Evolución de modelos jerárquicos de reglas en problemas anidados y no anidados

Francesc Teixidó-Navarro, Ester Bernadó-Mansilla
GRSI - Grup de Recerca en Sistemes Intel·ligents
Enginyeria i Arquitectura La Salle - Universitat Ramon Llull
Quatre Camins 2, 08022 Barcelona (Spain)
{fteixido,esterb}@salle.url.edu

Resumen

Este artículo estudia el comportamiento del sistema HIDER, que se caracteriza por ser un sistema clasificador incremental que evoluciona un conjunto jerárquico de reglas. Mediante un conjunto de problemas artificiales, se analiza la capacidad del algoritmo incremental para evolucionar representaciones en entornos formados por ejemplos distribuidos de forma anidada y no anidada. A la vez, se estudia el efecto de la función de *fitness* en el conjunto de reglas final. Asimismo, se destaca el interés por el sistema HIDER por ser competitivo respecto a otras aproximaciones basadas en algoritmos evolutivos, ya que explora eficientemente el espacio de búsqueda y evoluciona conjuntos reducidos de reglas.

1. Introducción

Recientemente ha habido un auge del estudio de los sistemas clasificadores basados en algoritmos genéticos. La mayor parte de estos estudios se han centrado en los sistemas clasificadores de tipo Michigan [5], especialmente motivados por los resultados exitosos de sistemas como XCS [9] y UCS [3]. Los sistemas Michigan se caracterizan por evolucionar una única población de reglas. El papel del algoritmo genético (AG) es el de un algoritmo de búsqueda de reglas. Los sistemas obtienen conjuntos de reglas con elevada precisión. No obstante, algunos problemas detectados son el elevado coste computacional y la obtención de conjuntos con gran cantidad de reglas, lo cual dificulta la interpretación del resultado. Por otra parte, los sistemas Pittsburgh [8] evolucionan una población de conjuntos de reglas, suponiendo un coste computacional todavía más elevado. Cada individuo del algoritmo genético codifica un conjunto de reglas que se comporta como una lista ordenada de reglas.

HIDER [2] se caracteriza por ser un sistema iterativo incremental que evoluciona un conjunto jerárquico de reglas. La ventaja de HIDER es un espacio de búsqueda reducido que implica un menor coste computacional. Este sistema ha sido testeado en problemas reales [2] y ha obtenido buenos resultados comparado con sistemas como el C4.5. El objetivo de este estudio es analizar su comportamiento y si los resultados son prometedores, recuperar su uso para posteriormente poder compararlo con otros esquemas actuales como XCS. Creemos que las características de HIDER lo hacen idóneo para problemas con elevado número de ejemplos, donde los sistemas clasificadores clásicos basados en AGs presentarían un coste computacional demasiado elevado. Incluso con problemas de dimensionalidad pequeña, HIDER puede ofrecer conjuntos de reglas más reducidos y por tanto, más fácilmente interpretables que los de un sistema Michigan. Posiblemente los conjuntos de reglas serían similares a los que obtendrían los sistemas Pittsburgh puesto que éstos también evolucionan un conjunto jerárquico de reglas.

Para analizar el comportamiento de HIDER se diseña un conjunto de problemas artificiales

que permiten testear la capacidad expresiva de los conjuntos jerárquicos, así como las características internas del algoritmo. La estructuración del artículo es la siguiente. En la sección 2 se describe brevemente el algoritmo HIDER. A continuación, se comenta el diseño de los problemas artificiales y en la sección 4 se analiza el comportamiento de HIDER para estos problemas. Finalmente, se presentan las conclusiones y las líneas de trabajo futuro.

2. Algoritmo HIDER

HIDER [2], *HIerarchical DEcision Rules*, es un algoritmo de aprendizaje incremental supervisado basado en reglas jerárquicas. Este sistema explora el espacio de búsqueda utilizando un algoritmo evolutivo que produce reglas ordenadas de forma similar a una lista de decisión [7].

