XFHL: UNA HERRAMIENTA DE INDUCCIÓN DE SISTEMAS DIFUSOS JERÁRQUICOS

Sergio Cala Cordero[1] **Francisco José Moreno Velo**[2]

[1]Departamento de Tecnologías de la Información, Universidad de Huelva, sergio.cala@alu.uhu.es
[2]Departamento de Tecnologías de la Información, Universidad de Huelva, francisco.moreno@dti.uhu.es

Resumen

Este trabajo presenta una herramienta de generación automática de sistemas difusos jerárquicos llamada *Xfhl*. La herramienta toma como entrada un conjunto de datos de entrenamiento y busca la mejor descomposición jerárquica en términos de módulos difusos de dos entradas y una salida. El funcionamiento de *Xfhl* se basa en una búsqueda exhaustiva entre todas las posibles descomposiciones modulares. Para evaluar cada estructura se utiliza el error cometido por la estructura tras un proceso de optimización paramétrica. La herramienta está integrada en el entorno Xfuzzy de desarrollo de sistemas difusos.

Palabras Clave: Sistemas Difusos Jerárquicos, Software para SoftComputing.

1 INTRODUCCIÓN

A medida que los sistemas difusos se han afianzado como forma de modelar comportamientos no lineales el grado de complejidad de los problemas tratados ha ido aumentando. En los sistemas difusos estándar, para un problema con n variables de entrada descritas por m etiquetas lingüísticas, el número de reglas del sistema difuso resulta m^n. Este crecimiento exponencial provoca, en la práctica, que para un número de variables alto el número de reglas es tan grande que la interpretabilidad del sistema se hace imposible. Este problema, que no es exclusivo de los sistemas difusos, se conoce como *la maldición de la dimensionalidad* [2].

Una de las formas de disminuir el número de reglas y, por tanto, de aumentar la interpretabilidad, es descomponer el sistema difuso en módulos más simples, lo que se conoce como *sistemas difusos jerárquicos* (SDJ). Existen numerosas propuestas de diseño de este tipo de sistemas [13]. Algunas de ellas consisten en identificar partes comunes del conjunto de reglas y crear módulos que generen estas partes comunes, lo que se conoce como SDJ limpios [1][9]. En otras propuestas el nivel de jerarquía de cada módulo se refiere a un aumento de la granularidad de las variables [6].

El tipo más tradicional de SDJ es aquel en el que cada módulo es un sistema difuso completo que relaciona a un conjunto reducido de variables, que pueden ser variables de entrada del sistema global o variables internas generadas como salidas de otros módulos [11]. En este tipo de SDJ el número de reglas es

$$\text{Nº de reglas} = \sum_{i=1}^{L} m^{c_i} \qquad (1)$$

donde L es el numero de módulos y c_i el número de entradas del módulo i-ésimo. Si los módulos no comparten entradas el SDJ tiene forma de árbol. Si los módulos pueden compartir entradas el SDJ tiene forma de grafo dirigido acíclico. Centrándonos en sistemas con una única salida (MISO) y con forma de árbol, el número de entradas de los módulos debe cumplir

$$\sum_{i=1}^{L} c_i = n + L - 1 \qquad (2)$$

Si todos los módulos tienen el mismo número de entradas, c, el número de módulos del SDJ es

$$L = \frac{n-1}{c-1} \qquad (3)$$

y el número de reglas total del SDJ es entonces

$$\text{Nº de reglas} = \left(\frac{n-1}{c-1}\right) \cdot m^c \qquad (4)$$

lo que representa un crecimiento lineal del número de reglas en función del número de variables de entrada del

sistema frente al crecimiento exponencial que sufren los sistemas difusos estándar. El menor número de reglas totales se consigue para c=2, es decir, utilizando módulos de dos entradas. Es por eso que los SDJs más utilizados son los formados por este tipo de módulos, que se denominan habitualmente *fuzzy logic units* (FLUs). El uso de módulos de dos entradas tiene una ventaja adicional y es que su funcionamiento puede representarse mediante una superficie, lo que permite no sólo una interpretabilidad lingüística del módulo sino también una interpretabilidad gráfica del mismo.

