EVFUZZYSYSTEM: EVOLUCIÓN DE SISTEMAS DIFUSOS PARA PROBLEMAS DE REGRESIÓN MULTI-DIMENSIONALES

M. Martínez-Ballesteros[1] **Víctor M. Rivas**[2]

[1] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, mariamartinez@us.es
[2] Departamento de Informática, Universidad de Jaén,vrivas@ujaen.es

Resumen

Este trabajo presenta EvFuzzySystem, un método evolutivo que permite el diseño completo de sistemas de lógica difusa, generando de forma simultánea funciones miembro y conjunto de reglas apropiados. EvFuzzySystem representa la extensión del método diseñado inicialmente para la resolución de problemas definidos por dos entradas y una salida. Esta extensión no ha sido trivial desde el punto de vista computacional. Los resultados muestran que puede ser aplicado a problemas de regresión compuestos de cualquier número de entradas y que los resultados obtenidos son comparables a los de métodos ya existentes.

Palabras Clave: Sistemas difusos, Algoritmos genéticos, Algoritmos evolutivos, Métodos híbridos, Función de aproximación.

1 INTRODUCCIÓN

En 2003, Rivas et al. [1] presentó EvFuzzy, un algoritmo evolutivo (AE) diseñado para evolucionar sistemas de lógica difusa (SLD) para problemas de dos dimensiones. El algoritmo evolutivo diseñaba simultáneamente tanto las funciones miembro como el conjunto de reglas necesarias para construir cada SLD. La extensión a problemas multidimensionales se mencionaba ya como una de las propuestas de futuro, siendo no obstante una nueva característica del algoritmo no trivial al representar cambios profundos tanto en la representación usada para los individuos como en los mecanismos para manejar el modelo, principalmente, los operadores genéticos diseñados específicamente para el algoritmo.

En este trabajo se presenta el nuevo algoritmo EvFuzzySystem, el cual puede ser usado para diseñar SLD para problemas de regresión de un número indeterminado de entradas. Desde el punto de vista computacional la extensión realizada no es trivial, por lo que se han debido de usar nuevas estructuras de datos más adecuadas para representar los individuos y, lo que es más importante, se han desarrollado nuevos operadores genéticos para evolucionar dichas estructuras.

2 EL ALGORITMO EVOLUTIVO

Las siguientes subsecciones detallan las arquitecturas tanto del SLD como del AE que lo optimiza, así como la función fitness y los operadores genéticos usados.

2.1 CODIFICACIÓN DE LOS INDIVIDUOS: EL SISTEMA DE LÓGICA DIFUSA

El esquema de representación elegido sigue el enfoque Pittsburg, siendo cada cromosoma un conjunto completo de base de reglas difusas. El cromosoma codifica tanto los antecedentes como los consecuentes, siendo las funciones de pertenencia de los términos lingüísticos funciones triangulares, como muestra la figura 1.

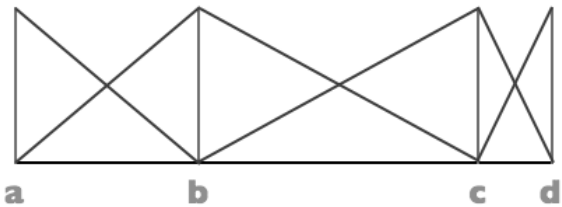

Figura 1: Conjuntos difusos de una variable del algoritmo implementado.

Cada cromosoma se implementa como dos estructuras; la primera representa el conjunto de n variables de entrada, cada una con número m_i de términos (pudiendo cada variable tener un numero distinto de términos); la segunda contiene el conjunto de consecuentes, siendo por tanto su tamaño igual al producto de los términos de cada una de las variables de entrada. Cada gen almacena directamente un valor real, no una codificación binaria del mismo.

Para cada variable de entrada se almacenan los centros de las particiones triangulares de las funciones de pertenencia. El extremo de una función de pertenencia, se hace coincidir con el centro de la siguiente. El primer y último centro corresponden al valor mínimo y máximo establecidos respectivamente para dicha variable. Se han fijado en 3 el número mínimo de términos que debe tener cada variable, mientras que el número máximo es uno de los parámetros de entrada del algoritmo evolutivo.

Por su parte, tal y como se hizo en [1], los valores del conjunto de consecuentes son generados aleatoriamente entre un valor mínimo y un máximo dados sin conservar ningún tipo de orden entre los mismos. Para obtener el valor de los consecuentes, no es necesario un método de defuzzificación ya que cada regla tiene un valor real exacto para los consecuentes (valor denominado CRISP o no-difuso).