El conjunto de reglas es una secuencia ordenada de reglas del tipo *condición* $\rightarrow$ *clase*. La disposición de las reglas en estructura jerárquica implica una dependencia entre ellas. Es decir, en un conjunto de n reglas, un ejemplo e queda clasificado por la regla r_k $(k \leq n)$ cuando satisface la condición de la regla r_k y no satisface la condición de ninguna de las $(k-1)$ reglas anteriores.

```
Proc HIDER(Instancias I) ret CjtReglas
Regla r;
CjtReglas R;
R := 0;
Mientras I conjunto no vacio Hacer
     r := genético(I);
     R := R + r;
     I := I - InstanciasClassif(r);
Fin Mientras
```

Algoritmo 1: Fase de entrenamiento

El algoritmo evoluciona incrementalmente el conjunto de reglas jerárquicas hasta que el espacio de búsqueda queda cubierto (ver algoritmo 1). Dado un conjunto de entrenamiento I, el algoritmo evolutivo devuelve una regla que clasifica los ejemplos de I de la manera más general y precisa posible. La regla evolucionada se archiva en el conjunto final de reglas R y los ejemplos cubiertos por la misma se borran del conjunto I. Este proceso se repite para el conjunto reducido de ejemplos hasta que se satisface la condición de finalización.

R1	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	...	A_{1m-1}	C_1
R2	A_{21}	A_{22}	A_{23}	A_{24}	A_{25}	...	A_{2m-1}	C_2
	...	...	...	...	...	...	...	...
Rn-1	A_{n-11}	A_{n-12}	A_{n-13}	A_{n-14}	A_{n-15}	...	A_{n-1m-1}	C_{n-1}
Rn	A_{n1}	A_{n2}	A_{n3}	A_{n4}	A_{n5}	...	A_{nm-1}	C_n

Figura 1: Conjunto de reglas

A continuación se especifican los detalles de la representación de las reglas y del algoritmo evolutivo.

2.1. Conjunto de reglas

Como se ha comentado previamente, el conjunto de reglas de HIDER sigue una estructura jerárquica. Cada regla es del tipo *condición* $\rightarrow$ *clase*. La condición consiste en un conjunto de tests sobre los atributos del ejemplo, del tipo $(T_1, T_2, \ldots, T_n)$, donde n es el número de atributos. Si el ejemplo satisface estos tests, entonces se clasifica como la clase codificada en la regla.

Para problemas con ejemplos de atributos continuos, cada test T_i se codifica mediante un intervalo $\lfloor l_i..u_i \rfloor$. Dado un ejemplo $e = (a_1, a_2, a_3, \ldots, a_n)$, donde a_i es el atributo $i \leq n$, siendo n el número total de atributos, y una regla $r = (T_1, T_2, \ldots, T_n) \rightarrow C$, el ejemplo satisface la condición de la regla si todos los atributos a_i están dentro del intervalo correspondiente. Es decir $\forall_i : 1 \leq i \leq n : l_i \leq a_i \leq u_i$. De esta forma, la regla definiría una región del espacio de búsqueda mediante un hiperrectángulo.

2.2. Algoritmo evolutivo

HIDER evoluciona una población de individuos, donde cada individuo codifica una única regla. El mejor individuo obtenido después de la evolución será el que contendrá la regla que clasifique de forma más general y precisa el conjunto de entrenamiento I. La función de *fitness* juega un papel básico puesto que define

cómo debe ser la regla, y hasta qué punto se pondera la generalización y la precisión de la misma. El algoritmo evolutivo tiene las fases habituales.

2.2.1. Inicialización

En la fase de inicialización se crea la población de reglas. Cada regla de la población es inicializada utilizando un ejemplo seleccionado aleatoriamente del conjunto de entrada. Cada atributo de la regla inicializada cumple $\forall_i : 1 \leq i \leq n : (l_i = a_i - v_{c1}) \wedge (u_i = a_i + v_{c2})$ siendo v_{c1} y v_{c2} valores del intervalo $[0..Covering]$. Los valores de *Covering* son parámetros de configuración del sistema.

