Aprendizaje Evolutivo de Sistemas Difusos Jerárquicos en Serie

Alicia D. Benítez y Jorge Casillas

Resumen— **Si intentamos dar solución a un problema donde el número de variables es alto mediante un sistema difuso convencional se produce un incremento exponencial del número de reglas, perdiendo así interpretabilidad. Una adecuada partición del espacio de variables con un sistema difuso jerárquico nos puede ayudar a paliar esta situación. Por ello, en este trabajo presentamos un Algoritmo Genético Multiobjetivo diseñado para el aprendizaje de Sistemas Difusos Jerárquicos en Serie. El algoritmo minimiza dos objetivos para el sistema: la raíz del error cuadrático medio (precisión) y el número de reglas (interpretabilidad). Se proponen operadores genéticos (uno de cruce y dos de mutación) adecuados para aprender este tipo de sistemas. Los estudios empíricos realizados indican que nuestro algoritmo obtiene buenos resultados en interpretabilidad y precisión con problemas en donde el número de variables es relativamente alto.**

Palabras clave—**Sistemas difusos jerárquicos, algoritmos genéticos multiobjetivo, regresión.**

I. Introducción

Los Sistemas Difusos son útiles para dar solución a problemas en los que la complejidad del proceso no permita un modelo matemático preciso para poder solucionarlo. Tienen numerosas aplicaciones en la actualidad como, por ejemplo, control, clasificación y extracción de conocimiento. Sin embargo, un sistema difuso convencional aplicado a problemas complejos puede hacer que se incremente exponencialmente el número de reglas atendiendo al número de entradas que reciba [1], [2]. Si tenemos n variables y k conjuntos difusos por variable, esto supone la necesidad de hasta n^k reglas para construir un sistema difuso completo y consecuentemente produce la ruptura del equilibrio entre la precisión y la interpretabilidad del mismo.

Para resolverlo, se han propuesto distintas líneas como selección de variables [3], [4], [5], [6] y reducción de reglas [7], [8], [9]. No obstante, cuando el número de variables se incrementa considerablemente, este tipo de reducciones no es suficiente para solventar el problema. Existe otra línea de investigación que trata esta problemática: los Sistemas Difusos Jerárquicos (SDJ). Un SDJ se compone de un conjunto de subsistemas difusos o módulos. Los módulos se encuentran concatenados entre sí, de forma que la salida de uno es la entrada de otro. Gracias a la descomposición del sistema difuso en un SDJ se produce una reducción de la complejidad de cada módulo, ya que se realiza una descomposición del espacio de entrada en otros subespacios de menor dimensión (cada variable de entrada se considera en un sólo nivel de jerarquía), siendo así el problema más fácil de abordar.

Existen distintos tipos de módulos:

Dpto. de Ciencias de la Computación e Inteligencia Artificial. Universidad de Granada. España, 18071. E-mail: aliciades@gmail.com, casillas@decsai.ugr.es

- *SISO (Single Inputs Single Outputs)*: Tiene una entrada y una salida.
- *MISO (Multiple Inputs Single Outputs)*: Posee varias entradas y una sola salida [10], [11]. Dentro de este tipo de módulos se encuentran los FLU (Fuzzy Logic Unit) que tienen solamente dos variables de entrada y una de salida. Es el que normalmente se suele considerar en la literatura [2], [12], [13], [14], [15], [16], [17], [18], [19], [20].
- *MIMO (Multiple Inputs Multiple Outputs)*: Tienen varias entradas y varias salidas [21].

Además de distinguir distintos tipos de módulos, también son posibles diferentes tipos de estructuras jerárquicas. Existen distintas clasificaciones [16], [22], [23], aunque la más general es la siguiente [20]:

- *SDJ en Serie (SDJS)*: Las entradas de un módulo son la salida de otros previos, junto con variables externas [13], [15], [19], [24].
- *SDJ en Paralelo*: Este sistema se organiza por capas. La primera capa está formada por un conjunto de módulos que reciben las variables de entrada. Cada variable está en un solo módulo. La salida de los módulos de la primera capa son la entrada de los módulos que componen la siguiente capa y así sucesivamente. También puede existir una operación de agregación para combinar las salidas de una capa [2], [15], [17], [18], [21].
- *SDJ Híbridos*: Este tipo de SDJ es una mezcla de los dos anteriores [11], [14], [25].

Otras líneas estudian la mejor forma de tratar las variables de salida de los módulos (mediante números difusos [16] y modelado de matrices [20]) y las reglas en diferentes niveles de jerarquía [17]. Además, en este campo se han aplicado algunas metaheurísticas con la finalidad de obtener un SDJ que consiga el mejor equilibrio entre precisión e interpretabilidad. Por ejemplo, se ha utilizado Evolución Diferencial [19] para encontrar las mejores funciones de pertenencia, estructuras de árbol que representan los SDJ y que evolucionan mediante Programación Genética y algoritmos PIPE (*Probabilistic Incremental Program Evolution*) [10], Algoritmos Genéticos [12], Sistemas de Hormigas [25] y el método de Gradiente Descendente [11] para aprender la estructura jerárquica del sistema difuso.

En este trabajo se propone un sistema difuso genético multiobjetivo con la finalidad de obtener la mejor distribución de variables de entrada y módulos pertenecientes a un SDJS para así poder hacer frente al problema del crecimiento exponencial del número de reglas en función del número de variables, lo cual perjudica seriamente la interpretabilidad del sistema difuso.

El trabajo se organiza en las siguientes secciones: en la

sección II, se realiza una descripción del algoritmo que se propone; la sección III, muestra el estudio empírico llevado a cabo; y la sección IV, relata las conclusiones y los trabajos futuros.

II. Algoritmo SDJSG

En esta sección describimos el algoritmo. Nuestra propuesta es un algoritmo de aprendizaje de estructuras de SDJS. La figura 1 muestra un ejemplo de la estructura de un SDJS.

