps and the use of a clustering algorithm to initialize the value of the centers of the Radial Basis Function (RBF). An algorithm proposed in previous work, designed specifically for this purpose, presented. Then, the most common heuristics ways to initialize the radii are presented. After this, an existing algorithm used to

design RBF Neural Networks (RBFNNs) using the global optimization methodology was introduced. This algorithm is an evolutive approach that has adapted the encoding of the individuals and the evolutive operators to evolve RBFNN.

3. Chapter 3 describes the flaws in the clustering algorithm presented in the previous chapter and proposes a new algorithm (Improved Clustering for Function Approximation, ICFA) which is able to obtain better solutions in less computational time. The way in which the input data is partitioned is one of the most important problems solved by this new algorithm. In the previous algorithm a hard partition of the data was used and while the ICFA algorithm proposes the use of a fuzzy partition. However, many other approaches to partition the data have been described in the literature. This chapter also analyzes the behavior of the ICGA algorithm using other types of partitions such us possibilistic, fuzzy-possibilistic, and possibilistic-fuzzy. After analyzing all these clustering techniques, it is seen that not only do the centers require careful initialization but also the initial radii as well. A new algorithm is introduced that, changing the perspective of classical clustering, is able to properly initialize both the centers and the radii.

4. Chapter 4 proposes the elements of a new evolutive algorithm that is able to evolve all the parameters needed to design an RBFNN, these are: the set of variables that feed the network, the number of RBFs, the position of the centers for each RBF and the value of their radii. The evolutive algorithm is based on the Nondominated Sorting Genetic Algorithm II (NSGA-II) which, thanks to its multiobjective optimization, helps to keep a balance between the size of the network and the accura-

cy of the approximation. This classical algorithm has been merged with the local optimization techniques presented in Chapter 2, producing a memetic algorithm which applies local optimization at the beginning and at the end of the global optimization. In the field of evolutive algorithms, the algorithm proposed in this chapter presents the following innovations:

   a) New crossover and mutation operators designed to evolve all the parameters that define an RBFNN.

   b) A new scheme that combines local and global optimization.

   c) A new method that helps to keep a balance between the approximation accuracy and the size of the network that is based on a fuzzy dominance criteria.

5. The elements of the algorithm described in Chapter 4 have been integrated together over a parallel programming paradigm. The algorithm has been designed to take advantage of the benefits obtained from the use of functional parallelism obtained from the design of the crossover and mutation operators. Thus, a heterogeneous distributed algorithm, where each island evolves a different aspect of the design of an RBFNN, has been developed. The parallel aspect of the algorithm not only helps to provide more accurate networks but also reduces the computational time due to the intrinsic data parallelism of genetic algorithms. A study of the scalability of the algorithm has been done, showing how the new algorithm is able to reduce the computation time as new processors are available without decreasing the quality of the results. This chapter presents as well a new interface so MATLAB applications can invoke Message Passing Interface (MPI) library functions. This new interface

has been developed in such a way that MATLAB is able to use any routine that adheres to the MPI standard. Although other interfaces have been previously developed, the new one provides a smaller overhead time on each function call.

6. Finally, Chapter 6 shows how the new proposed algorithm is able to solve two real world problems. The first problem consists in the prediction of fuel consumption during the process of mineral reduction. The aim here is, based on a prediction of the fuel consumption, to define a control system that regulates the temperature of the ovens where the mineral is extracted. The approximations of the fuel consumption where satisfactory and the time restriction to predict is fulfilled thanks to the parallelization of the algorithm. The second application requires the prediction of the weight of a newborn baby in order to prevent complications during childbirth and also to help in the diagnosis of any other disease that might be present since the weight is a risk factor for many of them. The results obtained in the identification of risk factors were very successful, showing how the algorithm is not only able to make good approximations of the birth weight but is also able to identify the more relevant input variables.

BIBLIOGRAPHY

Acker, D., Sachs, B., and Friedman, E. (1986). Risk factors for shoulder dystocia in the average weight infant. *Obstet Gynecol*, 67:614–618.

Adamidis, P. and Petridis, V. (1996). Co-operating populations with different evolution behaviors. In *Proceedings of the 3th IEEE Conference on Evolutionary Computation*, pages 188–191.

Adamidis, P. and Petridis, V. (2002). On modelling evolutionary algorithm implementations through co-operating populations. In *VII Parallel Problem Solving from Nature*, pages 321–330.