Para obtener la posición del consecuente correspondiente a una combinación de n valores de entrada, se utiliza la siguiente ecuación:

$$F(v_1, ..., v_n) = \sum_{i=1}^{n} (v_i \prod_{j=i+1}^{n} n_j) \quad (1)$$

El SLD así creado, genera reglas de la siguiente forma:

SI X_1 es $MV[1, i_1]$ y X_2 es $MV[2, i_2]$ y ... y X_n es $MV[n, i_n]$ ENTONCES Z es $MC[F(i_1, i_2, ..., i_n)]$

Siendo i_n la posición del término correspondiente a la variable X_n.

2.2 PROCESO EVOLUTIVO

El algoritmo evolutivo usado utiliza selección elitista, mantiene el tamaño de población constante y aplica una serie de operadores específicamente diseñados para trabajar con estos cromosomas, y que se explican en los siguientes apartados . He aquí los principales pasos del algoritmo:

1. Crear y evaluar la primera población; constará de p individuos generados aleatoriamente de tamaño aleatorio.
2. Seleccionar una subpoblación con los q mejores individuos. A ellos se aplicarán los operadores evolutivos
3. Repetir $p-q$ veces: seleccionar un individuo de la subpoblacin mediante el método de la ruleta; duplicarlo; aplicarle uno de los operadores; evaluar el nuevo individuo.
4. Generar la nueva poblacin uniendo a la subpoblación de q individuos los $p-q$ nuevos creados.
5. Repetir desde el paso 2 hasta alcanzar el número máximo de generaciones deseado.
6. Finalmente, el mejor SLD encontrado es el mejor individuo de la última generación.

2.3 GENERACIÓN DE LA POBLACIÓN INICIAL

La población inicial se genera de forma aleatoria tanto el conjunto de variables de entrada como el conjunto de valores de la variable de salida, así como el número de términos para cada variable de entrada según un máximo dado.

2.4 OPERADORES GENÉTICOS

Se han diseñado operadores genéticos específicos para trabajar con las estructuras utilizadas. Los operadores permiten tanto el intercambio de información entre los individuos como la modificación de los mismos. Además, cada operador genera un SLD válido respetando los valores mínimo y máximo de cada una de las variables de entrada, el número mínimo y máximo de términos lingisticos y el orden en el que están definidos los centroides de las funciones de pertenencia.

2.4.1 Recombinación de consecuentes

Este operador intercambia valores entre un SLD y otro en un slo sentido. Teniendo en cuenta que estos dos SLD no necesariamente tienen la mismas dimensiones, la recombinación no puede ser realizada de manera trivial.

El operador trabaja de la siguiente manera:

Aleatoriamente se eligen un SLD para ser modificado (M_r, o SLD receptor), y otro que proporcionará los genes para ser recombinados (M_d, o SLD donante).

De la estructura de consecuentes correspondiente a M_r, se seleccionan aleatoriamente un bloque aleatorio de celdas adyacentes. Para especificar este bloque basta seleccionar dos celdas que consideraremos los extremos o esquinas del bloque. Sean dichas celdas

$M_r[r_1]$ y $M_r[r_2]$. Imaginando espacialmente la estructura multidimensional que constituyen los consecuentes, las celdas a modificar con valores procedentes de M_d, serán las incluidas en el bloque espacial que forman las dos celdas seleccionadas aleatoriamente, como extremos de bloque, es decir, aquellas celdas del consecuente cuya posición al ser decodificada (aplicándole una función de decodificación para conocer la combinación de posiciones de términos pertenecientes a las variables) contiene las posiciones de los términos de cada variable dentro del rango establecido por las posiciones elegidas aleatoriamente como esquinas del bloque.

Para cada celda $M_r[r_i]$, así decodificada, para saber la posición del término correspondiente en cada variable a la que se refiere la celda en cuestión, $M_r[r_{1i}, r_{2j}, ..., r_{nm}]$, (donde r_{1i} representa los valores r_{11} a r_{12}, r_{2j} los valores r_{21} a r_{22} y r_{nm} los valores r_{n1} a r_{n2}) el operador hace lo siguiente:

Del conjunto de variables de entrada perteneciente al SLD donante, M_d, de cada variable se selecciona el término cuyo valor se acerca más al término de la misma variable correspondiente al SLD receptor M_r.