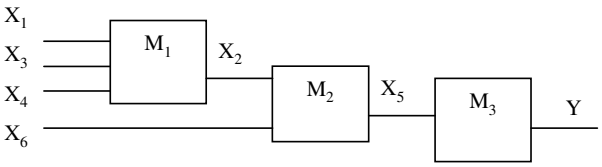

Fig. 1. Sistema difuso jerárquico en serie

Este tipo de sistemas está formado por módulos encadenados de forma que la salida de uno es una de las entradas del siguiente módulo. Cada módulo tiene su propia base de reglas, que permite deducir la variable de salida a partir de las variables de entrada al mismo. El SDJS de ejemplo consta de seis variables de entrada, de las cuales cuatro actúan como variables exógenas (variables externas al sistema: X_1, X_3, X_4 y X_6) y dos como endógenas (variables de interconexión de módulos: X_2 y X_5).

Nuestro objetivo con el algoritmo propuesto es obtener la mejor estructura difusa jerárquica en serie. Esto se consigue mediante el uso de un algoritmo genético multiobjetivo llamado SDJS Genético (SDJSG), en donde tenemos dos principales objetivos a minimizar: la raíz cuadrada del error cuadrático medio (RECM) y el número de reglas del sistema. Para este algoritmo se han diseñado tres operadores genéticos específicos para estructuras jerárquicas (un operador de cruce y dos operadores de mutación) que permiten a SDJSG orientarse en la búsqueda de la solución que más se adecúe a nuestro propósito.

El comportamiento del algoritmo se explica más detalladamente en las siguientes subsecciones.

A. Estructura del Algoritmo

El algoritmo de aprendizaje que proponemos en este trabajo, llamado *SDJS Genético (SDJSG)*, se ha diseñado de tal forma que dado un conjunto de variables y un conjunto de base de reglas, uno para cada módulo que compone el SDJS, sea capaz de encontrar la solución que proporcione el SDJS que tiene menor RECM y menor número de reglas. Su esquema se muestra en el algoritmo 1.

B. Esquema de Codificación

Para nuestro algoritmo, cada individuo de la población representa un SDJS. La codificación consiste en una concatenación de genes en los que cada uno de ellos tiene dos campos: la variable y un marcador que toma valores binarios (0 si es una variable exógena y 1 si es una variable endógena). En la figura 1, si nos centramos en el primer módulo, las variables X_1, X_3 y X_4 tendrían su marcador a 0 y a la variable X_2 le correspondería un marcador a 1. Así, el SDJS cromosoma sería como se muestra en la figura 2.

Algoritmo 1 Algoritmo SDJSG

Entrada: Tamaño de la población, probabilidad de cruce y probabilidad de mutación. Un conjunto de datos: $D = \{(x, y)|x \in \mathbb{R}^n, y \in \mathbb{R}^m\}$. Definición de las funciones de pertenencia.
Salida: Un conjunto Pareto con las mejores soluciones encontradas.
Inicialización(P);
Evaluación(P);
Mientras (*no se cumpla la condición de parada*) **hacer**
P1 ← Selección_Multiobjetivo(P);
P2 ← Cruce(P1);
P3 ← Mutación_Intercambio_Variables(P2);
P4 ← Mutación_Inserción_Variables(P3);
Evaluación(P4);
P ← Reemplazamiento_Multiobjetivo(P4);
Fin Mientras

Variable	X_1	X_3	X_4	X_2	X_6	X_5
Marcador	0	0	0	1	0	1

Fig. 2. Codificación del SDJS de la figura 1

Destacar que cuando una variable tiene su marcador a 1, además de indicar que es endógena (variable de interconexión de módulos), también simboliza la existencia de un módulo y que es la entrada del siguiente.

Todas las variables que tratamos son las variables de entrada del problema, es decir, no se crean nuevas variables endógenas, sino que en cualquier momento una variable de entrada del problema puede ser endógena o exógena. De esta forma, se puede considerar la correlación existente entre las variables del problema. Fijamos la longitud del cromosoma al número de variables de entrada del problema. El número máximo de módulos que puede existir es igual al número de variables de entrada. El caso de nivel de jerarquización más alta es cuando todos los módulos son SISO, donde todas las variables serían endógenas excepto la primera.

C. Inicialización

La población del algoritmo, de tamaño k, parte inicialmente de $k-1$ soluciones generadas aleatoriamente según una distribución uniforme, sin importar el orden de las variables ni el marcador que determina el tipo de variable. La solución restante se crea de forma que sólo existen variables exógenas y por lo tanto, un sólo módulo.

Una vez creada la población inicial de la que va a partir el algoritmo, el siguiente paso es el aprendizaje de la base de reglas de cada módulo que compone el SDJS. Usamos el algoritmo de Wang-Mendel (WM) [26] como método de aprendizaje con reglas de tipo Mamdani. Las funciones de pertenencia son triangulares y equidistribuidas.

D. Operador de Cruce

El operador de cruce se aplica según una probabilidad entre parejas de padres y proporciona explotación en el proceso de búsqueda del mejor SDJS. Este operador tiene como misión realizar intercambios de módulos entre dos padres para generar dos hijos. Para ello, el procedimiento elige un punto de cruce aleatorio distinto para cada padre. El primer hijo hereda del primer padre parte de la información hasta el punto de cruce. El resto de variables las hereda del segundo padre siguiendo el orden relativo de éstas en dicho padre. Lo mismo ocurre con el segundo hijo, pero al contrario: hereda la información del segundo padre hasta el punto de corte, para después completar el resto de variables atendiendo al orden existente en el primer padre. La figura 3 muestra un ejemplo de cómo se realiza el cruce.

E. Operadores de Mutación

El operador de mutación causa una alteración local en el cromosoma. Se aplica según una probabilidad por cromosoma y proporciona exploración en el proceso de búsqueda del mejor SDJS. Se han diseñado dos tipos de mutaciones:

- *Mutación por Intercambio:* Este operador consiste en realizar un intercambio de variables dentro de un mismo módulo. Para ello selecciona aleatoriamente un módulo con variables exógenas, es decir, con al menos dos entradas. De las variables exógenas existentes en el módulo se selecciona una aleatoriamente y se realiza el intercambio de ésta con la variable de salida del módulo. La figura 4 muestra un ejemplo de cómo se realiza este tipo de mutación.
- *Mutación por Inserción:* Tiene como función la inserción de variables en un módulo. En este operador se eligen dos módulos aleatoriamente del cromosoma: uno es el módulo origen, del cuál se extrae la variable, y otro el módulo destino en donde se inserta dicha variable. En el módulo origen se analizan las variables exógenas y se escoge una aleatoriamente.