Un aspecto muy importante de los SDJs es su capacidad de aproximación. Como es bien sabido, los sistemas difusos estándar son aproximadores universales [4]. Aunque existen propuestas de SDJs que han demostrado ser aproximadores universales [8][15], estas propuestas se refieren a sistemas difusos de tipo Takagi-Sugeno de orden muy alto (equivalente al número de entradas), por lo que el problema de la dimensionalidad no se resuelve sino que se desplaza del antecedente de las reglas a unos consecuentes de un alto grado de complejidad [16]. Por su parte, los sistemas difusos jerárquicos de tipo Mamdani no son aproximadores universales. En este caso el estudio se dirige a conocer cuales son las funciones que pueden ser aproximadas por un sistema difuso jerárquico de tipo Mamdani. En [17] se define el concepto de *función continua con estructura jerárquica separable* y se demuestra que un SDJ de tipo Mamdani puede aproximar estas funciones siempre que su estructura jerárquica corresponda a la de la función.

Para encontrar la estructura de un SDJ que permita aproximar una cierta función continua algunos autores han propuesto el uso de algoritmos genéticos [3][5]. En este trabajo se presenta una herramienta que realiza una búsqueda exhaustiva entre todas las descomposiciones modulares que puedan realizarse en base a módulos de dos entradas (FLUs). La estructura jerárquica generada por la herramienta es aquella que tras un proceso de optimización obtenga el menor error cuadrático sobre el conjunto de datos de entrenamiento.

2 ALGORITMO DE DESCOMPOSICIÓN MODULAR

La metodología propuesta se basa en realizar una búsqueda exhaustiva entre todas las posibles estructuras jerárquicas con n entradas formadas por módulos de dos entradas y una salida. Para generar todas estas estructuras jerárquicas se utiliza un algoritmo recursivo en el que las estructuras de n entradas se generan a partir de todas las estructuras de n-1 entradas.

El caso base del algoritmo es el formado por 2 entradas. En este caso sólo existe una estructura que es la formada por un único módulo, tal y como se muestra en la Figura 1.a.

Dada una estructura jerárquica con n entradas, es posible generar un conjunto de estructuras con n+1 entradas incluyendo una nueva FLU en cada variable de la estructura. La Figura 1 muestra un ejemplo de las estructuras que pueden generarse a partir del caso base. La estructura de la Figura 1.b se obtiene incluyendo la nueva FLU en la posición de la variable *i0*. La Figura 1.c se obtiene incluyendo la FLU en la posición de la variable *i1*. La estructura mostrada en la Figura 1.d se obtiene incluyendo la nueva FLU en la posición de la variable de salida.

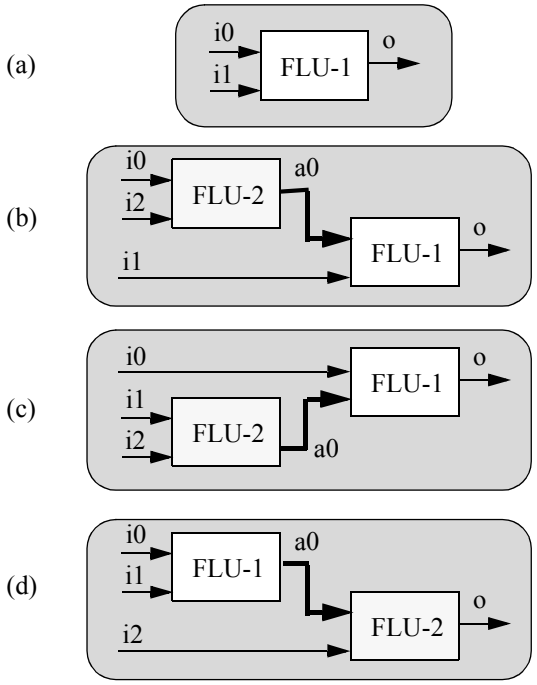

Figura 1: Algoritmo de descomposición modular para el caso de tres variables.