Aickelin, U. and Bull, L. (2002). Partnering strategies for fitness evaluation in a pyramidal evolutionary algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 263–270.

Alander, J. T. (1992). On Optimal Population size of Genetic Algorithms. In Dewilde, P. and Vandewalle, J., editors, *Proceedings of the 6th Annual European Conference on Computer Systems and Software Engineering* pages 65–70, The Hague. IEEE Computer Society Press.

Alba, E., Luna, F., and Nebro, A. J. (2003). Parallel heterogeneous genetic algorithms for continuous optimization. In *Proceedings of the International Parallel and Distributed Processing Symposium*

Alba, E., Luna, F., and Nebro, A. J. (2004). Advances in parallel heterogeneous genetic algorithms for continuous optimization. *Int. J. Appl. Math. Comput. Sci.*, 14:317–333.

Antonisse, H. J. (1989). A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint. In (Schaffer, 1989), pages 191–197.

Bäck, T., Fogel, D. B., and Michalewicz, Z., editors (1997a). *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, Bristol, New York.

Bäck, T., Hammel, U., and Schwefel, H. P. (1997b). Evolutionary Computation: Comments on the History and Current State. *IEEE Trans. Evol. Comput.*, 1(1):3–17.

Barni, M., Capellini, V., and Mecocci, A. (1996). Comments on 'A possibilistic approach to clustering'. *IEEE Transactions on Fuzzy Systems*, 4:393–396.

Barreto, A. M. S., Barbosa, H. J. C., and Ebecken, N. F. F. (2006). Gols - genetic orthogonal least squares algorithm for training rbf networks. *Neurocomputing*, 69(16-18):2041–2064.

Baskett, T. and Allen, A. (1995). Perinatal implications of shoulder dystocia. *Obstet Gynecol*, 86:14–17.

Bellman, R. (1966). Dynamic programming, system identification, and suboptimization. *SIAM Journal on Control and Optimization*, 4:1–5.

Benoudjit, N. and Verleysen, M. (2003). On the kernel widths in radial basis function networks. *Neural Processing Letters*, 18(2):139–154.

Berard, J., Dufour, P., Vinatier, D., Subtil, D., Vanderstichele, S., Monnier, J., and et al. (1998). Fetal macrosomia: risk factors and outcome. A study of the outcome concerning 100 cases >4500 g. *Eur J Obstet Gynecol Reprod Biol*, 77:51–59.

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum, New York.

Bors, A. G. (2001). Introduction of the Radial Basis Function (RBF) networks. *OnLine Symposium for Electronics Engineers*, 1:1–7.

Bounds, D. G. (1987). New Optimization Methods from Physics and Biology. *Nature*, 329:215–219.

Box, G. E. P. and Jenkins, G. M. (1976a). *Time Series Analysis, Forecasting and Control*. Holden-Day, San Francisco, CA.

Box, G. E. P. and Jenkins, G. M. (1976b). *Time series analysis, forecasting and control*. Holden Day: San Francisco, CA.

Boyd, M., Usher, R., and McLean, F. (1983). Fetal macrosomia: prediction, risks, proposed management. *Obstet Gynecol*, 61:715–722.

Broomhead, D. S. and Lowe, D. (1988). Multivariate Functional Interpolation and Adaptive Networks. *Complex Systems*, 2:321–355.

Buckles, B. P. and Petry, F. E., editors (1993). *Genetic Algorithms*. Electronica Books Ltd., Middlesex (UK).

B.V. Bonnlander and A.S. Weigend (2004). Selecting input variables using mutual information and nonparametric density estimation. In *Proc. of the ISANN*, Taiwan.

Cantú-Paz, E. (2000). Markov chain of parallel genetic algorithms. *IEEE. Trans. Evolutionary Computation*, 4:216–226.

Chankong, V. and Haimes, Y. Y. (1983). *Multiobjective Decision Making Theory and Methodology*. North-Holland, New York.

Chauhan, S., Lutton, P., Bailey, K., Guerrieri, J., and Morrison, J. (1992). Intrapartum clinical, sonographic, and parous patients? estimates of newborn birth weight. *Obstet Gynecol*, 79:956–958.

Chen, S., Cowan, C. F., and Grant, P. M. (1991). Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks. *IEEE Trans. Neural Networks*, 2:302–309.