En el caso de que no existan variables exógenas en el módulo de origen, y consecuentemente el módulo sea de una entrada y una salida (módulo *SISO*), se escoge la variable endógena (variable de interconexión de módulos) de entrada al módulo origen. En este caso, este módulo desaparece tras la mutación y la variable escogida se inserta como variable exógena en el módulo de destino.
Si el módulo origen y destino es el mismo, se escoge aleatoriamente una de las variables exógenas del módulo aleatoriamente, que pasa a ser la nueva variable de salida. A continuación de éste módulo se crea un nuevo módulo de tipo *SISO*, cuya entrada es la salida del módulo mutado y su salida es la antigua variable de salida antes de la transformación. La figura 5 ilustra la forma en que se realiza esta mutación.

F. Mecanismo de Inferencia

El mecanismo de inferencia que vamos a considerar es el Max-Min (la T-conorma del máximo como agregación y la T-norma del mínimo como operador relacional) y la

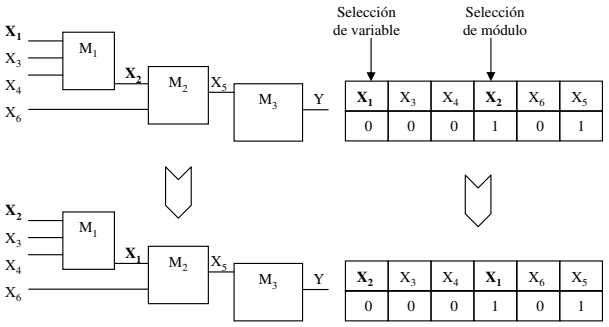

Fig. 4. Ilustración de la mutación de un individuo por Intercambio

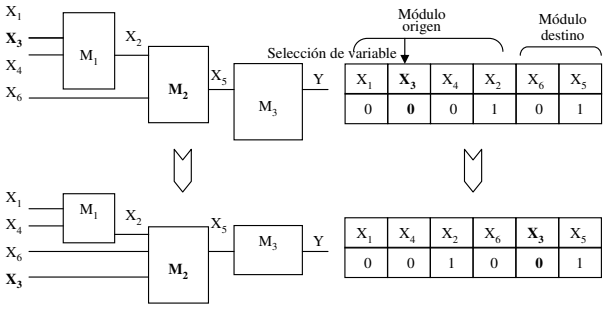

Fig. 5. Ilustración de la mutación de un individuo mediante Inserción

T-norma del mínimo como conjunción, la T-conorma del máximo como disyunción y el centro de gravedad para la defuzzificación.

G. Aprendizaje de la Base de Reglas

El aprendizaje de la base de reglas del SDJS se realiza de la siguiente forma: cada módulo aprende su base de reglas según las variables de entrada al mismo mediante WM [26]. La forma en la que se evalúa el sistema obtenido se hace a través de los ejemplos de entrenamiento escogiendo sólo los valores correspondientes a las variables exógenas del sistema para ese módulo. Dicho módulo, según la base de reglas aprendida, genera una salida (variable endógena) con un determinado valor. Este valor de salida será la entrada del siguiente módulo además de otras variables exógenas (si las tuviera).

H. Enfoque Multiobjetivo

En nuestro algoritmo usamos un enfoque multiobjetivo mediante el algoritmo NSGA-II [27]. Consideramos la distancia de *crowding* como métrica de espacio de la función objetivo. La selección de individuos se basa en el *ranking* de dominancia (o la distancia de *crowding* cuando ambas soluciones pertenecen al mismo frente). La distancia de *crowding* se normaliza para cada objetivo según los valores extremos de las soluciones contenidas en el frente que se esté analizando.

I. Funciones Objetivo

Consideramos dos funciones objetivo a minimizar con la finalidad de mejorar la precisión y la interpretabilidad del sistema:

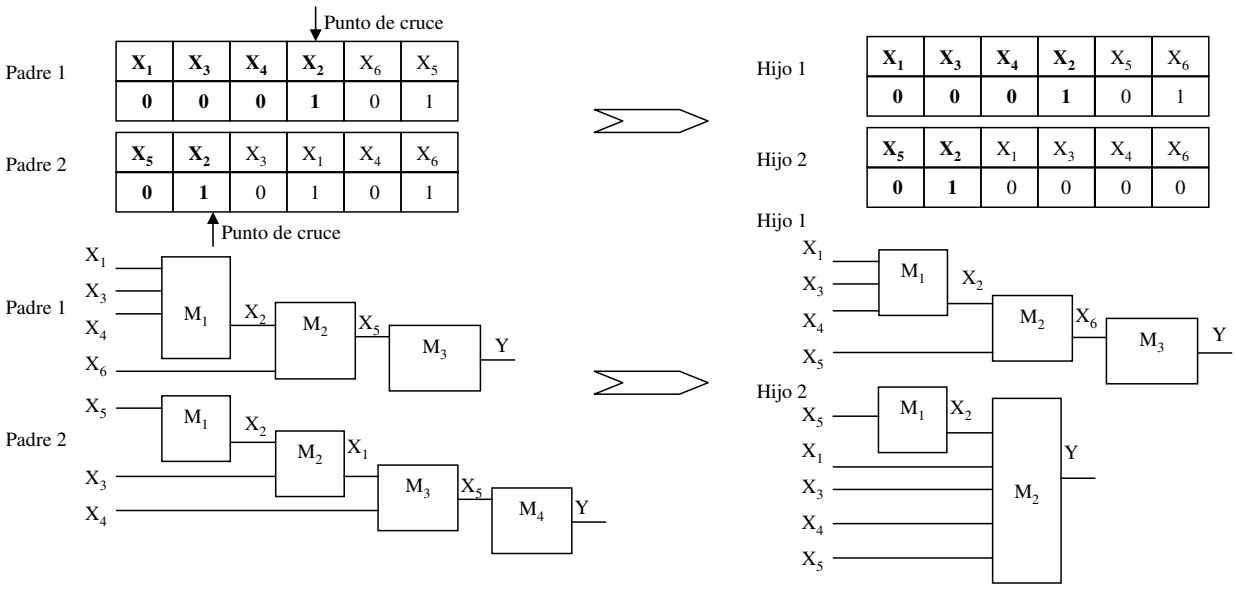

Fig. 3. Ilustración del cruce de dos cromosomas

▪ *Raíz del Error Cuadrático Medio (RECM):* Al minimizar este error, se mejora la precisión del sistema. Se calcula mediante la siguiente ecuación:

$$F_1(S) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(S(x^i) - y^i)^2} \qquad (1)$$

donde (x^i, y^i) es la i-ésima pareja de entrada-salida del conjunto de datos, S es la salida del SDJS que se va a evaluar y N es el tamaño del conjunto de datos.

▪ *Número de Reglas:* Su minimización permite una mayor interpretabilidad del sistema. Se calcula de la siguiente forma:

$$F_2(S) = r(S) \qquad (2)$$

donde $r()$ es el número de reglas del sistema difuso.

Aunque el número de módulos o el número de variables por módulo son también criterios relevantes para valorar la calidad del SDJS aprendido, resulta más intuitivo el número de reglas para validar la interpretación ya que condensa ambos criterios en un solo objetivo. Considerando el número de reglas, el número de módulos y de variables se regula de forma natural: a menor número de reglas, mayor número de módulos con pocas variables y viceversa.

III. Estudio Empírico

Esta sección presenta los resultados empíricos de los experimentos llevados a cabo con nuestro algoritmo. Consideramos ocho problemas de regresión (los valores de las variables son reales) y los comparamos con el método de WM. El objetivo de este experimento, más que analizar la precisión final obtenida, se centra en comprobar la reducción del número de reglas y por tanto su mejora de interpretabilidad, que se puede alcanzar sobre un método de aprendizaje inicial (WM en nuestro caso).

A. Problemas

Hemos considerado los siguientes ocho problemas de regresión:

▪ *Dee:* Consiste en la predicción del precio medio diario de la energía eléctrica TkWhe en España. El conjunto de datos consiste en una serie de datos reales de 2003 sobre el consumo diario en España de energía hidroeléctrica, eléctrica nuclear, carbón, combustible y gas natural. El conjunto de datos se encuentra disponible en la web[1].

▪ *Ele2:* Este problema hace referencia a la estimación del coste de mantenimiento de una red eléctrica de una línea de voltaje medio basado en la suma de las longitudes de todas las calles de la ciudad, el área total de la ciudad, el área ocupada por edificios y el suministro de energía en la ciudad [28]. La información se obtiene muestreando un modelo óptimo de red eléctrica para una ciudad. El conjunto de datos se encuentra disponible en la web de uno de los autores[2].

▪ *Laser:* Usa un conjunto de datos de laser de predicción de series temporales y competición de análisis del Instituto de Santa Fe (ISF) [29]. El conjunto de datos original de ISF consistían en 1000 observaciones de las fluctuaciones de un laser infrarrojo. Esta serie temporal de datos ha sido adaptada para regresión considerando los cuatro últimos valores como entrada y el siguiente valor como salida. El conjunto de datos está disponible en la web del proyecto KEEL[1].

▪ *Concrete:* El hormigón (*concrete*, en inglés), es el material más importante en la Ingeniería Civil. Se trata de predecir la fuerza de compresión del hormigón a partir de la edad y otros siete componentes de la mezcla. Es un problema con valores reales que se puede encontrar en UCI[3].

[1]KEEL: Knowledge extraction based on evolutionary learning. http://www.keel.es

[2]J. Casillas, FMLib: fuzzy modeling library. http://decsai.ugr.es/ casillas/FMLib/

[3]UCI Machine Learning Repository. Collection of regression data-

▪ *Census-House:* Este problema trata de la predicción de los precios medios de las casas a partir de los datos del censo estadounidense de 1990. Consta de dos problemas de 8 y 16 variables divididos en correlación entre las variables alta y baja. Nosotros hemos realizado la experimentación con el conjunto de datos de ocho variables en el que la correlación entre variables es menor. Este problema se puede obtener de Delve[4].
▪ *Elevators:* El conjunto de datos se obtiene del control de un avión F16. El objetivo es predecir la acción que se ha de realizar sobre el elevador del avión. Tiene 18 variables de entrada. Está disponible en la web[5].
▪ *Computer Activity:* Este problema consiste en predecir la actividad de un ordenador a partir de las métricas de funcionamiento del sistema (21 variables). Este conjunto de datos se puede descargar de la web [4].
▪ *Ailerons:* Es un problema de control de direcciones de un avión F16, aunque las variables están en un dominio distinto a *Elevators*. El estado del avión consta de 40 variables de entrada. El objetivo es predecir la acción de control en función del alerón del avión. Este problema está disponible en la web[5].

La tabla I reúne las principales características de los problemas considerados en la experimentación, siendo *#Entr.* el número de variables de entrada, *#Inst.* el número de instancias del problema y *#Term. Ling.* el número de términos lingüísticos triangulares por variable distribuidos uniformemente que se han considerado.

TABLA I
CONJUNTO DE DATOS CONSIDERADOS EN LA EXPERIMENTACIÓN

Problema	*#Entr.*	*#Inst.*	*#Term. Ling.*
Laser	4	993	5
Ele2	4	1066	5
Dee	6	365	5
Concrete	8	1030	5
Census-House 8H	8	22784	5
Elevators	18	16559	3
Computer Activity	21	8192	3
Ailerons	40	13750	3

Los experimentos han sido realizados mediante validación cruzada con cinco particiones de datos, es decir, el conjunto de datos está dividido en cinco subconjuntos de aproximadamente el mismo tamaño. Se construyen cinco particiones de datos, escogiendo en cada caso cuatro subconjuntos de entrenamiento de datos y dejando el quinto para prueba.