2.2.2. Fase de evaluación

La función de evaluación califica la bondad de las reglas evolucionadas. El algoritmo evolutivo debe evolucionar una única regla que clasifique el conjunto de instancias. De hecho, clasificar todas las instancias mediante una única regla es a veces imposible y da lugar a errores de clasificación. Por tanto, el objetivo del algoritmo es encontrar una regla que clasifique **el mayor número de instancias** del conjunto de entrenamiento de la forma **más precisa** posible. El objetivo es doble y a veces contrapuesto. Si intentamos clasificar muchas instancias a la vez, se incrementa el error de clasificación. Nos encontramos pues con un problema multiobjetivo. La función de *fitness* usada por los autores de HIDER en [2] realiza una ponderación de estos objetivos, tal como se ilustra en la fórmula 1.

$$f(\varphi) = 2(N - CE(\varphi)) + G(\varphi) + Coverage(\varphi) \quad (1)$$

donde
φ Individuo
N Número de ejemplos
CE Error de clase
G Ejemplos bien clasificados

$CE(\varphi)$ es el error de clasificación, medido como el número de ejemplos que están cubiertos por la regla pero cuya clase no coincide con la de la regla. $G(\varphi)$ es el número de ejemplos correctamente clasificados. $Coverage(\varphi)$ es una medida del volumen cubierto por una regla. Concretamente, es la fracción del volumen de la región definida por la regla por el volumen total del espacio de búsqueda. Se define $[l_i..u_i]$ como el intervalo continuo asociado al atributo i de la regla y $[L_i..U_i]$ es el rango de un atributo continuo i. El *coverage* con atributos continuos se calcula con la fórmula siguiente:

$$\begin{aligned} Coverage(\varphi) &= \prod_{i=1}^{m} \frac{Cov(\varphi, i)}{Range(\varphi, i)} \quad (2) \\ Cov(\varphi, i) &= u_i - l_i \\ Range(\varphi, i) &= U_i - L_i \end{aligned}$$

2.2.3. Fase de selección

El proceso de selección se utiliza para seleccionar entre la población los candidatos a formar la población futura [6]. El algoritmo de selección utilizado es por torneo. Estudios recientes demuestran que este algoritmo es más robusto que el algoritmo de la ruleta [1]. Este método propone que un número S de individuos compita. Para ello se seleccionan aleatoriamente S individuos, donde $S \leq P_S$ y P_S es el número total de individuos de la población. Para hacer un muestreo más equitativo de los candidatos a competir se ha elegido el muestreo sin repetición, *Sampling without repetition*. El individuo con mayor evaluación según la función de *fitness* será el seleccionado. Este proceso se repite hasta tener una población final del tamaño deseado.

2.2.4. Operadores genéticos

La mutación se aplica a nivel de gen. En la representación basada en hiperrectángulos. La mutación consiste en sumar o restar un pequeño valor al gen seleccionado, ya sea este un límite superior o inferior.

El proceso de cruce utilizado es $(2, 2)$, es decir, de dos individuos seleccionados como padres r_{p1} y r_{p2} se crean dos nuevos individuos hijos r_{h1} y r_{h2}, los cuales sustituirán a los padres dentro de la población. El operador de cruce está adaptado a individuos que codifican intervalos. Para obtener más detalles, consultar [2].

2.2.5. Fase de recuperación

Es posible que individuos potencialmente buenos se pierdan durante el ciclo evolutivo. La fase de recuperación se encarga de reestablecer los individuos desestimados de poblaciones anteriores. Concretamente, se ha usado *steady state*, el cual recupera un porcentaje de los mejores individuos de la población anterior.

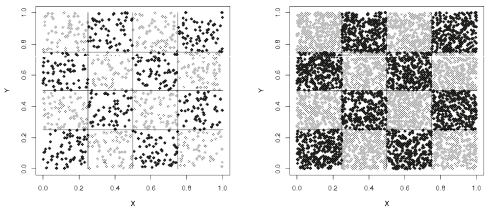
3. Síntesis del entorno

Para estudiar el comportamiento de HIDER se han diseñado varios problemas artificiales. El requisito principal es que los problemas diseñados sean fácilmente representables visualmente para, posteriormente, diagnosticar los resultados obtenidos. Para ello, los problemas tienen únicamente dos atributos continuos y dos clases. Para facilitar el estudio, no se ha incluido ruido y no existen valores desconocidos en los datos. No obstante, reconocemos que sería interesante añadir estas características como trabajo futuro para profundizar el estudio del comportamiento de HIDER.