Cherkassky, V., Gehring, D., and Mulier, F. (1996). Comparison of adaptative methods for function estimation from samples. *IEEE Transactions on Neural Networks*, 7:969–984.

Cherkassky, V. and H.Lari-Najafi (1991). Constrained topological mapping for nonparametric regression analysis. *Neural Networks*, 4(1):27–40.

Davis, L. (1989). Adapting Operator Probabilities in Genetic Algorithms. In (Schaffer, 1989), pages 61–69.

Davis, L., editor (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

De Jong, K. A. and Spears, W. M. (1991). An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms. In Schwefel, H. P. and Männer, R., editors, *Proceedings of the First International Conference on Parallel Problem Solving from Nature, PPSN I*, volume

496 of *Lecture Notes in Computer Science*, pages 38–47, University of Dortmund. Springer-Verlag.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation*, 6(2):182–197.

Deb, K. and Goel, T. (2001). Controlled elitist non-dominated sorting genetic algorithms for better convergence. In *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 67–81. Springer-Verlag.

Delpapa, E. and Mueller-Heubach, E. (1991). Pregnancy outcome following ultrasound diagnosis of macrosomia. *Obstet Gynecol*, 78:340–343.

Eshelman, L. J., Caruana, A., and Schaffer, J. D. (1993). Realcoded genetic algorithms and interval schemata. In Schaffer, J. D., editor, *Foundation of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann.

Falkenauer, E. (1998). *Genetic Algorithms and Grouping Problems*. Wiley.

Fernández, J., Anguita, M., Ros, E., and Bernier, J. (2006). SCE Toolboxes for the development of high-level parallel applications. *Lecture Notes in Computer Science*, 3992:518–525.

Fogarty, T. C. (1989). Varying the Probability of Mutation in Genetic Algorithms. In (Schaffer, 1989), pages 104–109.

Fogel, D. B. (1992). *Evolving Artificial Intelligence*. PhD thesis, University of California, San Diego.

Forrest, S. (1993a). Genetic Algorithms - Principles of Natural Selection applied to Computation. *Science*, 261(5123):872–878.

Forrest, S., editor (1993b). *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann.

Friedman, J. (1981). Projection pursuit regression. *Journal of the American Statistical Association*, 76:817–823.

Friedman, J. H. (1991). Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, 19:1–141.

Friesen, C., Miller, A., and Rayburn, W. (1995). Influence of spontaneous or induced labor on delivering the macrosomic fetus. *Am J Perinatol*, 12:63–66.

Gandibleux, X., Morita, H., and Katoh, N. (2001). The supported solutions used as a genetic information in a population heuristic. In *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 429–442. Springer-Verlag.

García-Pedrajas, N., Hervás-Martínez, C., and Muñoz-Pérez, J. (2002). Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks). *Neural Netw.*, 15(10):1259–1278.

Gersho, A. (1979). Asymptotically Optimal Block Quantization. *IEEE Trans. Inf. Theory*, 25(4):373–380.

Gherman, R., Ouzounian, J., and Goodwin, T. (1998). Obstetric maneuvers for shoulder dystocia and associated fetal morbidity. *Am J Obstet Gynecol*, 178:1126–1130.

Goldberg, D. E. (1985). Optimal Initial Population Size for Binary-Coded Genetic Algorithms. Technical Report TCGA 85001, University of Alabama, Tuscaloosa.

Goldberg, D. E. (1989a). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley.

Goldberg, D. E. (1989b). Sizing Populations for Serial and Parallel Genetic Algorithms. In (Schaffer, 1989), pages 70–79.

Goldberg, D. E. (1994). Genetic and Evolutionary Algorithms come of age. *Communications of the ACM.*, 37(3):113–119.

Golditch, I. and Kirkman, K. (1978). The large fetus. Management and outcome. *Obstet Gynecol*, 52:26–30.

González, J., Rojas, I., Ortega, J., Pomares, H., Fernández, F., and Díaz, A. (2003). Multiobjective evolutionary optimization of the size, shape, and position parameters of radial basis function networks for function approximation. *IEEE Transactions on Neural Networks*, 14(6):1478–1495.

González, J., Rojas, I., Pomares, H., Ortega, J., and Prieto, A. (2001a). A new Clustering Technique for Function Approximation. *IEEE Trans. Neural Networks*. In press. Paper code TNN B267.

González, J., Rojas, I., Pomares, H., Ortega, J., and Prieto, A. (2002). A new Clustering Technique for Function Aproximation. *IEEE Transactions on Neural Networks*, 13(1):132–142.