Usamos WM [26] para comparar nuestro algoritmo. WM es un algoritmo simple que, a pesar de no obtener resultados exactos, es una referencia tradicional en la comunidad de investigación de sistemas difusos.

Nuestro algoritmo se ha ejecutado con los siguientes parámetros: 1000 generaciones, un tamaño de población de 60 individuos, una probabilidad de cruce de 0,7 y una probabilidad de mutación por cromosoma de 0,2.

sets. http://archive.ics.uci.edu/ml/datasets.html
[4]Delve. Regression Datasets.
http://www.cs.utoronto.ca/ delve/data/datasets.html
[5]L. Torgo, Collection of regression datasets.
http://www.liacc.up.pt/ ltorgo/Regression/DataSets.html

B. Resultados Obtenidos

Las tablas II-IX recogen los resultados para cada problema. Las columnas $RECM_{entr}$ y $RECM_{prue}$ hacen referencia a la ecuación 1 de los valores de entrenamiento y prueba del conjunto de datos respectivamente. Las columnas *#M* y *#R* hacen alusión al número de módulos y de reglas obtenido, respectivamente. Todos estos datos son medias de las cinco particiones de datos consideradas. Nuestro algoritmo, al ser multiobjetivo, obtiene varias soluciones por ejecución. Por ello, del conjunto Pareto finalmente obtenido hemos seleccionado algunas soluciones representativas de datos atendiendo al $RECM_{entr}$: la solución más y menos precisa, el primer y tercer cuartil y la mediana. Los resultados mostrados son la media de estos valores sobre las cinco particiones.

TABLA II
RESULTADOS OBTENIDOS EN DEE

Método	$RECM_{entr}$	$RECM_{prue}$	*#M*	*#R*
WM	0,3757	0,4784	1,0	178,4
SDJSG más preciso	0,3757	0,4784	1,0	178,4
SDJSG 1^{er} cuartil	0,7127	0,6808	3,2	75,4
SDJSG mediana	0,8062	0,8180	3,6	58,3
SDJSG 3^{er} cuartil	0,8483	0,8400	4,4	38,1
SDJSG menos preciso	0,9712	0,9740	6,0	30,0

TABLA III
RESULTADOS OBTENIDOS EN ELE2

Método	$RECM_{entr}$	$RECM_{prue}$	*#M*	*#R*
WM	335,0694	335,7363	1,0	65,0
SDJSG más preciso	335,0694	335,7363	1,0	65,0
SDJSG 1^{er} cuartil	1152,1186	1173,7915	1,9	43,6
SDJSG mediana	1599,4771	1616,8408	2,9	25,0
SDJSG 3^{er} cuartil	1764,2282	1768,7122	3,4	22,1
SDJSG menos preciso	1893,0378	1893,1012	4,0	20,0

TABLA IV
RESULTADOS OBTENIDOS EN LASER

Método	$RECM_{entr}$	$RECM_{prue}$	*#M*	*#R*
WM	15,6751	16,0398	1,0	67,8
SDJSG más preciso	15,6751	16,0398	1,0	67,8
SDJSG 1^{er} cuartil	27,4700	27,7857	2,0	39,3
SDJSG mediana	33,1413	33,7958	2,2	31,4
SDJSG 3^{er} cuartil	37,6845	38,5980	3,0	24,5
SDJSG menos preciso	46,8611	46,0379	4,0	20,0

TABLA V
RESULTADOS OBTENIDOS EN CONCRETE

Método	$RECM_{entr}$	$RECM_{prue}$	*#M*	*#R*
WM	8,4128	9,7616	1,0	309,8
SDJSG más preciso	8,4128	9,7616	1,0	309,8
SDJSG 1^{er} cuartil	13,9531	14,2454	5,2	77,7
SDJSG mediana	14,5933	14,7124	6,1	57,7
SDJSG 3^{er} cuartil	15,3685	15,6135	6,9	48,2
SDJSG menos preciso	17,5686	17,5692	8,0	40,3

C. Análisis

El principal objetivo es obtener un equilibrio entre precisión e interpretabilidad en el sistema difuso. Como es

TABLA VI
RESULTADOS OBTENIDOS EN CENSUS-HOUSE 8H

Método	$RECM_{entr}$	$RECM_{prue}$	*#M*	*#R*
WM	48322,2973	49501,8248	1,0	713,4
SDJSG más preciso	48322,2973	49501,8248	1,0	713,4
SDJSG 1^{er} cuartil	50560,1673	50979,3469	4,1	239,9
SDJSG mediana	51780,1598	51852,6486	5,8	53,1
SDJSG 3^{er} cuartil	53304,7213	53338,6962	6,3	43,3
SDJSG menos preciso	68541,3049	68540,2835	7,0	37,6

TABLA VII
RESULTADOS OBTENIDOS EN ELEVATOR

Método	$RECM_{entr}$	$RECM_{prue}$	*#M*	*#R*
WM	0,0580	0,0582	1,0	511,2
SDJSG más preciso	0,0580	0,0582	1,0	511,2
SDJSG 1^{er} cuartil	0,0672	0,0672	9,2	72,8
SDJSG mediana	0,0730	0,0731	10,2	55,1
SDJSG 3^{er} cuartil	0,0886	0,0889	10,7	48,5
SDJSG menos preciso	0,1778	0,1775	11,7	44,2

TABLA VIII
RESULTADOS OBTENIDOS EN COMPUTER ACTIVITY

Método	$RECM_{entr}$	$RECM_{prue}$	*#M*	*#R*
WM	9,0398	9,0814	1,0	425,6
SDJSG más preciso	7,8449	7,8347	6,5	195,5
SDJSG 1^{er} cuartil	10,1052	10,2103	9,9	71,4
SDJSG mediana	10,5706	10,6644	10,9	63,1
SDJSG 3^{er} cuartil	11,2802	11,3641	11,5	59,1
SDJSG menos preciso	16,4782	16,6617	12,4	55,8

TABLA IX
RESULTADOS OBTENIDOS EN AILERONS

Método	$RECM_{entr}$	$RECM_{prue}$	*#M*	*#R*
WM	0,2540	0,2556	1,0	1080,6
SDJSG más preciso	0,2532	0,2555	2,9	956,6
SDJSG 1^{er} cuartil	0,4026	0,4011	16,7	126,2
SDJSG mediana	0,4772	0,4758	17,6	103,0
SDJSG 3^{er} cuartil	0,5973	0,5993	18,2	96,0
SDJSG menos preciso	0,9157	0,9145	19,0	91,0

sabido, a medida que se reduce el número de reglas en un sistema difuso se gana en interpretabilidad, pero se pierde en precisión y viceversa. Esto se puede observar claramente en los resultados obtenidos por cada uno de los problemas, tanto en problemas con menor número de variables como en los problemas con un número considerable de variables. Queremos destacar que la base de reglas se aprende individualmente para cada módulo, lo que hace que una solución donde #M=1 sea más precisa, por lo que el sistema no mejora globalmente.