Finalmente, se modifica el valor almacenado en $M_r[r_i]$, por el almacenado en la celda perteneciente a M_d que resulta tras codificar las posiciones de los términos de cada variable obtenidos en el punto anterior.

2.4.2 Adición de funciones de pertenencia (Mutador de incremento de tamaño)

Este operador modifica la estructura de un SLD añadiendo una función de pertenencia a las variables de entrada. Una vez seleccionado el individuo a modificar, se selecciona aleatoriamente una de las variables de entrada. A continuación, se elige aleatoriamente la posición del nuevo término dentro de la variable seleccionada y se genera un valor aleatorio para el nuevo término entre los valores de los términos anterior y posterior. Por último, dentro del conjunto de consecuentes se insertarán nuevos valores aleatorios correspondientes a la combinación del nuevo término con todos los términos del resto de variables de entrada.

2.4.3 Eliminación de funciones de pertenencia (Mutador de decremento de tamaño)

Este operador modifica la estructura de un SLD decrementando el número de funciones de pertenencia en las variables de entrada. El operador, una vez seleccionado el SLD a modificar, destruye uno de los términos de una de las variables de entrada, así como todos los consecuentes asociados a dicho término.

2.4.4 Modificación de un valor centroide (Mutador de precedentes)

Este operador modifica el valor almacenado en un término de una de las variables del conjunto de variables de entrada elegidas al azar. Si el término elegido para cambiar es el i-ésimo, su valor se generará aleatoriamente entre los valores $i-1$ e $i+1$.

2.4.5 Modificación de un valor consecuente (Mutador de consecuentes)

Este operador modifica el valor almacenado en un consecuente del conjunto de consecuentes de un determinado SLD. El consecuente es seleccionado al azar y su valor varía dento del rango delimitado para los consecuentes.

2.5 Función fitness

El fitness asignado al SLD es el valor correspondiente a la inversa del Error Cuadrático Medio Normalizado de las salidas correctas conocidas a las salidas producidas por el SLD, calculadas usando el conjunto de entrenamiento.

$$F_j = \frac{N}{\sum_{i=1}^{N}((z_i - z_{ij})/(z_{max} - z_{min}))^2)} \qquad (2)$$

Dónde F_j es el fitness para el individuo número j, N es el número de ejemplos del conjunto de entrenamiento (training set), z_i es la salida correcta, y z_{ij} es la salida proporcionada por el SLD. Un caso especial es cuando $z_i = z_{ij}$, $\forall i\ 1 \leq \mathrm{i} \leq \mathrm{m}$, porque F_j sería igual a $N/0$; en este caso, F_j es asignado con un valor muy alto. Como puede observarse, el tamaño del SLD no interviene en la computación o cálculo del fitness.

Este algoritmo se ejecuta con los siguientes parámetros:

- Tamaño de la pobiación.
- Proporción de individuos que son eliminados en cada nueva generación (non-elite-rate).
- Probabilidad de aplicación de cada operador.
- Máximo número de términos para la primera generación.
- Máximo número de generaciones.

3 EXPERIMENTOS Y RESULTADOS

Los experimentos llevados a cabo pretenden comprobar el funcionamiento del algoritmo desarrollado y

validar el comportamiento de los operadores creados utilizado la misma metodología del algoritmo original publicado en [1], y además se comparan los resultados entre ambos. Las funciones que se ha pretendido aproximar son:

$$F(x,y) = sin(xy) \tag{3}$$

$$F(x,y) = e^{xsin(xy)} \tag{4}$$

$$F(x,y) = \frac{40e^{8(x-0.5)^2+(y-0.5)^2}}{e^{8((x-2)^2+(y-0.7)^2)} + e^{((x-0.7)^2+(y-0.2)^2)}} \tag{5}$$

$$F(x,y) = sin(2\pi\sqrt{x^2+y^2}) \tag{6}$$

Para cada función se ha generado un fichero con 400 ejemplos, donde los datos de entrada X e Y tienen valores igualmente espaciados. Los valores que toma la salida Z, son los resultantes tras aplicar la fórmula en cada caso.

Para crear los conjuntos de entrenamiento y test de los experimentos, se han generado 5 particiones distintas para cada función, sin usar ningún tipo de particionador específico. Para cada partición, fueron elegidos aleatoriamente 320 puntos para ser usados como conjunto de entrenamiento (80% del total de puntos), mientras que los 80 restantes fueron como conjunto de test (20% del total de puntos) para examinar las aptitudes del mejor SLD obtenido.