González, J., Rojas, I., Pomares, H., and Salmerón, M. (2001b). Expert Mutation Operators for the Evolution of Radial Basis Function Neural Networks. In Mira, J. and Prieto, A., editors, *Connectionist Models of Neurons, Learning Processes and Artificial Intelligence*, volume 2084 of *Lecture Notes in Computer Science*, pages 538–545. Springer-Verlag.

González, J., Rojas, I., Pomares, H., Salmerón, M., and Prieto, A. (2000). Evolution of Fuzzy Patches for Function Approximation. In Ollero, A., Sánchez, S., Arrue, B., and Baturone, I., editors, *Actas del X Congreso Español sobre Tecnologías y Lógica Difusa, ESTYLF 2000*, pages 489–495, Seville, Spain.

Gregory, K., Henry, O., Ramicone, E., Chan, L., and Platt, L. (1998). Maternal and infant complications in high and normal weight infants by method of delivery. *Obstet Gynecol*, 92:505–513.

Guillén, **A.**., González, J., Rojas, I., Pomares, H., Herrera, L., Valenzuela, O., and Prieto, A. (2007a). Improving Clustering Technique for Functional Approximation Problem Using Fuzzy Logic: ICFA algorithm.*Neurocomputing*, DOI:10.1016/j.neucom.2006.06.017.

Guillén, **A.**., González, J., Rojas, I., Pomares, H., Herrera, L., Valenzuela, O., and Rojas, F. (2007b). Output Value-Based Initialization For Radial Basis Function Neural Networks. *Neural Processing Letters*, 25(3): 209-225, DOI:10.1007/s11063-007-9039-8.

Guillén, **A.**., González, J., Rojas, I., Pomares, H., Herrera, L., Valenzuela, O., and Rojas, F. (2007c). Studying Possibility in a Clustering for Function Approximation Algorithm. *Neural Computing & Applications*.

Guillén, **A.**., Pomares, H., Rojas, I., ález, J. G., and Herrera, L. (2007d). A First Approach to Birth Weight Prediction Using RBFNNs. *Lecture Notes in Artificial Intelligence*, 4527:253–260.

Guillén, **A.**., Pomares, H., Rojas, I., ález, J. G., Herrera, L., and Prieto, A. (2007e). Parallel Multi-objective Memetic RBFNNs Design and Fea-

ture Selection for Function Approximation Problems. *Lecture Notes in Artificial Intelligence*, 4507:341–350.

Guillén, **A.**., Rojas, I., ález, J. G., Pomares, H., Herrera, L., and Paechter, B. (2006a). Improving the Performance of Multi-objective Genetic Algorithm for Function Approximation Through Parallel Islands Specialisation. *Lecture Notes in Artificial Intelligence*, 4304:1127–1132.

Guillén, **A.**., Rojas, I., ález, J. G., Pomares, H., Herrera, L., and Paechter, B. (2007f). Boosting the performance of a multiobjective algorithm to design RBFNNs through parallelization. *Lecture Notes in Artificial Intelligence*, 4431:85–92.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., Herrera, L., and Fernández, F. (2007g). A New Multiobjective RBFNNs Designer and Feature Selector for a Mineral Reduction Application. In *2007 IEEE International Conference on Fuzzy Systems*.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., Herrera, L., and Prieto, A. (2004a). Using a new clustering algorithm to design RBF Networks for functional approximation problem. In *Proceedings of the Learning'04 International Conference*, pages 19–24, Elche.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., Herrera, L., and Prieto, A. (2006b). Supervised RBFNN Centers and Radii Initialization for Function Approximation Problems. In *International Joint Conference on Neural Networks, 2006. IJCNN '06*, pages 5814–5819.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., Herrera, L., Valenzuela, O., and Prieto, A. (2005a). A Possibilistic Approach to RBFN Centers Initialization. *Lecture Notes in Computer Science*, 3642:174–183.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., Herrera, L., Valenzuela, O., and Prieto, A. (2005b). Improving Clustering Technique for Functional Approximation Problem Using Fuzzy Logic: ICFA algorithm. *Lecture Notes in Computer Science*, 3512:272–280.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., and Herrera, L. J. (2004b). Clustering based algorithm for RFBs centers initialization. In *XI SIGEF Congress, Techniques and Methodologies for the Information and Knowledge Economy*, pages 44–63, Reggio Calabria-Messina.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., and Herrera, L. J. (2005c). RBF centers initialization using fuzzy clustering technique for function approximation problems. *Fuzzy Economic Review*, 10(2):27–45.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., Herrera, L. J., and Fernández, F. (2006c). Multiobjective RBFNNs Designer for Function Approximation: An Application for Mineral Reduction. *Lecture Notes in Computer Science*, 4221:511–520.