En la tabla II se puede apreciar cómo las cinco soluciones extraídas del Pareto obtenido por nuestro algoritmo en el problema *Dee*, cuanto mayor es la precisión (SDJSG más preciso), el número de reglas es mayor, contamos con un sólo módulo y consecuentemente resulta poco interpretable. En el primer cuartil (SDJSG 1^{er} cuartil) podemos observar cómo se incrementa el $RECM_{entr}$ y el $RECM_{prue}$ con respecto a la solución más precisa, es decir, perdemos precisión. Sin embargo, hemos reducido el número de reglas a más de la mitad. Esto se debe al particionamiento del espacio de variables mediante módulos. Se puede apreciar que se pasa de tener un sistema difuso de un solo módulo en la solución más precisa a un SDJS con tres módulos en el primer cuartil. Si nos centramos en la solución menos precisa (SDJSG menos preciso), vemos que se tiene un SDJS en donde se ha reducido el número de reglas a la sexta parte de la solución más precisa. Con esto se ha ganado mucho en interpretabilidad, pero si miramos tanto el $RECM_{entr}$ como el $RECM_{prue}$, podemos apreciar claramente que casi se llega a triplicar. Si comparamos nuestro algoritmo en este problema con WM, claramente vemos que no lo mejora ya que iguala la solución. Si atendemos al resto de problemas en los que el número de variables es pequeño (cuatro variables), vemos que nuestro algoritmo tiene un comportamiento similar.

Sin embargo, si observamos los problemas en los que el número de variables es considerable (mayor o igual a ocho variables), nuestro algoritmo tiene otro comportamiento. Centrémonos en el problema *Census-House 8H* (tabla VI). Salta a la vista que nuestro algoritmo no es capaz de mejorar a WM, sólo lo iguala con la solución más precisa. Pero si consideramos detenidamente cada una de las soluciones obtenidas por nuestro algoritmo, notamos una mejoría del comportamiento: en el primer cuartil se empeora la solución más precisa tanto en $RECM_{entr}$ como en $RECM_{prue}$ pero pasa a tener 240 reglas y cuatro módulos. Básicamente el sistema ha perdido precisión, pero hay que destacar que esta pérdida es menos significativa que en los problemas pequeños. En efecto, en *Census-House 8H* conseguimos una reducción del número de reglas del 66 % respecto a la solución de WM en el primer cuartil a más de la mitad de la solución más precisa sin perder excesiva precisión. Si observamos la mediana, el tercer cuartil y la solución menos precisa, tampoco se llega a doblar ni el $RECM_{entr}$ ni el $RECM_{prue}$. Con respecto al número de reglas, la mediana ha reducido en un 93 % el número de reglas respecto a WM.

Esto nos puede hacer pensar que a medida que vamos incrementando el número de variables, nuestro algoritmo obtiene un SDJS más equilibrado. Vamos a observar qué ocurre con el problema *Computer Activity* (tabla VIII) de 21 variables para ver si podemos confirmar esta premisa. Si nos centramos en la solución más precisa, podemos apreciar que ha mejorado a WM: nuestro algoritmo ha mejorado tanto en $RECM_{entr}$ como en $RECM_{prue}$, pero además, hemos conseguido reducir el número de reglas a más de la mitad. Si atendemos a la solución menos precisa obtenida por nuestro algoritmo para este problema se puede comprobar que ni el $RECM_{entr}$ ni el $RECM_{prue}$ se llega a duplicar. Este sistema está formado por 12 módulos y 56 reglas. Esto supone una reducción considerable en el número de reglas.

Si observamos los resultados obtenidos en *Ailerons*, problema con 40 variables, podemos comprobar que la solución más precisa obtenida por nuestro algoritmo mejora a la de WM, distribuyendo en tres módulos las variables y reduciendo el número de reglas en un 10 %.