Teniendo en cuenta que cada estructura jerárquica con n entradas contiene n-1 FLUs, el número total de variables de la estructura es de 2n-1. Por tanto, el número de estructuras con n+1 entradas es

$$S(n+1) = (2n-1)\cdot S(n) = \prod_{i=1}^{n}(2i-1) \quad (5)$$

La Tabla 1 muestra los valores del número de estructuras jerárquicas diferentes que se pueden generar utilizando módulos de dos entradas. Como se puede apreciar, este número crece exponencialmente con respecto al número de entradas. Para problemas con un número de entradas menor de 7 u 8 la búsqueda exhaustiva de la mejor estructura resulta factible. A partir de este tamaño el número de estructuras resulta demasiado elevado, lo que conduce a la utilización de técnicas de búsqueda aleatoria.

Tabla 1: Número de estructuras jerárquicas

Número de entradas	Número de estructuras
2	1
3	3
4	15
5	105
6	945
7	10.395
8	135.135
9	2.027.025

3 OPTIMIZACIÓN PARAMÉTRICA DE SISTEMAS DIFUSOS JERÁRQUICOS

De entre todas las estructuras jerárquicas en las que se puede descomponer un sistema de *n* entradas, la metodología propuesta selecciona aquella que mejor consigue aproximarse al conjunto de datos de entrenamiento. Para evaluar la capacidad de aproximación de cada estructura se utilizan algoritmos de optimización paramétrica basados en descenso por gradiente. El uso de algoritmos de descenso por gradiente para ajustar sistemas difusos jerárquicos ya ha sido propuesto en [14]. En nuestro caso la propuesta es utilizar los algoritmos RPROP [12] y Levenberg-Marquardt [7], que presentan una velocidad de convergencia mucho mayor que el utilizado en dicho trabajo. Estos algoritmos y muchos otros se encuentran ya integrados en el entorno Xfuzzy [10], lo que ha facilitado en gran medida el desarrollo de la herramienta *Xfhl* presentada en este trabajo.

Para seleccionar la mejor estructura jerárquica se evalúa el error cuadrático medio (*ECM*) que produce la estructura sobre el conjunto de datos de entrenamiento, que viene dado por la expresión

$$ECM = \frac{1}{E} \cdot \sum_{e=1}^{E} (\tilde{y}_e - y(\tilde{x}_e))^2 \qquad (6)$$

donde E es el número de instancias del conjunto de datos, $\tilde{x}_e$ es el valor de las entradas de la instancia e-ésima, $\tilde{y}_e$ es el valor de la salida de la instancia e-ésima y la función $y(x)$ representa el comportamiento del sistema jerárquico.

Los parámetros de los que depende el comportamiento del sistema jerárquico corresponde al conjunto de parámetros que definen el comportamiento de cada FLU. Estos parámetros son los que definen las funciones de pertenencia de las variables de entrada del módulo y los relacionados con el cálculo de la salida. Para calcular la derivada del ECM respecto de cada uno de estos parámetros se utiliza la regla de la cadena. Por ejemplo, la derivada del ECM respecto de los parámetros del módulo FLU-1 de la Figura 2 viene dada por la expresión:

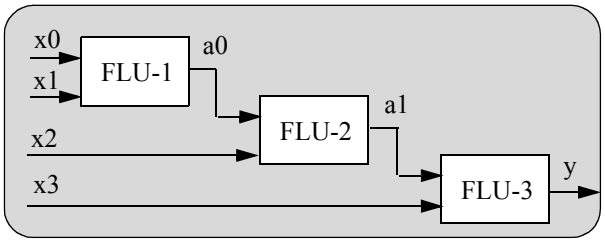

Figura 2: Ejemplo de estructura jerárquica

$$\frac{\partial ECM}{\partial p} = \sum_{e} \left(\frac{\partial ECM}{\partial y}\right) \cdot \left(\frac{\partial y}{\partial a1}\right) \cdot \left(\frac{\partial a1}{\partial a0}\right) \cdot \left(\frac{\partial a0}{\partial p}\right) \qquad (7)$$

La derivada del ECM respecto de la salida del sistema para el ejemplo e-ésimo es la siguiente

$$\frac{\partial ECM}{\partial y_e} = \frac{2}{E} \cdot (y_e - \tilde{y}_e) \qquad (8)$$