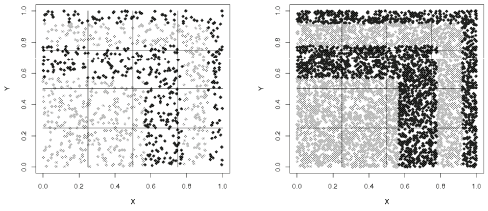
Se han definido dos tipos de problemas: con anidamiento descendente y sin anidamiento. HIDER tiene la particularidad que aprende incrementalmente un conjunto de reglas jerárquicas. Por tanto, es un sistema ideal para aprender problemas en que los ejemplos estén distribuidos de forma anidada. Los problemas diseñados permitirán testear esta capacidad. Asimismo, se han diseñado fronteras de separación entre clases rectas y curvas para evaluar el comportamiento con la representación de hiperrectángulos.

3.1. Problema sin anidamiento

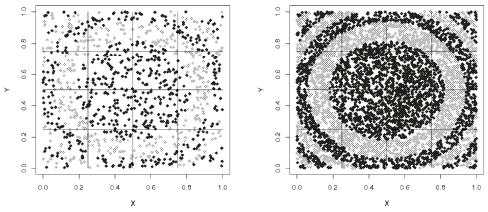
Este problema se caracteriza por tener subconjuntos rectangulares alternados pertenecientes a clases distintas, de forma similar a un tablero de ajedrez (*Checkerboard*). Los ejemplos se distribuyen uniformemente en el espacio de atributos de forma que las dos clases son equiprobables. La figura 2(a) muestra dos conjuntos de entrenamiento, formados por 1024 y 5120 ejemplos respectivamente.



(a) Tablero de ajedrez



(b) Cuadrados anidados



(c) Círculos anidados

Figura 2: Problemas de entrenamiento diseñados (a la izquierda con 1024 instancias y a la derecha con 5120)

3.2. Problemas con anidamiento

Este conjunto de problemas está compuesto por dos problemas de anidamiento distintos: uno de regiones cuadradas (*Nested squares*) y otro de regiones circulares (*Nested circles*) anidadas. Los problemas se han diseñado de manera que el área anidada contiene más ejemplos que la contenedora. Las figuras 2(b) y 2(c) muestran la distribución de los ejemplos de entrenamiento en los dos problemas respectivamente (cada uno con 1024 y 5120 ejemplos).

Iteraciones del genético:	300
Tamaño población:	100
Tamaño selección:	3
Probabilidad cruce:	0.5
Probabilidad mutación individual:	0.2
Probabilidad mutación gen:	0.1
Offset mutación:	0.25
Covering numérico:	0.25
Covering nominal:	0.5
Porcentaje *Steady state*:	0.1

Cuadro 1: Configuración de los parámetros

4. Experimentación

Se ha ejecutado el sistema HIDER con la configuración de parámetros detallada en el cuadro 1. La figura 4 muestra el resultado obtenido por HIDER con los problemas para conjuntos de entrenamiento de 1024 y 5120 instancias. Cada figura representa la clasificación realizada por HIDER de un conjunto intensivo de puntos que están muestreados uniformemente en el espacio de búsqueda.

En los tres problemas se observa que HIDER obtiene mejor rendimiento para conjuntos de entrenamiento más reducidos. Esto significa que HIDER tiene más dificultades para conjuntos de ejemplos mayores, aunque las fronteras de decisión sean las mismas. El motivo es que resulta más difícil evolucionar reglas precisas y a la vez generales, que cubran gran cantidad de ejemplos.

Por lo que se refiere al problema del tablero de ajedrez, HIDER evoluciona casi a la perfección las fronteras de clasificación. La representación con hiperrectángulos es óptima para este tipo de problema. Asimismo, la distribución de ejemplos en cuadros alternos no supone ningún obstáculo para el esquema de aprendizaje incremental de HIDER. El número de reglas evolucionadas para este problema es mínimo (ver cuadro 2).

En el segundo problema el rendimiento de HIDER es prácticamente óptimo (ver la figura 3(b)). Las fronteras de clasificación son correctas y de nuevo, el conjunto de reglas evolucionado es el mínimo posible. En este caso, la distribución de ejemplos se describe muy fácilmente con un conjunto de reglas jerárquico.