Como hemos visto en los resultados, en problemas pe-

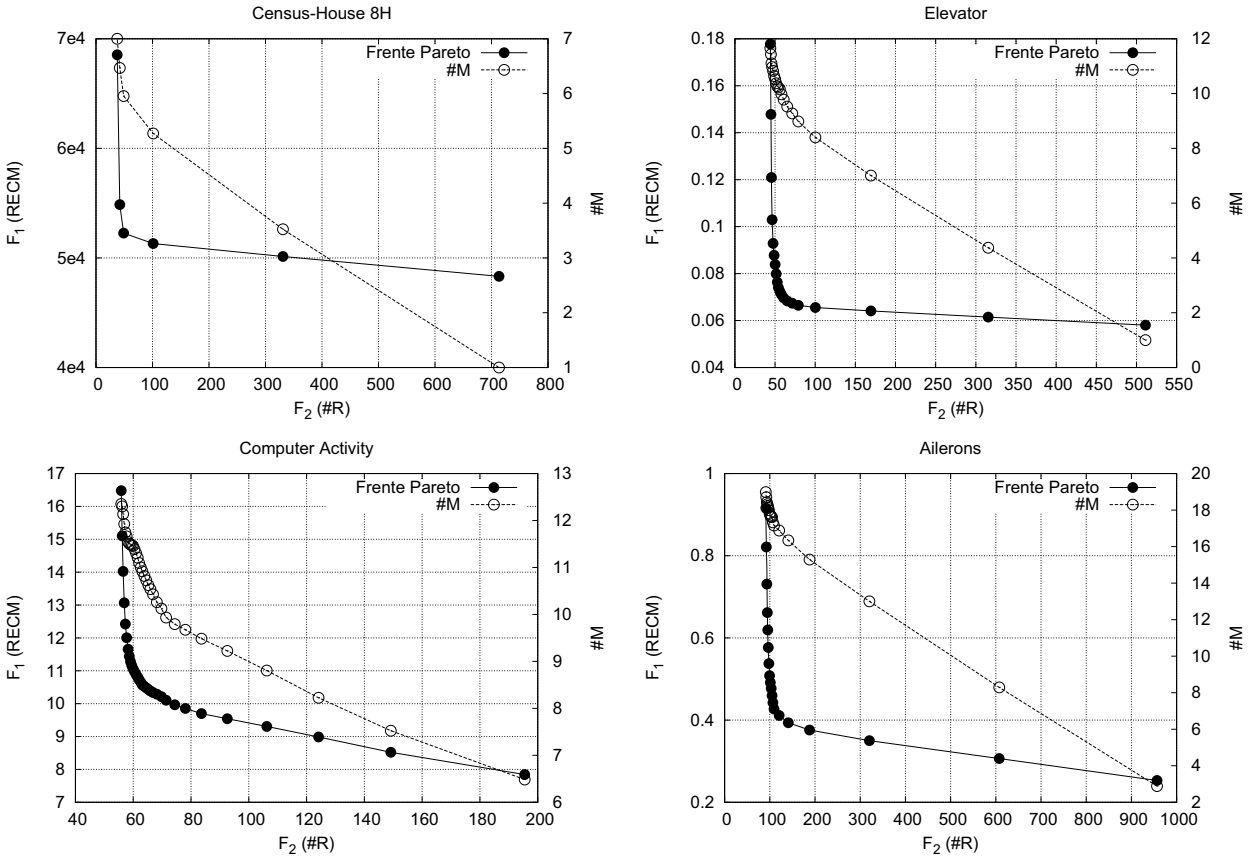

Fig. 6. Representación del frente Pareto y el número de módulos en función del número de reglas

queños se degrada rápidamente la precisión al reducir el número de reglas mientras que en los problemas grandes no sucede así. Se puede deber a que en los problemas con menor número de variables la correlación entre ellas es baja y todas son relevantes, sin embargo en los problemas con mayor número de variables existe mayor dependencia y una estructura jerárquica ayuda a reducir la complejidad del modelo difuso aprendido.

Esto afirma la hipótesis de que un particionamiento adecuado del espacio de las variables puede llegar a paliar el problema del crecimiento exponencial del número de reglas en función del número de variables.

Otro aspecto a destacar es la relación de dependencia entre el número de módulos y reglas. La figura 6 muestra cuatro representaciones gráficas del frente Pareto y el número de módulos (*#M*) en función del número de reglas de cuatro de los problemas considerados: *Census-House 8H*, *Elevator*, *Computer Activity* y *Ailerons*. En ambas representaciones se puede comprobar esa dependencia entre el número de módulos en función del número de reglas que hemos comentado. Es fácilmente apreciable que ambas son magnitudes inversamente proporcionales, ya que a medida que el número de reglas se incrementa, el número de módulos disminuye.

Para finalizar, queremos mostrar como ejemplo la estructura jerárquica de un SDJS obtenida por nuestro algoritmo. Esta estructura jerárquica en serie queda reflejada en la figura 7. Se trata de una de las soluciones del Pareto generado en el problema *Census-House 8H* para una partición de datos. Como se puede apreciar, aunque el $RECM_{prue}$ se ha incrementado alrededor de un 5 %, la reducción del número de reglas es de aproximadamente de un 90 %.

IV. Conclusiones

Hemos propuesto un algoritmo multiobjetivo para el aprendizaje de SDJS con el objetivo de paliar el crecimiento exponencial del número de reglas cuando aumenta el número de variables de entrada. Este algoritmo evoluciona una población de soluciones, donde cada individuo es un SDJS, intentando mantener el equilibrio entre precisión e interpretabilidad. El algoritmo particiona el conjunto de variables en módulos. Hemos demostrado que este particionamiento mediante un tipo de estructuras jerárquicas en serie puede ser bastante fructífero en problemas cuyo número de variables es elevado.

Como trabajos futuros, nos proponemos añadir mecanismos que permitan una mayor precisión, como por ejemplo la eliminación de variables, la realización de un aprendizaje global de la base de reglas, un estudio más detallado de la interpretabilidad de cada regla del sistema y una extensión del algoritmo para el apredizaje de estructuras paralelas e híbridas.

Agradecimientos

Este trabajo ha sido parcialmente subvencionado por el Ministerio de Educación y Ciencia (proyecto

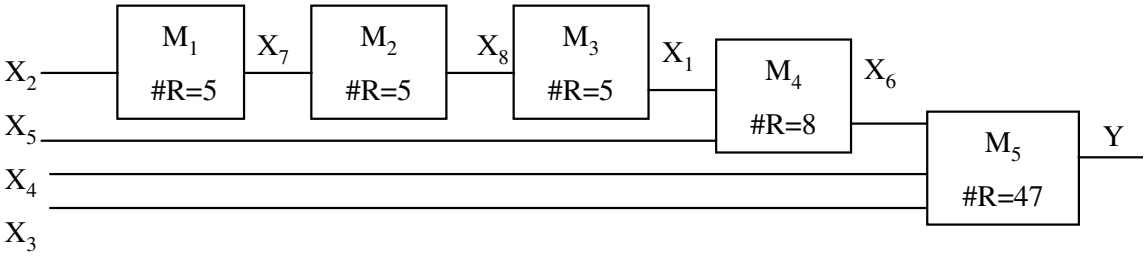

Fig. 7. Solución del Pareto óptimo encontrado por SDJSG en una partición de *Census-House 8H*. $RECM_{entr}$ = 51108; $RECM_{prue}$ = 52814; #M = 5; #R = 70. Resultados con WM: $RECM_{entr}$ = 47677; $RECM_{prue}$ = 50495; #M = 1; #R = 709

TIN2005-08386-C05-01) y por la Junta de Andalucía (proyecto P07-TIC-3185).