Para calcular el resto de derivadas parciales es necesario conocer el funcionamiento de las FLUs. Si el operador de conjunción es el producto y el método de concreción es la media difusa, el principio de transformación de la FLU es el siguiente:

$$y(x_1, x_2) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} \mu_{1i}(x_1) \cdot \mu_{2j}(x_2) \cdot c_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{m} \mu_{1i}(x_1) \cdot \mu_{2j}(x_2)} \qquad (9)$$

donde $\mu_{1i}(x_1)$ son las funciones de pertenencia asociadas a la variable x_1, $\mu_{2j}(x_2)$ son las funciones de pertenencia asociadas a la variable x_2, c_{ij} es el centroide de la conclusión de la regla "x_1 = label-i AND x_2 = label-j" y *m* es el número de etiquetas lingüísticas asociadas a cada variable. A partir de la expresión anterior se puede calcular la derivada parcial de la salida de una FLU respecto de los centroides de las reglas:

$$\frac{\partial y(x_1, x_2)}{\partial c_{ij}} = \frac{\mu_{1i}(x_1) \cdot \mu_{2j}(x_2)}{\sum_{i=1}^{m} \sum_{j=1}^{m} \mu_{1i}(x_1) \cdot \mu_{2j}(x_2)} \qquad (10)$$

La ec. 9 permite también calcular la derivada parcial de la salida de una FLU respecto al grado de activación de cada función de pertenencia.

$$\frac{\partial y(x_1,x_2)}{\partial \mu_{1i}(x_1)} = \frac{\left(\sum_{j=1}^{m} \mu_{2j}(x_2)\cdot c_{ij}\right)}{\left(\sum_{i=1}^{m}\sum_{j=1}^{m} \mu_{1i}(x_1)\cdot\mu_{2j}(x_2)\right)} - \frac{\left(\sum_{j=1}^{m} \mu_{2j}(x_2)\right)\cdot\left(\sum_{i=1}^{m}\sum_{j=1}^{m} \mu_{1i}(x_1)\cdot\mu_{2j}(x_2)\cdot c_{ij}\right)}{\left(\sum_{i=1}^{m}\sum_{j=1}^{m} \mu_{1i}(x_1)\cdot\mu_{2j}(x_2)\right)^2} \quad (11)$$

Por último hay que considerar la derivada parcial de la función de pertenencia respecto de sus parámetros de definición. Por ejemplo, para una función gaussiana

$$\mu(x,a,b) = \exp\left(-\left(\frac{x-a}{b}\right)^2\right) \quad (12)$$

las derivadas quedan

$$\begin{aligned}
\frac{\partial \mu(x,a,b)}{\partial x} &= -\frac{2}{b}\cdot\left(\frac{x-a}{b}\right)\cdot\exp\left(-\left(\frac{x-a}{b}\right)^2\right)\\
\frac{\partial \mu(x,a,b)}{\partial a} &= \frac{2}{b}\cdot\left(\frac{x-a}{b}\right)\cdot\exp\left(-\left(\frac{x-a}{b}\right)^2\right) \quad (13)\\
\frac{\partial \mu(x,a,b)}{\partial b} &= \frac{2}{b}\cdot\left(\frac{x-a}{b}\right)^2\cdot\exp\left(-\left(\frac{x-a}{b}\right)^2\right)
\end{aligned}$$

Con todas estas ecuaciones se puede calcular el gradiente del error cuadrático medio, tal y como se indica en la ec. 7 y utilizarlo en el algoritmo de optimización paramétrica seleccionado. En el caso de que el comportamiento de las FLUs se basen en otro operador de conjunción, otro método de concreción u otro tipo de funciones de pertenencia habría que recalcular las ecuaciones 8 a 13.

4 LA HERRAMIENTA XFHL

Xfhl es una nueva herramienta integrada en el entorno Xfuzzy de desarrollo de sistemas difusos. La herramienta *Xfhl* implementa la metodología de inducción de sistemas difusos jerárquicos comentada en los apartados anteriores. El funcionamiento de la herramienta se basa en generar todas las estructuras jerárquicas de *n* entradas y una salida que se pueden construir en base a módulos de dos entradas y una salida. La estructura seleccionada es la que presenta un menor error cuadrático medio tras un proceso de optimización paramétrica basado en un algoritmo de descenso por gradiente (RProp o Levenberg-Marquardt).