Guillén, **A.**., Rojas, I., González, J., Pomares, H., Herrera, L. J., and Prieto, A. (2006d). A Fuzzy-Possibilistic Fuzzy Ruled Clustering Algorithm for RBFNNs Design. *Lecture Notes in Computer Science*, 4259:647–656.

Guillén, **A.**., Rojas, I., Ros, E., and Herrera, L. (2005d). Using Fuzzy Clustering Technique for Function Approximation to Approximate ECG Signals. *Lecture Notes in Computer Science*, 3562:538–547.

Gustafson, E. and Kessel, W. (1979). Fuzzy clustering with a fuzzy covariance matrix. *IEEE CDC*, pages 761–766.

Hans, A. E. (1988). Multicriteria Optimization for Highly Accurate Systems. In Stadler, W., editor, *Multicriteria Optimization in Engineering*

and Sciences, Mathematical Concepts and Methods in Science and Engineering, volume 19, pages 309–352, New York. Plenum Press.

Harald, S., Alexander, K., Sergey, A., and Peter, G. (2004). Least dependent component analysis based on mutual information. *Physics Review*.

Harik, G., Cantú-Paz, E., Goldberg, D. E., and Miller, B. (1999). The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations. *Evolutionary Computation*, 7(3):231–253.

Hartigan, J. A. (1975). *Clustering Algorithms*. John Wiley, New York.

Hatanaka, T., Kondo, N., and Uosaki, K. (2003). Multi-objective structure selection for radial basis function networks based on genetic algorithm. In *The 2003 Congress on Evolutionary Computation*, pages 1095–1100.

Herrera, F. and Lozano, M. (2000). Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 4(1):43.

Herrera, F., Lozano, M., and Verdegay, J. L. (1998). Tackling real-coded genetic algorithms: operators and tools for the behavioural analysis . *Artificial Intelligence Reviews*, 12(4):265–319.

Hilliard, M., Richardson, J. T., Palmer, M. R., and Liepins, G. (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In (Schaffer, 1989), pages 191–197.

Hinterding, R., Michalewicz, Z., and Peachey, T. (1996). Selfadaptive genetic algorithm for numeric functions. In *IV Parallel Problem Solving from Nature*, pages 420–429.

Holland, J. J. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press.

Homaifar, A., Lai, S. H. Y., and Qi, X. (1994). Constrained Optimization via Genetic Algorithms. *Simulation*, 62:242–254.

Horn, J. (1997). Multicriteria Decision Making. In (Bäck et al., 1997a).

Hwang, C. L. and Masud, A. S. M. (1979). *Multiple Objective Decision Making – Methods and Applications*, volume 164 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin.

Irion, O. and Boulvain, M. (2000). Induction of labour for suspected fetal macrosomia. *Cochrane Database Syst Rev*.

Ishibuchi, H. and Kaige, S. (2004). Implementation of simple multiobjective memetic algorithms and its applications to knapsack problems. *Int. J. Hybrid Intell. Syst.*, 1(1):22–35.

Jang, J.-S. R. and Sun, C.-T. (1995). Neuro-fuzzy modeling and control. *Proceedings of the IEEE*.

János Abonyi and Robert Babuska and Ferenc Szeifert (2002). Modified gath-geva fuzzy clustering for identification of takagi-sugeno fuzzy models. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(5):612–621.

Kanjilal, P. P. and Banerjee, D. (1995). On the Application of Orthogonal Transformation for the Design and Analysis of Feedforward Networks. *IEEE Trans. Neural Networks*, 6(5):1061–1070.

Karayiannis, N. B. and Mi, G. W. (1997). Growing Radial Basis Neural Networks: Merging Supervised and Unsupervised Learning with Network Growth Techniques. *IEEE Trans. Neural Networks*, 8(6):1492–1506.

Kelly, J. D. and Davis, L. (1991). Further research on feature selection and classification using genetic algorithms. In *Proc. 4th Int. Conf. Genetic Algorithms Appl.*, pages 377–383.