Referencias

[1] G.V.S. Raju, J. Zhou, and R.A. Kisner, "Hierarchical fuzzy control," *International Journal Control*, vol. 54, no. 5, pp. 1201–1216, 1991.

[2] M.G. Joo and J.S. Lee, "Hierarchical fuzzy control scheme using structured takagi-sugeno type fuzzy inference," in *Proceedings of IEEE International Fuzzy Systems Conference*, Seoul, Korea, 1999, pp. 78–83.

[3] T.P. Hong and J.B. Chen, "Finding relevant attributes and membership functions," *Fuzzy Sets and Systems*, vol. 103, no. 3, pp. 389–404, 1999.

[4] Y. Jin, "Fuzzy modeling of high-dimensional systems: complexity reduction and interpretability improvement," *IEEE Transactions in Fuzzy Systems*, vol. 8, no. 2, pp. 212–221, 2000.

[5] H.M. Lee, C.M. Chen, J.M. Chen, and Y.L. Jou, "An efficient fuzzy classifier with feature selection based on fuzzy entropy," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 31, no. 3, pp. 426–432, 2001.

[6] A. González and R. Pérez, "Selection of relevant features in a fuzzy genetic learning algorithm," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 31, no. 3, pp. 417–425, 2001.

[7] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Transactions in Fuzzy Systems*, vol. 3, no. 3, pp. 260–270, 1995.

[8] T. Taniguchi, K. Tanaka, H. Ohtake, and H.O. Wang, "Model construction, rule reduction, and robust compensation for generalized form of takagi-sugeno fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 9, no. 4, pp. 525–538, 2001.

[9] J. Casillas, O. Cordón, M.J. del Jesús, and F. Herrera, "Genetic tuning of fuzzy rule deep structures preserving interpretability and its interaction with fuzzy rule set reduction," *IEEE Transactions in Fuzzy Systems*, vol. 13, no. 1, pp. 13–29, 2005.

[10] Y. Chen, B. Yang, A. Abraham, and L. Peng, "Automatic design of hierarchical takagi-sugeno type fuzzy systems using evolutionary algorithms," *IEEE Transactions in Fuzzy Systems*, vol. 15, no. 3, pp. 385–397, 2003.

[11] D. Wang, X.J. Zeng, and J.A. Keane, "Learning for hierarchical fuzzy systems based on gradient-descent method," in *Proceedings of IEEE International Conference on Fuzzy Systems*, 2006, pp. 92–99.

[12] K. Shimojima, T. Fukuda, and Y. Hasegawa, "Self-tuning fuzzy modeling with adaptive membership function, rules, and hierarchical structure based on genetic algorithm," *Fuzzy Sets and Systems*, vol. 71, pp. 295–309, 1995.

[13] L.X. Wang, "Universal approximation by hierarchical fuzzy systems," *Fuzzy Sets and Systems*, vol. 93, pp. 223–230, 1998.

[14] M.G. Joo and J.S. Lee, "Universal approximation by hierarchical fuzzy system with constrains on the fuzzy rule," *Fuzzy Sets and Systems*, vol. 130, pp. 175–188, 2002.

[15] M.L. Lee, H.Y. Chung, and F.M. Yu, "Modeling of hierarchical fuzzy systems," *Fuzzy Sets and Systems*, vol. 138, pp. 343–361, 2003.

[16] A.E. Gaweda and R. Scherer, "Fuzzy number-based hierarchical fuzzy system," *Lecture Notes in Computer Science*, vol. 3070, pp. 302–307, 2004.

[17] T.M. Jelleli and A.M. Alimi, "Improved hierarchical fuzzy control scheme," *Adaptive and Natural Computing*, vol. 1, pp. 128–131, 2005.

[18] X. Zhang and N. Zhang, "Universal approximation of binary-tree hierarchical fuzzy system with typical flus," *Lecture Notes in Computer Science*, vol. 4114, pp. 177–182, 2006.

[19] F. Cheong, "A hierarchical fuzzy system with high input dimensions for forecasting foreign exchange rates," *IEEE Congress on Evolutionary Computation, CEC*, pp. 1642–1647, 2007.

[20] S. Aja-Fernández and C. Alberola-López, "Matriz modeling of hierarchical fuzzy systems," *IEEE Transactions in Fuzzy Systems*, vol. 16, no. 3, pp. 585–599, 2008.

[21] P. Salgado, "Rule generation for hierarchical collaborative fuzzy system," *Applied Mathematical Modelling, Science Direct*, vol. 32, pp. 1159–1178, 2008.

[22] J.C. Duan and F.L. Chung, "Multilevel fuzzy relational systems: structure and identification," *Soft Computing*, vol. 6, pp. 71–86, 2002.

[23] V. Torra, "A review of the construction of hierarchical fuzzy systems," *International Journal of Intelligent Systems*, vol. 17, pp. 531–543, 2002.

[24] L.X. Wang, "Analysis and design of hierarchical fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 5, pp. 617–624, 1999.

[25] Y. Chen, J. Dong, and B. Yang, "Automatic design of hierarchical ts-fs model using ant programming and pso algorithm," in *Proceedings 12th International Conference on Artificial Intelligence, Methodology, Systems and Applications, Lecture Notes on Artificial Inteligence, LNAI 3192*, C. Bussler and D. Fensel, Eds., 2004, pp. 285–294.

[26] L.X. Wang and J. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1414–1427, 1992.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarevian, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[28] O. Cordón, F. Herrera, and L. Sánchez, "Solving electrical distribution problems using hybrid evolutionary data analysis techniques," *Applied Intelligence*, vol. 10, no. 1, pp. 5–24, 1999.

[29] A.S. Weigend and N.A. Gershenfeld, Eds., *Time series prediction: forecasting the future and understanding the past*, 1992 NATO Advanced Research Workshop on Comparative Time Series Analysis, Addison-Wesley, Santa Fe, New Mexico, 1993.