La generación del conjunto de estructuras jerárquicas para *n* variables de entrada se basa en el algoritmo presentado en la segunda sección. Para ello se utiliza un iterador basado en una pila de estructuras. Cada nivel de la pila corresponde a un número de entradas comenzado desde la estructura base con dos entradas. La cima de la pila contiene la estructura de *n* entradas que genera el iterador. En cada iteración se desapila la última estructura generada y se genera la siguiente estructura a partir de la estructura de *n*-1 entradas almacenada en la pila. Cuando se han generado todas las estructuras de n entradas correspondientes, se desapila la estructura de *n*-1 entradas y se genera una nueva estructura de *n*-1 entradas en función de la estructura de n-2 entradas almacenada en la pila, y así sucesivamente. Gracias a este iterador se evita tener que almacenar en memoria todas las estructuras jerárquicas a la vez.

Para evaluar cada estructura se realiza un proceso de optimización paramétrica sobre todos los parámetros de la estructura. Para ello se calcula el gradiente del error cuadrático medio tal y como se explica en la sección 3 y se aplica el algoritmo de descenso por gradiente elegido.

Puesto que la optimización y evaluación de una estructura se puede realizar de manera independiente del de las demás, el proceso de búsqueda es fácilmente paralelizable. La herramienta *Xfhl* ha sido programada para aprovechar las capacidades de paralelización de los sistemas actuales (arquitecturas multicore, multiprocesador y tecnologías de hyperthreading) de manera que el proceso de estudio de las diferentes estructuras jerárquicas se realiza en paralelo en varios hilos de ejecución.

La ventana principal de la herramienta se muestra en la Figura 3. En la parte superior de la ventana se encuentra el menú de configuración que permite seleccionar las dife-

Figura 3: Ventana principal de la herramienta Xfhl

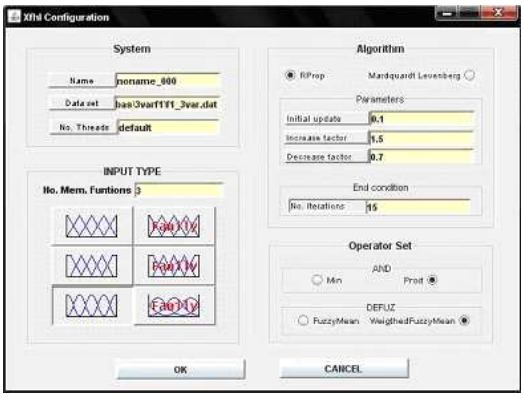

Figura 4: Ventana de configuración de la herramienta

rentes opciones de ejecución de la herramienta. Bajo la barra de menú se encuentra un área con el mejor sistema difuso encontrado hasta el momento y una barra de progreso que indica el porcentaje de estructuras estudiadas hasta el momento. Bajo esta barra se encuentra un panel en el que se muestra el estado de cada hilo de ejecución de la herramienta, que incluye para cada hilo la representación de la estructura en estudio y una barra de progreso del proceso de optimización. En la parte inferior de la ventana se encuentran los botones de control que permiten lanzar la ejecución, detenerla, almacenar el mejor resultado obtenido hasta el momento y cerrar la herramienta.

La configuración de la herramienta *Xfhl* se realiza por medio de la ventana mostrada en la Figura 4. Esta configuración requiere de la selección de los valores de los siguientes campos:

- Conjunto de datos de entrenamiento.
- Número de hilos concurrentes a ejecutar. El valor "Default" especifica el número de hilos igual al número de núcleos del sistema.
- Número de funciones de pertenencia de cada variable de entrada
- Estilo de las funciones de pertenencia de las variables de entrada. Pueden ser de los siguientes tipos: Free Triangles, Free Shouldered Triangles, Free Gaussians, Triangles Family, Shouldered-Triangular Family y B-Splines Family.
- Función asociada al operador de conjunción. Puede ser el mínimo o el producto.
- Algoritmo de concreción utilizado en los módulos. Puede ser la media difusa o la media difusa ponderada.
- El algoritmo de optimización paramétrica a usar para evaluar cada estructura. Se puede elegir entre RProp y Levenberg-Marquardt y se deben configurar sus parámetros de control y el número de iteraciones a realizar en el proceso de ajuste.