Problema	Num. de instancias **1024**	**5120**
Checkerboard, B	15	14
Nested Squares, NS	4	4
Nested Circles, NC	29	46

Cuadro 2: Número de reglas generadas con el sistema normalizado

El algoritmo resulta muy eficiente; es rápido y la solución obtenida es óptima. Si se usara un conjunto de reglas no jerárquico, como por ejemplo una disyunción de reglas, el número de reglas tendría que ser superior. Éste sería el caso de las reglas evolucionadas por sistemas clasificadores como XCS y UCS (del tipo Michigan) que a pesar de ser muy competitivos (en términos de precisión), tienden a obtener conjuntos grandes de reglas.

El tercer problema es el que presenta más dificultades para HIDER, tal como se muestra por las fronteras de clasificación obtenidas (ver figura 3(c)). El motivo de esta dificultad es doble. Por un lado, la representación en hiperrectángulos tiene menos precisión puesto que las fronteras de clasificación son curvas. Esto implica el uso de mayor número de reglas y por tanto, menos generalización de las mismas. Por otro lado, y a raíz del resultado obtenido, se observa que el conjunto de reglas tiende a iniciarse desde los límites del espacio de búsqueda. Las ecuaciones (3)-(5) muestran las matrices de confusión de HIDER para los tres problemas. Cabe notar que el error de clasificación del problema del tablero y de los cuadrados anidados es prácticamente nulo. En cambio, el error en el caso de los círculos anidados es mucho mayor.

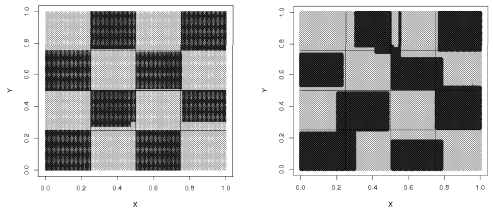
$$CM_B^{1024} = \begin{bmatrix} 535 & 4 \\ 16 & 469 \end{bmatrix} \quad (3)$$

$$CM_{NS}^{1024} = \begin{bmatrix} 591 & 3 \\ 9 & 421 \end{bmatrix} \quad (4)$$

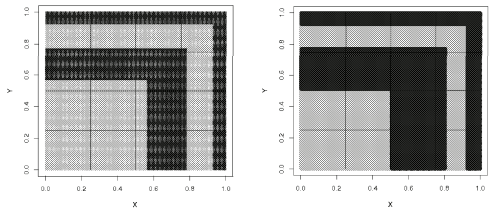
$$CM_{NC}^{1024} = \begin{bmatrix} 295 & 119 \\ 33 & 577 \end{bmatrix} \quad (5)$$

4.1. Sistema con *fitness* normalizado

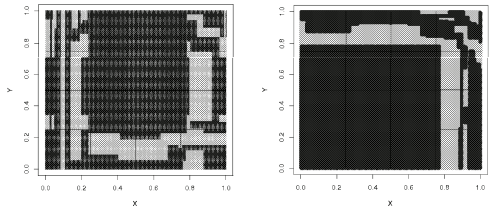
A raíz de los resultados obtenidos, se han estudiado las distintas fases del algoritmo evo-



(a) Tablero



(b) Cuadrados anidados



(c) Círculos anidados

Figura 3: Fronteras de clasificación obtenidas por HIDER con conjuntos de entrenamiento de 1024 instancias (izquierda) y 5120 instancias (derecha).

lutivo. Nuestro estudio se ha centrado principalmente en la función de *fitness*, ya que juega un papel muy importante en el tipo de reglas que se evolucionan. Como se ha comentado anteriormente, el aprendizaje de un conjunto de reglas general y preciso es un problema multiobjetivo complejo. En HIDER, la función de *fitness* simplemente realiza una suma ponderada de los objetivos deseados (ver fórmula 1). Específicamente, la función pondera tres aspectos: 1) el error (número de ejemplos mal clasificados), 2) la bondad (número de ejemplos bien clasificados) y 3) la generalización (medida como la proporción del volumen del espacio de búsqueda cubierto por la regla).

La minimización del error se realiza con el término $2(N - CE(\varphi))$. Por tanto sus valores están comprendidos entre 0 y 2N. La bondad (o precisión) equivale a las instancias bien clasificadas por dicha regla, valor entero del intervalo $[0, N]$. La generalización es la proporción del volumen cubierto y por tanto presenta valores en el intervalo $[0, 1]$.