Knowles, J. and Corne, D. (2002). On metrics for comparing non-dominated sets. In Congress on Evolutionary Computation (CEC 2002).

Kraskov, A., Stögbauer, H., and Grassberger, P. (2004). Estimating mutual information. *Physics Review*.

Krishnapuram, R. and Keller, J. M. (1993). A Possibilistic Approach to Clustering. *IEEE Transactions on Fuzzy Systems*, 1(2):98–110.

Langer, O., Berkus, M., Huff, R., and Samueloff, A. (1991). Shoulder dystocia: should the fetus weighing greater than or equal to 4000 grams be delivered by cesarean section? *Am J Obstet Gynecol*, (165):831–837.

Lazer, S., Biale, Y., Mazor, M., Lewenthal, H., and Insler, V. (1986). Complications associated with the macrosomic fetus. *J Reprod Med*, 31:501–505.

Leski, J. M. (2003). Generalized Weighted Conditional Fuzzy Clustering. *IEEE Transactions on Fuzzy Systems*, 11(6):709–715.

Levine, A., Lockwood, C., Brown, B., Lapinski, R., and Berkowitz, R. (1992). Sonographic diagnosis of the large for gestational age fetus at term: does it make a difference? *Obstet Gynecol*, 79:55–58.

Lin, S.-L., III, W. P., and Goodman, E. (1994). Coarsegrain parallel genetic algorithms: Categorization and new approach. In *Proceedings of the 6th IEEE Symp. Parallel and Distributed Processing* pages 28–37.

Lipscomb, K., Gregory, K., and Shaw, K. (1995). The outcome of macrosomic infants weighing at least 4500 grams: Los Angeles County + University of Southern California experience. *Obstet Gynecol*, 85:558–564.

Mackey, M. C. and Glass, L. (1977). Oscillation and Chaos in Physiological Control Systems. *Science*, 197(4300):287–289.

Marquardt, D. W. (1963). An Algorithm for Least-Squares Estimation of Nonlinear Inequalities. *SIAM J. Appl. Math.*, 11:431–441.

Mártínez, A., Mártínez, F., Hervás, C., and García, N. (2005). Evolutionary product unit based neural networks for regression. *Neural Networks*.

Martínez-Estudillo, F., Hervás-Martínez, C., Martínez-Estudillo, A., and Ortiz-Boyer, D. (2005). Memetic algorithms to product-unit neural networks for regression. In *IWANN*, pages 83–90.

McLaren, R., Puckett, J., and Chauhan, S. (1995). Estimators of birth weight in pregnant women requiring insulin: a comparison of seven sonographic models. *Obstet Gynecol*, 85:565–569.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 3rd edition.

Miki, M., Hiroyasu, T., Kaneko, M., and Hatanaka, K. (1999). A parallel genetic algorithm with distributed environment scheme. In *Proceedings of the IEEE Conf. Systems, Man and Cybernetics*, pages 695–700.

Miki, T. H. M. and Negami, M. (1999). Distributed genetic algorithms with randomized migration rate. In *Proceedings of the IEEE Conf. Systems, Man and Cybernetics*, pages 689–694.

Milca. *http://www.klab.caltech.edu/~kraskov/MILCA/*.

Miller, J., Brown, H., Khawli, O., 2d Pastorek, J., and Gabert, H. (1988). Ultrasonographic identification of the macrosomic fetus. *Am J Obstet Gynecol*, 159:1110–1114.

Moody, J. E. and Darken, C. J. (1989). Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation*, 1:281–294.

Moscato, P. (1992). A memetic approach for the travelling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *Parallel Computing and Transputer Applications*, pages 177–176.

Murata, T., Kaige, S., and Ishibuchi, H. (2003). Generalization of dominance relation-based replacement rules for memetic emo algorithms. In *GECCO*, pages 1234–1245.

Musavi, M. T., Ahmed, W., Chan, K., Faris, K., and Hummels, D. (1992a). On the training of radial basis functions classifiers. *Neural Networks*, 5(4):595–603.

Musavi, M. T., Ahmed, W., Chan, K. H., Faris, K. B., and Hummels, D. M. (1992b). On the Training of Radial Basis Function Classifiers. *Neural Networks*, 5(4):595–603.

of Health, U. D. and Human Services, P. H. S. (1994). National Center for Health Statistics. Vital Statistics of the United States. 1.