5 PRUEBAS

Para demostrar el funcionamiento de la herramienta *Xfhl* se ha considerado la siguiente función continua con 4 variables de entrada y estructura jerárquica separable:

$$f(w,x,y,z) = (w^2+z^2)\cdot\cos\left(\frac{\pi}{3}\cdot(x-2y)\right) \quad (14)$$

Para generar el conjunto de datos se ha considerado que las 4 variables están definidas en el intervalo [-1.0, 1.0] y se han generado todas las instancias posibles barriendo los valores de las variables en intervalos de 0.25, lo que produce un conjunto de datos con un total de 6561 instancias.

El proceso de ejecución de la herramienta *Xfhl* se ha desarrollado considerando una división de las variables en tres funciones de pertenencia gaussianas y seleccionando el producto como operador de conjunción y la media difusa ponderada como método de concreción. El ajuste paramétrico se ha llevado a cabo por medio de 40 iteraciones del algoritmo Rprop. La ejecución se ha realizado con dos hilos sobre una máquina con procesador Intel Core2 Duo de 2.0 GHz, mostrando una duración aproximada de 3 minutos. De entre las 15 posibles descomposiciones la mejor estructura encontrada, con un RMSE = 2.9%, es la mostrada en la Figura 5.a, que corresponde a la estrutura jerárquica de la función objetivo. Un ajuste paramétrico posterior de nuevo con RProp consigue una mejora de la aproximación hasta un RMSE=1.4%.

La Figura 5 muestra la representación gráfica del comportamiento de los diferentes módulos de la estructura jerárquica obtenida. Como se puede observar en la Figura 5.b, el comportamiento del módulo FLU-1 se aproxima al de la función coseno sobre una relación lineal de las variables *x* e *y*. (En realidad la función está desplazada ya que el rango de salida se encuentra entre 0 y 1). Un estudio más detallado permite incluso deducir esta relación. Con respecto al comportamiento del módulo FLU-2, la Figura 5.c permite también deducir que la relación entre las variables *w* y *z* corresponde a una parábola de revolución, es decir, a la función w^2+z^2 multiplicada por un factor negativo. En el caso del módulo FLU-3 el comportamiento es algo más difícil de interpretar ya que este módulo se encarga no sólo de calcular el producto entre las dos partes de la función, sino tambien de compensar el desplazamiento y los factores de escala de los módulos anteriores.

Agradecimientos

Este trabajo ha sido financiado parcialmente por los Proyectos de Investigación TEC2008-04920/TEC y TIN2008-06681-C06-06 de la CICYT y los Proyectos de Excelencia P07-TIC-03179 y P08-TIC-03674 de la Junta de Andalucía.