Los valores utilizados en los parámetros durante las distintas ejecuciones del algoritmo son: 500 para el tamaño de la población, 300 para el número de generaciones, 40 máximo número de términos, 0.1 probabilidad de recombinación, 0.001 probabilidad del mutador de dimensión, 0.01 probabilidad del mutador de precedentes, 0.01 probabilidad del mutador de consecuentes y 0.35 como porcentaje de individuos que no van a pertenecer a la población élite.

A continuación se muestran gráficamente las salidas de cada función original para los conjuntos de entrenamiento y test, así como la mejor solución obtenida durante las cinco ejecuciones del algoritmo en cada problema.

Una vez representadas las funciones que debe ajustar el algoritmo evolutivo, así como la mejor salida producida durante las cinco ejecuciones y la evolución del fitness y el tamaño, a continuación, en la tabla

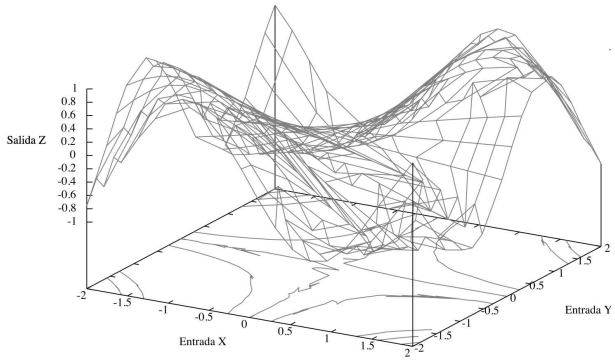

Figura 2: Gráfica Función 1 con salidas reales

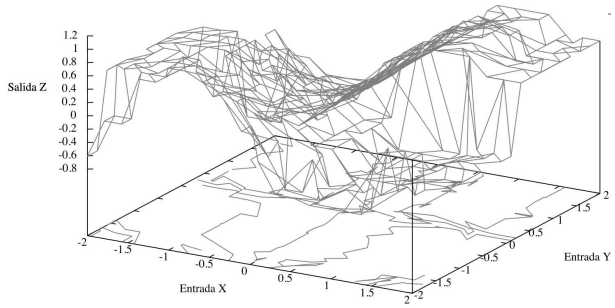

Figura 3: Gráfica Función 1 con salidas obtenidas por EvFuzzySystem

1, se muestra la media y la desviación típica para el fitness final, para el error de generalización y para el tamaño del mejor SLD encontrado para cada ejecución. El error de generalización se calcula como el error cuadrático medio normalizado del mejor individuo obtenido durante la ejecución, en base al conjunto de test. En cuanto al tamaño, o complejidad de las soluciones encontradas, suma de todos los términos de todas las variables más el número de consecuentes

Tabla 1: Resultados del algoritmo implementado para cada función considerada

ID	FITNESS	ECMN	TAMAÑO
1	39.18	0.029	95.4
	(±6.17)	(±0.0049)	(±82.97)
2	124.72	0.013	99.4
	(±37.72)	(±0.0048)	(±39.17)
3	209.3	0.008	99.8
	(±165.23)	(±0.0034)	(±71.32)
4	32.75	0.056	159.2
	(±7.27)	(±0.0152)3	(±95.64)

Por su parte, la tabla 2 presenta los mismos resultados obtenidos en el trabajo orignal.

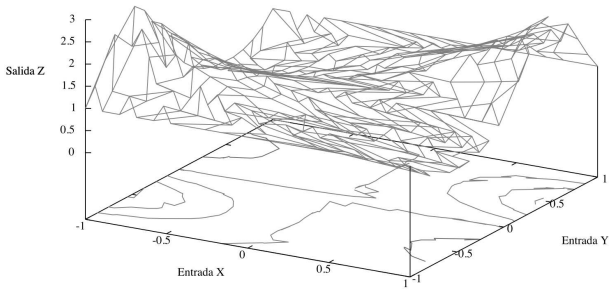

Figura 4: Gráfica Función 2 con salidas reales

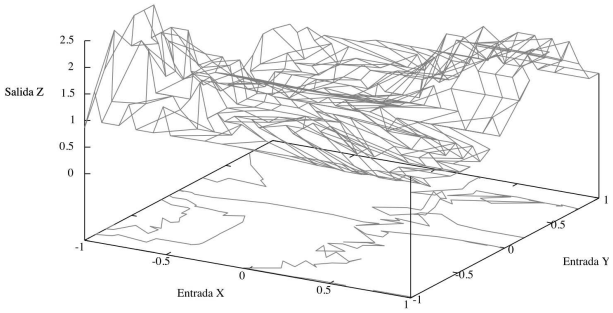

Figura 5: Gráfica Función 2 con salidas obtenidas por EvFuzzySystem

Para cada función de las cuatro con las que hemos trabajado, se ha aplicado el test de t de Student tanto para el fitness obtenido en el algoritmo implementado como en el algoritmo del artículo, así como para el error de generalización y el tamaño de los individuos.