Se observa que el factor de generalización apenas influye en la función de *fitness*, dado que siempre está comprendido entre 0 y 1. Además, su influencia en la función de *fitness* y la importancia relativa respecto al error y precisión depende del número de ejemplos de entrenamiento N. Por este motivo, y para poder entender más a fondo la contribución de cada factor en el conjunto de reglas final, decidimos normalizar a N el factor de error y de precisión. Con ello, conseguimos que la generalización tenga más importancia en el cómputo final del *fitness*. Además su contribución relativa no depende de N. La fórmula 6 muestra la función de *fitness* normalizada.

$$f(\varphi) = 2 - \frac{2 * CE(\varphi)}{N} + \frac{G(\varphi)}{N} + Coverage(\varphi) \quad (6)$$

La figura 4 muestra el resultado obtenido en los tres problemas. Por brevedad, sólo se muestran los resultados para los conjuntos de entrenamiento de 1024 instancias. Se observa que el rendimiento en este caso es mejor. En el problema del tablero y los círculos anidados, se observan mejores fronteras de clasificación. Asimismo, las ecuaciones (7), (8) y (9) muestran las matrices de confusión con error de clasificación nulo. En cambio, el número de reglas obtenidas es mucho mayor en los tres problemas (ver cuadro 3).

El resultado obtenido es en cierta manera sorprendente. Al aumentar la importancia relativa de la generalización se esperaría, en principio, reglas más generales y por tanto, conjuntos de reglas más compactos. Sin embargo, sucede lo contrario: se obtienen conjuntos más numerosos de reglas menos generales y con mejor precisión. Analizando el comportamiento interno de HIDER en este caso, hemos visto que lo que sucede es lo siguiente. Al aumentar el peso de la cobertura de las reglas en el *fitness*, las reglas tienden a

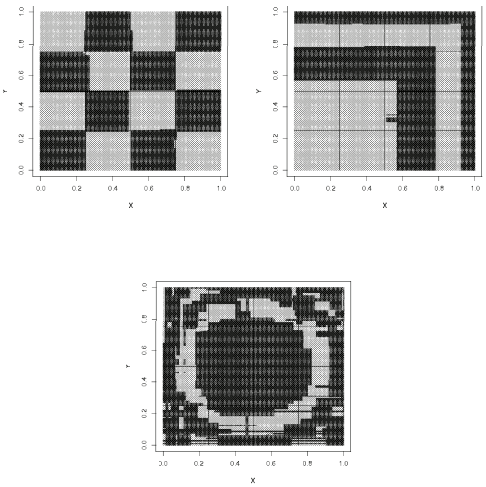


Figura 4: Fronteras de clasificación obtenidas con la función de *fitness* normalizada

ser más generales. Sin embargo, al aumentar la generalización de las reglas vemos que éstas tienden a cubrir ejemplos pertenecientes a las dos clases. Como se penaliza mucho más el error de clase (CE) que los ejemplos bien clasificados (G), una regla general que cubre ejemplos de las dos clases tiende a tener un bajo *fitness*. Por tanto, lo que estamos haciendo es presionando implícitamente hacia reglas más precisas (es decir, con menor error). Esto explica que el error obtenido sea nulo.

Este resultado muestra las relaciones implícitas entre los tres factores de la función de *fitness*. Los tres términos están relacionados de forma que si modificamos el peso de uno de ellos también estamos alterando implícitamente la contribución de los otros. Es por eso que es difícil ajustar la función de *fitness*. Un cambio en la misma produce resultados significativamente distintos tanto en términos de error como en el número de reglas obtenido. Estos resultados nos animan a seguir estudiando la función de *fitness*. Una posible línea de trabajo futuro es analizar a fondo cómo ajustar la función de *fitness* en función de los requisitos esperados. Puede que para determinados problemas interese obtener el mínimo error y para otros que prioricemos la interpretación del conjunto de reglas y por tanto, prefiramos

Problema	Num. de instancias **1024**
Checkerboard, B	30
Nested Squares, NS	27
Nested Circles, NC	114

Cuadro 3: Número de reglas generadas con el sistema normalizado

conjuntos más compactos aunque menos precisos. Además, se podría codificar una función de *fitness* multiobjetivo para que sea el usuario el que decida el compromiso entre precisión y generalización a posteriori (cuando la evolución ha terminado).