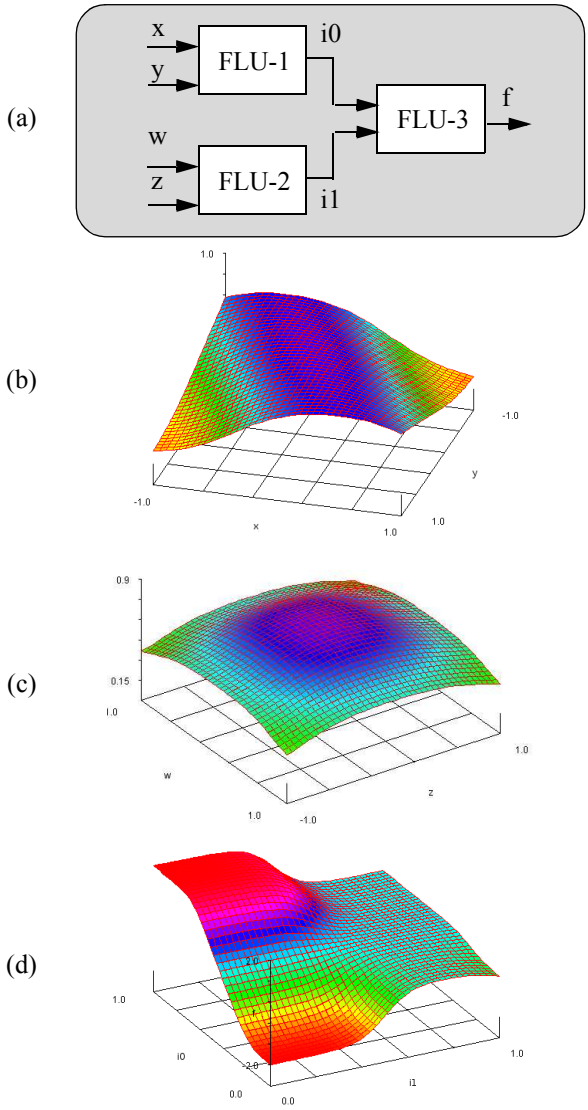

Figura 5: Resultado de *Xfhl* sobre la función de prueba. (a) estructura jerárquica encontrada; (b,c,d) comportamiento de los módulos FLU-1, FLU-2 y FLU-3.

Referencias

[1] S. Aja-Fernández, C. Arbeloa-López. Matrix modeling of hierarchical fuzzy systems, *IEEE Transactions on Fuzzy Systems*, Vol. 16, N. 3, Pág. 585-599, 2008.

[2] R. Bellman, *Adaptive Control Processes*, Princeton University Press, 1966.

[3] A.D. Benítez, J. casillas. Aprendizaje Evolutivo de Sistemas Difusos Jerárquicos en Serie, *Actas del VI Congreso Español sobre Metaheurística, Algoritmos Evolutivos y Bioinspirados (MAEB'09)*, Pág. 255-262, 2009.

[4] J.L. Castro. Fuzzy logic controllers are universal approximators, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 25, N. 4, Pág. 629-635, 1995.

[5] Y. Chen, B. Yang, A. Abraham, L. Peng, Automatic Design of Hierarchical Takagi-Sugeno Type Fuzzy Systems Using Evolutionary Algorithms, IEEE Transactions on Fuzzy Systems, Vol. 15; N. 3, Pág. 385-397, 2007.

[6] O. Cordón, F. Herrera, I. Zwir. A hierarchical knowledge-based environment for linguistic modeling: models and iterative methodology, *Fuzzy Sets and Systems*, Vol. 138, Pág 307-341, 2003.

[7] R. Fletcher. *Practical Methods of Optimization*, John Wiley & Sons, Ltd., 1986

[8] M.G. Joo, J.S. Lee, Universal approximation by hierarchical fuzzy system with constraints on the fuzzy rule, *Fuzzy Sets and Systems*, Vol. 130, N. 2, Pág. 175-188, 2002.

[9] M.L. Lee, H.Y. Chung, F.M. Yu. Modeling of hierarchical fuzzy systems, *Fuzzy Sets and Systems*, Vol. 138, Pág. 343-361, 2003.

[10] F.J. Moreno-Velo, I. Baturone, A. Barriga, S. Sanchez-Solano. Automatic tuning of complex fuzzy systems with Xfuzzy, *Fuzzy Sets and Systems*, Vol. 158, Pág. 2026-2038, 2007.

[11] G.V.S. Raju, J. Zhou, R.A. Kisner. Hierarchical fuzzy control, *International Journal of Control*, Vol. 54, Pág. 1201-1216, 1991.

[12] M. Riedmiller, H. Braun. RPROP: A fast and robust backpropagation learning strategy, *Proc. 4th Australian Conference on Neural Networks*, Pág 169-72, 1993.

[13] V. Torra. A Review of the Construction of hierarchical Fuzzy Systems, *International Journal of Intelligent Systems*, Vol. 17, Pág. 531-543, 2002.

[14] D. Wang, X.J. Zeng, J. Keane. Learning for Hierarchical Fuzzy Systems Based on the Gradient-Descent Method, *Proc. 2006 IEEE International Conference on Fuzzy Systems*, Pág. 92-99, 2006.

[15] L.X. Wang. Universal approximation by hierarchical fuzzy systems, *Fuzzy Sets and Systems*, Vol. 93, Pág. 223-230, 1998.

[16] L.X. Wang. Analysis and Design of Hierarchical Fuzzy Systems, *IEEE Transactions on Fuzzy Systems*, Vol. 7, N. 5, 1999.

[17] X.J. Zeng, J.A. Keane. Approximation Capabilities of Hierarchical Fuzzy Systems, *IEEE Transactions on Fuzzy Systems*, Vol. 13, N. 5, Pág. 659-672, 2005.