Los resultados han mostrado que, aún cuando los resultados referentes al fitness puedan variar sustancialmente, no hay diferencias significativas en cuanto al error de generalización para las funciones 1 y 4. Por otro lado, el algoritmo original aproxima mejor la función 2, mientras que el nuevo algoritmo descrito ajusta mejor la función 3. En cuanto al tamaño, no existen diferencias significativas para ninguna de las funciones. Como ya se ha mencionado, el tamaño no influye en el cálculo del fitness además de no existir restricción o penalización alguna en cuanto al tamaño para generar los individuos.

Además de las comparaciones entre el algoritmo implementado en este trabajo y el propuesto en [1] para cada función presentada, es necesario comparar con otras técnicas de regresión así como la funcionalidad del mismo para otros conjuntos de datos donde intervienen un número superior a dos variables de entrada, lo cual fue la motivación para la implementación del algoritmo. Para tal fin, se ha utilizado la herramienta de Minería de Datos KEEL [9] que incluye diversos al-

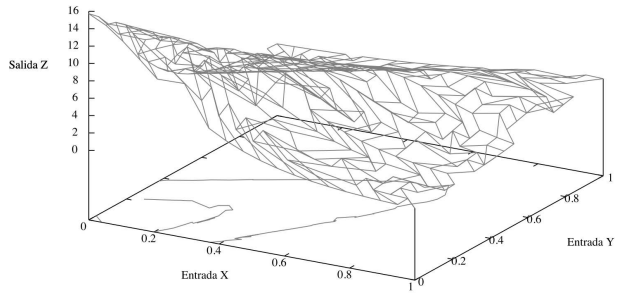

Figura 6: Gráfica Función 3 con salidas reales

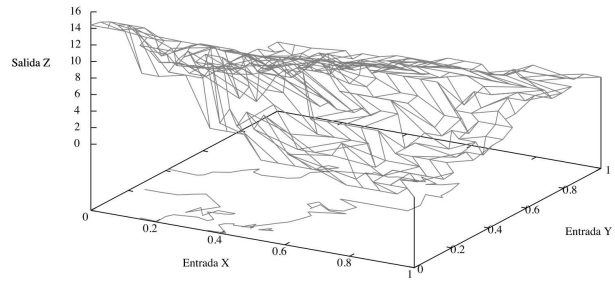

Figura 7: Gráfica Función 3 con salidas obtenidas por EvFuzzySystem

goritmos de regresión así como distintas bases de datos con más de dos variables de entrada.

Los algoritmos seleccionados pertenecen a diferentes familias de las técnicas disponibles de regresión dentro de la herramienta KEEL. Estos son: Fuzzy-Thrift (Aprendizaje de reglas difusas), Regr-NU-SVR (Máquinas de vectores de soporte) y Regr-GAP (Regresión simbólica).

La base de datos utilizada pertenece al problema MachineCPU, dnde intervienen 6 variables de entrada y una de salida. La configuración de los parámetros de EvFuzzySystem para el problema MachineCPU es igual que en el experimento anterior, salvo el número de términos máximo que es 8. Los resultados para la media del Fitness y del Error de Generalización se muestran en la tabla 3.

Los resultados muestran que el algoritmo propuesto funciona para más de dos variables de entrada, además de obtener los mejores resultados de los 4 algoritmos ejecutados con la base de datos MachineCPU.