Ong, Y.-S., Lim, M.-H., Zhu, N., and Wong, K. W. (2006). Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(1):141–152.

Ott, W. (1998). The diagnosis of altered fetal growth. *Obstet Gynecol Clin North Am*, 15:237–263.

Ouzounian, J., Korst, L., and Phelan, J. (1998). Permanent Erb's palsy: a lack of a relationship with obstetrical risk factors. *Am J Perinatol*, 15:221–223.

Pal, N. R., Pal, K., and Bezdek, J. C. (1997). A Mixed C–Means Clustering Model. In *Proceedings of the 6th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'97)*, volume 1, pages 11–21, Barcelona.

Pareto, V. (1896). *Cours D'Economie Politique*, volume I and II. F. Rouge, Lausanne.

Park, J. and Sandberg, I. (1993a). Approximation and Radial Basis Function Networks. *Neural Computation*, 5:305–316.

Park, J. and Sandberg, I. W. (1991a). Universal Approximation using Radial-Basis-Function Networks. *Neural Computation*, 3:546–557.

Park, J. and Sandberg, I. W. (1993b). Approximation and Radial-Basis-Function Networks. *Neural Computation*, 5:305–316.

Park, J. and Sandberg, J. W. (1991b). Universal approximation using radial basis functions network. *Neural Computation*, 3:246–257.

Parks, D. and Ziel, H. (1978). Macrosomia. A proposed indication for primary cesarean section. *Obstet Gynecol*, 52:407–409.

Patanè, G. and Russo, M. (2001). The Enhanced-LBG algorithm. *Neural Networks*, 14(9):1219–1237.

Pedrycz, W. (1998). Conditional Fuzzy Clustering in the Design of Radial Basis Function Neural Networks. *IEEE Transactions on Neural Networks*, 9(4):601–612.

Pelikan, M. and Lobo, F. (1999). Parameter-less Genetic Algorithm: A Worst-case Time and Space Complexity Analysis. Technical Report IlliGAL 99014, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.

Poggio, T. and Girosi, F. (1990). Networks for approximation and learning. In *Proceedings of the IEEE*, volume 78, pages 1481–1497.

Pomares, H. (2000). Nueva metodología para el diseño automático de sistemas difusos. *PhD. dissertation, University of Granada, Spain*

Pomares, H., Rojas, I., Ortega, J., González, J., and Prieto, A. (2000). A Systematic Approach to a Self-Generating Fuzzy Rule-Table for Function Approximation. *IEEE Trans. Syst., Man, Cyber. Part B*, 30(3):431–447.

Punch, W. F., Goodman, E. D., Pei, M., Chia-Shun, L., Hovland, P., and Enbody, R. (1993). Further research on feature selection and classification using genetic algorithms. In *Proceedings of the Int. Conf. Genetic Algorithms*, pages 557–564.

Radtke, P. V. W., Wong, T., and Sabourin, R. (2005). A multi-objective memetic algorithm for intelligent feature extraction. In *EMO*, pages 767–781.

Raymer, M. L., Sanschagrin, P. C., Punch, W. F., Venkataraman, S., Goodman, E. D., and Kuhn, L. A. (1997). Predicting conserved water-mediated and polar ligand interactions in proteins using a k-nearest-neighbors genetic algorithm. *J. Mol. Biol.*, 265:445–464.

Reeves, C. R. (1993). Using Genetic Algorithms with Small Populations. In (Forrest, 1993b), pages 92–99.

Rojas, I., Anguita, M., Prieto, A., and Valenzuela, O. (1998). Analysis of the operators involved in the definition of the implication functions and in the fuzzy inference proccess. *International Journal of Approximate Reasoning*, 19:367–389.

Rojas, I., González, J., Cañas, A., Díaz, A. F., Rojas, F. J., and Rodriguez, M. (2000). Short-Term Prediction of Chaotic Time Series by Using RBF Network with Regression Weights. *Int. Journal of Neural Systems*, 10(5):353–364.

Rojas, I., González, J., Pomares, H., Rojas, F. J., Fernández, F. J., and Prieto, A. (2001). Multidimensional and Multideme Genetic Algorithms for the Construction of Fuzzy Systems. *International Journal of Approximate Reasoning*, 26:179–210.

S. Hettich, C. B. and Merz, C. (1998). UCI repository of machine learning databases.

Schaffer, J. D., editor (1989). *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann.