Aunque el sistema con *fitness* normalizado obtiene mayor número de reglas, todavía sigue siendo competitivo con respecto a otros sistemas clasificadores basados en computación evolutiva. Por ejemplo, según [4] el sistema XCS obtuvo en el problema del tablero de ajedrez (usando el mismo conjunto de entrenamiento) 45 reglas de clasificación (representadas también mediante hiperrectángulos). En el caso de los cuadrados anidados, XCS obtuvo 43 reglas. En los dos casos el aprendizaje incremental de HIDER obtiene conjuntos más compactos. El motivo es que XCS no aprende de forma jerárquica: dado un conjunto de entrenamiento evoluciona un conjunto de reglas para todos los ejemplos a la vez. Esto aumenta el espacio de búsqueda y tiende a generar muchas más reglas. Además, en los problemas anidados la representación jerárquica es más sencilla.

$$CM_B^{1024} = \begin{bmatrix} 539 & 0 \\ 0 & 485 \end{bmatrix} \quad (7)$$

$$CM_{NS}^{1024} = \begin{bmatrix} 594 & 0 \\ 0 & 430 \end{bmatrix} \quad (8)$$

$$CM_{NC}^{1024} = \begin{bmatrix} 414 & 0 \\ 0 & 610 \end{bmatrix} \quad (9)$$

5. Conclusiones

Este artículo estudia la capacidad de extracción de conocimiento de HIDER el cual aprende incrementalmente un conjunto de reglas jerárquicas. Se ha estudiado el conjunto de reglas obtenido en problemas artificiales

definidos de forma anidada y no anidada. En ambos casos HIDER es capaz de obtener fronteras de clasificación prácticamente óptimas. En general, los problemas anidados facilitan el aprendizaje de HIDER, ya que el conjunto de reglas jerárquico es más natural para describir estos problemas.

El estudio realizado también destaca la sensibilidad del resultado a la función de *fitness*. Pequeñas variaciones en la función de *fitness* pueden dar lugar a resultados significativamente distintos. Además, los factores de error, precisión y generalización de las reglas están estrechamente ligados, lo cual dificulta el ajuste de una función de *fitness* óptima. El uso de algoritmos genéticos multiobjetivo podría evolucionar una serie de reglas con distintos compromisos de generalización, precisión y error. El resultado podría ser más robusto, y permitiría que el usuario seleccionase la regla que mejor se adapta a sus intereses.

En resumen, destacamos que HIDER es un sistema que ofrece ventajas respecto a otros sistemas clasificadores basados en algoritmos genéticos. El aprendizaje incremental jerárquico permite evolucionar conjuntos de reglas más compactos que sistemas del tipo Michigan y con menos recursos computacionales.

Agradecimientos

Los autores agradecen el apoyo de *Enginyeria i Arquitectura La Salle*, Universitat Ramon Llull, y del *Ministerio de Ciencia y Tecnología* con el proyecto TIN2005-08386-C05-04.

Referencias

[1] Orriols-Puig A., Sastry K., P. L. Lanzi, D. E. Goldberg, and Bernadó-Mansilla E. Modeling Selection Pressure in XCS for Proportionate and Tournament Selection. *IlliGAL Rep. 2007004*, 2007.

[2] Jesús S. Aguilar-Ruiz, José Cristóbal Riquelme Santos, and Miguel Toro. Evolutionary learning of hierarchical decision rules. *IEEE Trans. on Systems, Man and Cybernetics, B*, 33(2):324–331, 2003.

[3] Ester Bernadó Mansilla and Josep M. Garrell. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

[4] Ester Bernadó-Mansilla and Tin K. Ho. Domain of Competence of XCS Classifier System in Complexity Measurement Space. *IEEE Trans. on Evolutionary Computation*, 9(1):82–104, 2005.

[5] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.

[6] George F. Luger. *Artificial Intelligence, Structures and Strategies for Complex Problem Solving Fourth edition.* Addison-Wesley, 2002.

[7] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[8] S. F. Smith. Flexible Learning of Problem Solving Heuristics through Adaptive Search. In *8th Int. Joint Conf. on Artificial Intelligence*, pages 422–425, 1983.

[9] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.