4 CONCLUSIONES

Los experimentos incluidos, han sido realizados para validar el comportamiento de los operadores creados. Los relativos a las funciones del trabajo orig-

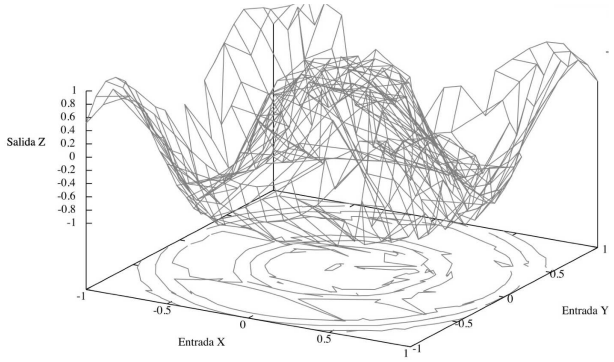

Figura 8: Gráfica Función 4 con salidas reales

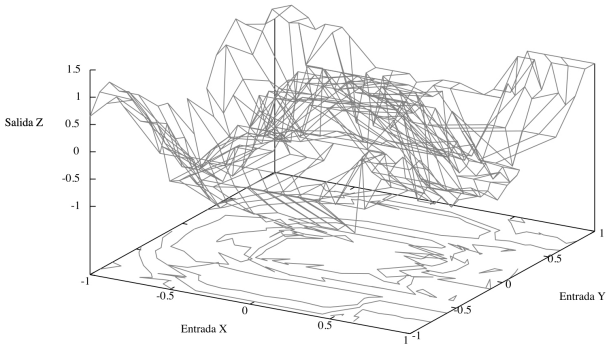

Figura 9: Gráfica Función 4 con salidas obtenidas por EvFuzzySystem

inal [1]demuestran que siguiendo las mismas características del algoritmo propuesto en el artículo, se obtienen resultados con diferencias no significativas en cuanto al tamaño de las soluciones, y de diverso sentido en cuanto al Error de Generalización. Tambin se ha verificado que el algoritmo funciona para más de 2 variables de entrada el cual era el principal objetivo a cumplir en este trabajo.

Agradecimientos

Este trabajo ha sido parcialmente financiado por los proyectos CICYT TIN2005-08386-C05-03, CICE PC06-TIC-02025 de la Junta de Andalucia, y el proyecto UJA_08_16_30 de la Universidad de Jaén.

Referencias

[1] V.M. Rivas, J.J. Merelo, I. Rojas, G. Romero, P.A. Castillo, J. Carpio. Evolving two-dimensional fuzzy systems. *Fuzzy Sets and Systems* 138 Pág. 381-398, 2003.

[2] Chang, Y.O.; Ayyub, B.M. Fuzzy regression methods a comparative assessment. *Fuzzy Sets and Systems* Pág.119, 197-203, 2003.

[3] Cordon, O.; Herrera, F.; Hoffmann, F.; and Magdalena, L.; Genetic Fuzzy Systems. Evolutionary tuning and learning of fuzzy knowledge bases. Advances in Fuzzy Systems: Applications and Theory, World Scientific, 2001.

[4] Eshelman, L.; Schaffer , J. Real-coded genetic algorithms and interval-schema. In L.D. Whitley (Ed.), Foundation of Genetic Algorithms 2, Pág. 187-202. Morgan Kaufmann.

[5] Howard, L. M.; D'Angelo, D.J. "The GA-P: A Genetic Algorithm and Genetic Programming Hybrid". IEEE Expert, junio 1995, Pág. 11-15, 1995.

[6] Karr ,C. Applying genetics to fuzzy logic, AI Expert 6. Pág. 26-33, 1991.

[7] Wright, A. Genetic algorithms for real parameter optimization. In G. J. E. Rawlin (Ed.), Foundations of Genetic Algorithms 1, Pág. 205-218. Morgan Kaumann.

[8] ZADEH, L.A. "Fuzzy sets". Information and Control 8 (3): Pág.338–353, 1965.

[9] KEEL: Knowledge Extraction based on Evolutionary Learning. www.keel.es.

Tabla 2: Resultados del algoritmo propuesto en el artículo.

ID	FITNESS	ECMN	TAMAÑO
1	54 (±8)	0.028 (±0.0007)	34 (±3)
2	197 (±35)	0.006 (±0.003)	41 (±40)
3	114 (±8)	0.012 (±0.001)	18.6 (±1.2)
4	34 (±12)	0.046 (±0.012)	117 (±36)

Tabla 3: Media de resultados obtenidos para el conjunto de datos MachineCPU.

ALGORITMO	FITNESS	ECMN
EvFuzzySystem	181.7 (±113.9)	0.0197 (±0.02)
Regr-UN-SVR	49.43 (±6.02)	0.0204 (±0.02)
Regr-GAP	164 (±32.07)	0.007 (±0.008)
Regr-Thrift	42.84 (±7.04)	0.023 (±0.009)