Schaffer, J. D. and Morishma, A. (1987). An Adaptive Crossover Mechanism for Genetic Algorithms. In Grefenstette, J. J., editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 36–40, Hillsdale, NJ. Lawrence Erlbaum Associates.

Schnecke, V. and Vornberger, O. (1996). An adaptive parallel algorithm for VLSI-layout optimization. In *IV Parallel Problem Solving from Nature*, pages 22–27.

Schwefel, H. P. (1981). *Numerical Optimization of Computer Models*. John Wiley, Chichester.

Sermer M, C.Ñ., Gare, D., Kenshole, A., Ritchie, J., Farine, D., and et al (1995). Impact of increasing carbohydrate intolerance on maternal-fetal outcomes in 3637 women without gestational diabetes. *Am J Obstet Gynecol*, 173:146–156.

Siedlecki, W. and Sklansky, J. (1989). A note on genetic algorithms for largescale feature selection. *Pattern Recognit. Lett.*, 10:335–347.

Spears, W. and De Jong, K. A. An Analysis of Multi-point Crossover. pages 301–315.

Spears, W. M., Jong, K. A. D., Bäck, T., Fogel, D. B., and de Garis, H. (1993). An overview of evolutionary computation. In Brazdil, P. B., editor, *Proceedings of the European Conference on Machine Learning (ECML-93)*, volume 667, pages 442–459, Vienna, Austria. Springer Verlag.

Srinivas, M. and Patnaik, L. M. (1994a). Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Trans. Syst. Man and Cyber.*, 24(4):656–666.

Srinivas, M. and Patnaik, L. M. (1994b). Genetic Algorithms: A Survey. *Computer*, 27(6):17–27.

Stender, J., editor (1893). *Parallel Genetic Algorithms*. IOS Press, Amsterdam.

Sutanto, E. L., Masson, J. D., and Warwick, K. (1997). Mean-Tracking Clustering Algorithm for Radial Basis Function Center Selection. *Int. J. Contr.*, 67(6):961–977.

Tate, D. M. and Smith, A. E. (1993). Genetic Optimization using a Penalty Function. In (Forrest, 1993b), pages 499–503.

Timm, H. and Kruse, R. (2002). A Modification to Improve Possibilistic Fuzzy Cluster Analysis. In *Proceedings of the 2002 IEEE International Conference on Fuzzy Systems*, volume 2, pages 1460 –1465.

Tryon, R. C. (1939). *Cluster Analysis*. Edward Brothers, Ann Arbor, MI.

Valdés, J. and Barton, A. (2000). A multi-objective optimization approach for training artificialneural networks. In *Sixth Brazilian Symposium on Neural Networks*.

Valdés, J. and Barton, A. (2006). Virtual Reality Visual Data Mining via Neural Networks Obtained from Multi-objective Evolutionary Optimization: Application to Geophysical Prospecting. In *International Joint Conference on Neural Networks, 2006. IJCNN '06*

van Veldhuizen, D. A., Zydallis, J. B., and Lamont, G. B. (2003). Considerations in engineering parallel multiobjective evolutionary algorithms. *IEEE Trans. Evolutionary Computation*, 7(2):144–173.

Wang, Maciejewski, Siegel, and Roychowdhury (1998). A comparative study of five parallel genetic algorithms using the traveling salesman problem. In *IPPS: 11th International Parallel Processing Symposium* IEEE Computer Society Press.

Watson, W., Soisson, A., and Harlass, F. (1988). Estimated weight of the term fetus. Accuracy of ultrasound vs. clinical examination. *J Reprod Med*, 33:369–371.

Weeks, J., Pitman, T., and 2d Spinnato, J. (1995). Fetal macrosomia: does antenatal prediction affect delivery route and birth outcome? *Am J Obstet Gynecol*, 173:1215–1219.

Whitehead, B. A. and Choate, T. D. (1996). Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction. *IEEE Trans. Neural Networks*, 7(4):869–880.

Wright, A. H. Genetic Algorithms for Real Parameter Optimization. pages 205–218.

Yao, X. (1999). *Evolutionary Computation: Theory and Applications* World Scientific.

Zamorski, M. A. and Biggs, W. (2001). Management of Suspected Fetal Macrosomia. *American Family Physician*, 63(2).

Zhang, J. and Leung, Y. (2004). Improved possibilistic C–means clustering algorithms. *IEEE Transactions on Fuzzy Systems*, 12:209–217.